On QoS Management in NFV-enabled IoT Platforms

Clovis Anicet Ouedraogo

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *07/07/2021* **par :**

Clovis Anicet OUEDRAOGO

On QoS Management in NFV-enabled IoT Platforms.

JURY

Gladys DIAZ	Maître de conférences HDR	Rapporteuse
Didier DONSEZ	Professeur des universités	Rapporteur
Frederic DESPREZ	Directeur de recherche	Examinateur
Valérie ISSARNY	Directrice de recherche	Examinatrice
Pascale VICAT-BLANC	Directrice de recherche	Examinatrice
Jose AGUILAR	Professeur des universités	Invité
Alain FILIPOWICZ	Ingénieur	Invité
Christophe CHASSOT	Professeur des universités	Directeur de thèse
Samir MEDJIAH	Maître de conférences HDR	Co-directeur de thèse
Khalil DRIRA	Directeur de recherche	Co-directeur de thèse

École doctorale et spécialité :

MITT : Domaine STIC : Informatique et Télécommunications

Unité de Recherche :

Laboratoire d'analyse et d'architecture des systèmes

Directeur(s) de Thèse :

Christophe CHASSOT, *Samir MEDJIAH* et *Khalil DRIRA*

Rapporteurs :

Gladys DIAZ et *Didier DONSEZ*

This page intentionally left blank.

"Once upon a time, I, Chuang Chou, dreamt I was a butterfly, fluttering hither and thither, to all intents and purposes a butterfly. I was conscious only of my happiness as a butterfly, unaware that I was Chou. Soon I awaked, and there I was, veritably myself again. Now I do not know whether I was then a man dreaming I was a butterfly, or whether I am now a butterfly, dreaming I am a man."

Chuang Chou - **Zhuangzi**

Abstract

The Internet of Things (IoT) will have to meet the Quality of Service (QoS) needs of new business applications in various fields such as remote supervision, personal assistance, and transport. The interactions between the software application and the underlying communicating objects will be based on communication networks and middleware (or platform) equipped with new, configurable, programmable, and dynamically deployable functionalities on both physical entities, i.e., pre-existing, but also virtual, i.e., created dynamically according to the need thanks to Cloud Computing. In this new ecosystem, meeting the end-to-end QoS needs of IoT applications remains a significant challenge. The challenge lies both at the intermediary entities belonging to the IoT platform and the level of the IP networks interconnecting these entities. The solutions being proposed are multiplying independently. In this problematic context, the general approach that we consider in this thesis consists of designing, developing, and experimenting with behavioral models for autonomous management of QoS in IoT platforms. This approach i) take advantage of the technological opportunities offered in the Cloud infrastructures (i.e., dynamic deployment of network functions, programmable networks), ii) take advantage of the technological opportunities offered by the dynamic deployment of software components, iii) take into account the de facto heterogeneity solutions deployed, vi) and rely autonomous computing concepts. Following this approach, the three main contributions are made in this thesis. Beyond and in addition to the classic concept of Virtualized Network Function (VNF), we first propose the concept of Application Network Function (ANF), which is based on a less resource-consuming isolation technique (i.e., software isolation technique). ANFs allow the deployment of network functions in resource-constrained environments, typically on end gateways of IoT platforms. They also lead to optimal use of available resources. On this basis and to maintain at the best level the QoS required by IoT applications, we have designed a set of IoT Traffic Control functions (TCF) implemented as VNF and ANF. To achieve optimal deployment of these TCFs, we proposed a second contribution. This contribution consists in the formulation of a multi-objective optimization problem. The proposed and implemented solution considers both the deployment of TCFs and scaling actions, intending to optimize the QoS of IoT applications. The proposed algorithm relies on the bottlenecks (e.g., CPU, RAM) of the platform nodes, initially provided manually by a human administrator. In a third contribution, we then turn to the automated identification of these bottlenecks. To do this, we propose an adaptive identification approach that considers the cost associated with the monitoring of the IoT platform. Indeed, it is not desirable that the overload generated by the monitoring system itself causes QoS problems in the IoT platform. To do this, we model the problem of identifying multiple bottlenecks by a multi-label classification problem. Different supervised learning algorithms are studied to solve this problem. Finally, we propose an algorithm for selecting metrics to monitor in IoT platforms according to the costs they generate.

Keywords: Quality of Service (QoS), Internet of Things (IoT), Network Functions Virtualization (NFV), Genetic Algorithm, Multi-Objective Optimization, Machine Learn-

ing (ML), Performance Bottlenecks Analysis, Fault Localization.

Résumé

L'Internet des objets (IoT) devra répondre aux besoins de qualité de service (QoS) des nouvelles applications métier dans divers domaines tels que la supervision à distance, l'assistance à la personne et le transport. Pour ce faire, les interactions entre les applications IoT et les objets communicants reposent sur des réseaux de communication et plateforme (ou middleware) équipés de nouvelles fonctionnalités configurables, programmables et déployables dynamiquement sur les deux entités physiques, c'est-à-dire préexistantes, mais aussi virtuel, c'est-à-dire créés dynamiquement en fonction du besoin. Dans ce nouvel écosystème, répondre aux besoins QoS de bout en bout des applications IoT reste un défi majeur. Les enjeux se situent à la fois au niveau des entités intermédiaires de la plateforme IoT, et au niveau des réseaux IP interconnectant ces entités, pour lesquels les solutions proposées se multiplient de manière indépendante. Dans ce contexte problématique, l'approche générale que nous considérons dans cette thèse consiste à concevoir, développer et expérimenter des modèles comportementaux pour une gestion autonome de la QoS dans les plateformes IoT: i) en tirant parti des opportunités technologiques offertes dans les infrastructures de type Cloud (par exemple, déploiement dynamique de fonctions réseau, réseaux programmables), ii) en tirant parti des opportunités technologiques offertes par le déploiement dynamique de composants logiciels, iii) et en s'appuyant sur l'Autonomic Computing. Suivant cette approche, les trois contributions principales sont apportées dans cette thèse. Au-delà et en complément du concept classique de fonction de réseau virtualisé (VNF), nous proposons d'abord le concept de fonction de réseau d'application (ANF), qui repose sur une technique d'isolement logiciel (moins consommatrice en ressources). Les ANFs permettent le déploiement de fonctions réseau dans des environnements à ressources limitées, généralement sur les passerelles d'extrémité des plateformes IoT. Ils conduisent également à une utilisation optimale des ressources disponibles. Sur cette base et pour maintenir au meilleur niveau la QoS requise par les applications IoT, nous avons conçu un ensemble de fonctions de contrôle du trafic IoT (TCF) implémentées en tant que VNF et ANF. Pour parvenir à un déploiement optimal de ces TCFs, notre deuxième contribution consiste en la formulation d'un problème d'optimisation multi-objectifs. La solution proposée et mise en œuvre prend en compte à la fois le déploiement des TCFs et les actions de mise à l'échelle, visant à optimiser la QoS des applications IoT. L'algorithme proposé repose sur les goulots d'étranglement (CPU, RAM, etc.) des nœuds de la plateforme, fournis manuellement par un administrateur humain. Dans une troisième contribution, nous nous tournons vers l'identification automatisée de ces goulots d'étranglement. Ainsi, nous proposons une approche d'identification adaptative qui prend en compte le coût associé à la surveillance de la plateforme IoT. En effet, il n'est pas souhaitable que la surcharge générée par le système de surveillance lui-même provoque des problèmes de QoS dans la plateforme IoT. Nous modélisons le problème de l'identification de plusieurs goulots d'étranglement par un problème de classification multi-label. Différents algorithmes d'apprentissage supervisé sont étudiés pour résoudre ce problème. Enfin, nous proposons un algorithme de sélection des métriques à surveiller dans les plateformes IoT en fonction des coûts

qu'elles génèrent.

Mots clés: Quality of Service (QoS), Internet of Things (IoT), Network Functions Virtualization (NFV), Genetic Algorithm, Multi-Objective Optimization, Machine Learning (ML), Performance Bottlenecks Analysis, Fault Localization.

Table of Contents

This page intentionally left blank.

List of Figures

List of Tables

This page intentionally left blank.

List of Acronyms

AE Application Entity
ANF Application Network Function
API Application Programming Interface

CoAP Constrained Application Protocol
CSE Common Service Entity

ETSI European Telecommunications Standards Institute

FCFS First-come, First-served

GA Genetic algorithms

HTTP HyperText Transfer Protocol

IoT Internet of Things
IP Internet Protocol
ITU International Telecommunication Union

LPWAN Low-Power Wide-Area Network
LSL Local Service Level

M2M Machine-to-Machine
MAPE-K Monitor, Analyzer, Planner, Knowledge Base
MBI Multiple Bottlenecks Identification
ML Machine Learning
MLC Multi-Label Classification
MQTT Message Queue Telemetry Transport

NFV Network Function Virtualization
NIP NFV-enabled IoT platform

OS Operating system
OSGi Open Services Gateway initiative

PAN Personal area network

QoS Quality of Service
QoS4NIP QoS for NFV-enabled IoT platform

REST Representational state transfer

SDN Software-Defined Network
SFC Service Function Chains
SOMS Simple Overhead-sensitive Metrics Selection

TCF Traffic Control Functions
TCP Transmission Control Protocol

UDP User Datagram Protocol

VM virtual machine
VNF Virtualized Network Functions

This page intentionally left blank.

CHAPTER 1

Introduction

1.1 Context and Research Scope

The last few years have seen the growing development of devices such as sensors, actuators, and cameras, equipped with communication and computation capacities, in all sectors of activity, both daily (e.g., lighting, temperature, humidity) and professional (such as the remote reading of electric meters or gas meters). The reduction in the costs of devices and the evolution of network technologies, particularly wireless, gave birth to the concept known as Machine-to-Machine (M2M), intending to reduce and even eliminate human intervention in the business processes. The Internet of Things (IoT) based on M2M network infrastructures, therefore, aims to extend the classic Internet to devices other than computers, thus paving the way for new applications like smart factories and smart homes, smart buildings, e-health.

The use of IoT in these contexts is likely to bring real added value from both the consumer and the service producer. Several architectural visions are proposed for the structuring of the IoT. The one we use for our context [oneM2M 2016] is made up of four layers. The first is the *Things layer*, which consists of all IoT devices (i.e., sensors and actuators). This layer is supported by the *Network layer* which includes all the interconnection technologies necessary for the different interactions. The *IoT platform layer* (a.k.a middleware layer) offering an abstraction layer to IoT applications and facilitating, therefore, their interaction with the underlying layers. Finally, the *Application layer*, which consists of all the software applications contributing, via their interactions with the connected devices, to the business activity.

The specificities of IoT lead to the reconsideration of multiple issues already addressed in other more traditional contexts (e.g., Internet Protocol (IP) based networks). We are interested, in this thesis, in the Quality of Service (QoS) problem in IoT platforms. The International Telecommunication Union (ITU) defines QoS as the totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service [Rec 1994].

In the IoT context, QoS refers to the ability of the IoT ecosystem and its different lay-

ers to support the non-functional needs corresponding to the requirements of the business applications. The issue of QoS has been widely addressed for the Internet. Nevertheless, it needs to be reconsidered for the IoT and its applications. Indeed, depending on the business scenario, the IoT applications can have several profiles defined in terms of data types (e.g., binary, text, audio, or image). Applications may also have different kinds of interactions (e.g., request/response, publication/subscription) and QoS needs that can evolve dynamically (i.e., at runtime). Applications can express these needs in terms of End-to-End latency, throughput, availability.

In this new ecosystem, meeting the End-to-End QoS needs of the IoT applications remains a significant challenge. The challenges lie at two-level between the Things layer and the Application layer: the network and the IoT platform layer. In this thesis, we examine the challenges in IoT platforms. Indeed, despite standardization efforts such as OneM2M [oneM2M 2016], the QoS management at this layer still is in its early ages. The existing studies (e.g., [Banouar 2017]) are focused on the structure models[1] of this layer via architectural frameworks and making little contribution to the behavioral models. In addition to the lack of behavioral frameworks for managing the QoS in the IoT platform, the complexity of the problem at this layer is phenomenal. The size and heterogeneity of the platform exacerbate this complexity which very quickly becomes difficult to manage for a human administrator. In this thesis, our proposal covers the behavioral aspect of the management of QoS requirements of IoT applications.

In the following section, we list the limitations of the existing approaches and define our research problem and the research questions.

1.2 Problem Statement and General Approach

Problem Statement The state-of-the-art, detailed in the Section 2.5.5, presents a general vision of works addressing QoS in IoT platforms. However, these works share the following limitations. Firstly, these works fail to address resource scarcity in IoT platforms. Indeed, the availability and capacity of the resources, namely computation, storage, and connectivity, decrease when moving toward Things. Typically, the IoT End Gateways, located close to Things, are small devices with limited computation, storage, and connectivity capabilities. This creates a scarcity of resources at the platform edges that none of the existing approaches address. Secondly, trying to offer a guaranteed QoS to all applications in an IoT platform can only work on a small scale. Still, as the system scales up to billions of devices and applications, it is not easy to track all of the reservations needed for such an approach. Thirdly, the existing solutions remain incomplete (only consider latency, for instance) and lack an overall framework (e.i.

[1] In this manuscript, structure models represent the static aspects of the QoS management framework. It emphasizes the things that must be present in the system being modeled. Conversely, the behavior models represent the dynamic aspect of the QoS management framework. It emphasizes what must happen in the system being modeled.

providing all the basic tools to manage End-to-End resources and traffic). Finally, acknowledging the complexity, heterogeneity, and scale of the IoT platform, there is a glaring lack of cognitive mechanisms to minimize the role of humans in the QoS management process.

Considering these limitations on the QoS management in IoT platforms we define the following research problem.

General definition of the problem: *The problem addressed in this thesis is the need for an approach that can autonomously handle complexity (due to the scale and resource scarcity) of today's IoT platforms and provide End-to-End QoS to IoT Applications.*

General Approach Under this problematic context, the general approach that we consider in this thesis consists in designing, developing, and experimenting with models for autonomous management of QoS in the IoT platform: i) taking advantage of the technological opportunities offered in the Cloud-like infrastructures (i.e., the dynamic deployment of network functions, programmable networks), ii) taking advantage of the technological opportunities offered by the dynamic deployment of software components, iii) and following autonomous computing concepts. Fig. 1.1 illustrates the expected position of our approach regarding the technological opportunities. For the sake of readability, this section briefly introduces each technology according to its application. In turn, the reader can find an in-depth analysis in Chapter 2.

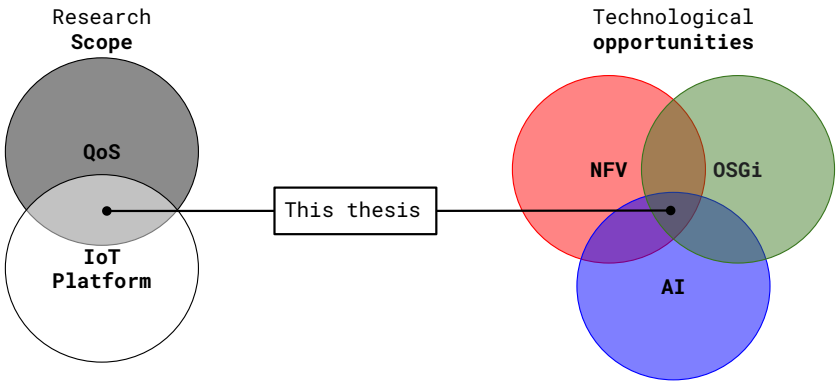


Figure 1.1: Thesis positioning: Taking advantage of technological opportunities

- **Open Services Gateway initiative (OSGi).** This standard is specified and maintained by the OSGi Alliance [Alliance 2018]. The OSGi specification describes a modular system and a service platform for the Java programming language. This modular system implements a complete and dynamic component model that does not exist in standalone Java Virtual Machine environments. Components coming in the form of plugins (or bundles) for deployment can be remotely (without requiring a program reboot) installed, started, stopped, updated, and uninstalled. The modular system life cycle management is implemented via Application Programming Interface (API) that allow for remote

downloading of management policies. A service registry allows bundles to detect the addition of new services or the removal of services and adapt accordingly. OSGi provides flexibility in the management and deployment of software components in already deployed compliant software.

- **Network Function Virtualization (NFV).** This paradigm proposes to overcome the dependence between network functions (e.g., filtering function) and the physical hardware (e.g., switch) on which they are usually deployed. NFV relies for connecting the network functions on another paradigm: The Software-Defined Network (SDN) paradigm. SDN aims to apply a logic of programmability to all elements of a network. For example, a switch can have its packet forwarding logic dynamically programmed rather than applying a predefined, static algorithm. The purpose of NFV is to provide flexibility in the management and deployment of initially operator networks, then more generally communication networks.

- **Artificial Intelligence.** Artificial intelligence enables computers to mimic the perception, learning, problem-solving, and decision-making capabilities of the human mind. One of the problems in building autonomous IoT platforms is the complexity that prevents one from accurately described them by mathematical models. It is, therefore, difficult to control such platforms using such existing methods (e.g., queuing theory). Artificial intelligence-based computational techniques [Choudhury 2016] (i.e., Soft computing) deals with partial truth, uncertainty, and approximation to solve complex problems.

1.3 Research Questions and Main Contributions

To address the research problem, we start by investigating the following research question:

> **RQ-1:** *How to maintain the applications' QoS the closest to their requirements while adapting to the resources' scarcity when moving from Cloud to Things?"*

Following the general approach described above, many advancements in the state-of-the-art are expected to solve the research problem above. To answering **RQ-1**, we made the following contributions.

1. We introduce the ANF concept, which relies on a minimal level of isolation technique dealing with software component execution (e.g., OSGi). The ANFs make possible the deployment of NFs on IoT End Gateways (i.e., the closest Gateways to the Things) and support reaching the best possible use of available heterogeneous resources capacity of the platform. We design a collection of Traffic Control Functions (TCF) that we implement as VNF and ANF, intending to sustain the QoS level required by the IoT applications. We

also provide the performance measurement results to get the quantitative characteristics associated with the different implementation packages (VNFs and ANFs) of the considered TCFs. We study the effects of the traffic arrival rates on the processing time and the resource usage (computation and memory) required for executing the TCFs.

2. To achieve optimal deployment of these TCFs, our second contribution consists of developing and solving a multiobjective optimization problem. The designed scheme, named QoS for NFV-enabled IoT platform (QoS4NIP), considers both TCFs deployment and scaling actions while optimizing for each IoT application its End-to-End QoS. The performance measurement results (obtained above) are used by QoS4NIP to solve the multiobjective optimization problem formulated for an efficient planning scheme of TCFs deployment on the available nodes in NFV-enabled IoT platform (NIP). We evaluate the benefits in terms of the cost-saving of the solutions provided by the QoS4NIP scheme. These benefits are compared to the solutions provided by First-come, First-served (FCFS), the autoscaling scheme, and the two variants of QoS4NIP that do not consider the scaling action but only TCFs (the first considers only TCFs deployed as VNFs, and the second considers TCFs deployed as VNFs and ANFs). We consider a realistic case study dealing with Connected Vehicles for validation of our approach. The validation results show that our scheme, QoS4NIP while sustaining each IoT application End-to-End QoS, achieves the best cost-saving amongst the existing competing approaches. A human manually provided information of the nodes regarding their status in terms of bottlenecks.

The problem of automatically identifying the bottleneck has brought us to the second Research Question (RQ-2):

> **RQ-2:** *"How to determine the metrics that maximize the efficiency of NIP performance analysis and lead to a minimum cost for an allocated monitoring overhead budget?"*

To answering **RQ-2**, we made our third contribution:

3. We model the problem of Multiple Bottlenecks Identification (MBI) in NIPs as a Multi-Label Classification (MLC) problem, and we propose a classification of main categories of bottlenecks in NIPs. We propose an algorithm (Simple Overhead-sensitive Metrics Selection – SOMS) 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 design the testbed to provide a training set that is representative of the real-world situation. We develop different supervised ML algorithms to perform the identification of the bottlenecks. We numerically evaluate these MBI models, using the collected data in terms of Subset accuracy, Coverage Error, Sensitivity, and Specificity. We implemented

the proposed SOMS to find which metrics should be considered for the efficiency of the NIP analysis while optimizing the performance of the MBI model, not to label as positive a negative sample and evaluates its performance. Our numerical results show that 81 metrics give the maximum precision (84%) of the MBI model. Up to 83% can be achieved even with a relatively limited metrics subset of 22 metrics.

1.4 Manuscript Organization

The remainder of this manuscript is organized as follows.

Chapter 2. We aim to provide an overview of the technological landscape in which the work presented in this manuscript was executed. First, we present the IoT paradigm (i.e. its characteristics and enabling technologies). Second, we present the NIP, which aims to decouple the IoT architecture from its current infrastructure. We also discuss the complexity problem of such a platform which can only be solved by a high degree of autonomy. Third, we present the autonomous computing paradigm, which provides a blueprint for constructing autonomous systems. We also present enabling techniques (i.e., computational techniques based on artificial intelligence) that allow the implementation of control loops (e.g., Autonomous Manager). We present why IoT applications need QoS and why it is particularly challenging. Finally, we review the state-of-the-art limitations of current approaches to support QoS for IoT applications.

Chapter 3. Beyond and in addition to the classic concept of Virtualized Network Function (VNF), we first propose the concept of Application Network Function (ANF), which is based on a software-level of isolation technique (e.g., OSGi). ANFs allow the deployment of network functions in resource-constrained environments, typically on End Gateways of IoT platforms. They also lead to optimal use of available resources. On this basis and to maintain, at the closest level possible, the QoS required by IoT applications, we have designed a set of IoT Traffic Control Functions (TCF) implemented as VNF and ANF. Then we study the use of Evolution strategies (ES) to design a planning algorithm. The planning algorithm's goal is to achieve optimal deployment of these TCFs through solving a multiobjective optimization problem. The proposed and implemented planning algorithm (QoS4NIP) takes into account both the deployment of TCFs and scaling actions, to optimize the QoS of IoT applications. The proposed algorithm relies on the bottlenecks (such as CPU, RAM) of the platform nodes, first provided manually by a human administrator.

Chapter 4. We then turn to the automated identification of these bottlenecks. To do this, we propose an adaptive identification approach that considers the cost associated with the monitoring of the IoT platform. Indeed, it is not desirable that the overload generated by the monitoring system itself causes QoS problems in the IoT platform. To do this, we study Machine Learning, especially supervised learning to design the Analyser that solves a multi-label classification problem. Finally, we propose an algorithm for selecting metrics to

monitor in IoT platforms according to the costs they generate.

Chapter 5. We discuss the final remarks of the thesis and present the future work. The Research questions are revisited to discuss the answers we provide and to highlight remaining gaps that are the subject of future investigations. Also, we explain the limits of our proposal, delimiting the appropriate cases of application.

CHAPTER 2

Background and State-of-the-Art

2.1 Introduction

Aiming to provide a clear understanding of the challenges inherent to QoS management in IoT platforms, in this Chapter, we review base concepts and discuss the state-of-the-art. To that end, we present the background knowledge regarding the IoT, NIP, Autonomic Computing, and QoS. We also present the state-of-the-art of approaches to sustain QoS for IoT applications.

2.2 Internet of Things

The term IoT, as many words with marketing value, tends to be used with a wide range of meanings. After defining what IoT stands for in this manuscript, representative IoT enablers are described.

2.2.1 Definition

The term "Internet of Things", was first used by Kevin Ashton [Ashton 2009] in 1999 while connecting the latest scheme of RFID in the supply chain of Procter and Gamble (P&G). A decade later, the word's meaning evolved with the emergence of active computing power ubiquitously deployed in connected devices.

The ITU defined IoT in 2012 [ITU-T 2012], as: "A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies". In this manuscript, we adopt this definition, and the term IoT thus refers to the area of technology and research enabling the deployment of Things networks. To defined the *thing* in "Internet of Things" the ITU proposed: a Thing is "an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks.". Note that this definition is not limited to devices such as temperature sensors or humidity sensors. It also includes services and elements of the environment about which characteristics may be collected or actuated (e.g., a vehicle or a robot).

Following the definition, the IoT implies a particular need for M2M communication architectures and protocols, in particular at the level allowing the interaction between Things and all the IoT applications that need to interact with them. The IoT applications include various kinds of applications, e.g., *intelligent transportation systems*, *smart grid*, *e-health* or

smart home. The intermediate level was originally referred to as a *middleware*[1]. With more and more services (e.g., tools for developers, analytics, advanced security, and privacy) added to this middleware layer, the terminologies shifted toward *platform*, and the IoT platforms were born. We will review this term in more detail in Section 2.2.3.

2.2.2 Enabling technologies

Various communication technologies support the IoT, and here we give an overview of the main ones.

2.2.2.1 Short-range technologies

The concentration of devices in a limited geographical space, potentially indoors, allows using telecommunication technologies like Personal area network (PAN) only able to reach a short-range. If necessary, multiple devices communicating locally at a short range can create a mesh covering a wide area.

- **Bluetooth Low Energy (BLE)** is an extension of the Bluetooth communication technology designed to have a much lower power consumption. BLE is, however, based on the same paradigm as Bluetooth, and only star topologies are allowed, with a central master and some peripheral slaves.
- **Zigbee** is a radio protocol developed by the Zigbee Alliance. Contrary to BLE, Zigbee devices may be organized in a mesh. Zigbee is an open standard and has a diverse ecosystem but generates interoperability issues among supposed devices even based on the same technology. The most popular use case for Zigbee is connected light bulbs.
- **6LowPan** was proposed by the Internet Engineering Task Force (IETF). Deploying 6LowPan devices enables creating a mesh network at the packet level (based on the OSI layered model).

2.2.2.2 Long-range technologies

To implement some use cases such as agriculture, IoT devices must be deployed over large areas, potentially not covered by traditional communication networks. Some technologies called Low-Power Wide-Area Network (LPWAN) have been developed to provide ad-hoc networks that allow long-range and low-power communication.

- **LoRa** is a communication technology that is supported by the LoRa alliance. In the network topology enabled by LoRa: devices communicate over LoRa with gateways

[1]This term borrowed from the distributed applications context meaning was – software that provides services beyond those provided by the Operating system (OS) to enable the various components of a distributed system to communicate and manage data.

connected to "traditional" networks and make the messages available to the user on dedicated servers. When a LoRa device wakes up to send a message, it is briefly possible to send a message to it, enabling bi-directional communication.

- **SigFox** is both a network operator and a communication technology deployed by said operator. Contrary to LoRa, SigFox is tied to an operator: only SigFox may deploy an ad-hoc SigFox network. SigFox is, however, quite similar to LoRa: SigFox devices communicate with SigFox gateways that are connected to the Internet. Therefore, messages produced by SigFox devices are stored on servers to be accessible via a Web interface from the client-side.

2.2.2.3 Application layer protocol

The following application layer protocols are used to retrieve data or control IoT devices from the Internet.

- **HyperText Transfer Protocol (HTTP)** is the protocol at the core of the Web. However, HTTP is based on Transmission Control Protocol (TCP), requiring a permanent connection between the communicating entities during the communication. Establishing such connection is costly, and HTTP is therefore not adapted to all IoT architectures, where more lightweight protocols might be preferred. The notion of Representational state transfer (REST) services is usually associated with the HTTP protocol: a Web server exposes an HTTP interface that is meant to be accessed by a REST client.
- **Constrained Application Protocol (CoAP)**, contrary to HTTP is based on User Datagram Protocol (UDP). CoAP is a protocol specially designed for constrained use cases, with reduced headers and limited packet body. UDP being a datagram-based protocol, the establishment of a connection is not necessary before exchanging messages. CoAP mimics the verbs of HTTP, such as GET or POST, and adds a new verb, OBSERVE, to enable notification of the client when a resource is changed.
- **Message Queue Telemetry Transport (MQTT)** is a publish-subscribe protocol standardized by the OASIS consortium. Messages are published to a broker in topics, and subscribers to a topic are notified on publication. To enable the notification, a connection must be established between the client and the broker: MQTT is based on TCP.

2.2.3 IoT platforms

IoT platforms originated in the form of IoT middleware, which was simple: act as a mediator between the Thing and application layers. Its main tasks included data collection from the devices over different protocols and network topology, remote device configuration and management, and over-the-air firmware updates. To be used in real-life heterogeneous IoT ecosystems, IoT

middleware support integration with almost all connected devices and blend in with third-party applications. The independence from the underlying hardware (devices) and overhanging software (applications) enable a single IoT middleware to manage any connected device in the same straightforward way.

Time passing, IoT middleware evolved into a multi-layer platform that enables straightforward provisioning, management, and automation of connected devices. This platform connects hardware, however diverse, to the Cloud by using flexible connectivity options, security mechanisms, and broad data processing powers. For developers, an IoT platform provides a set of ready-to-use features (e.g., device fleet management) that significantly speed up the development of applications for connected devices and take care of cross-device compatibility.

Thus, an IoT platform has a different meaning depending on its view. It is often referred to as middleware when one talks about connecting remote devices to applications and manages all the interactions between the devices and the applications. It is also known as a cloud-enabled platform or IoT-enabled platform to pinpoint its significant business value, empowering standard devices with cloud-based applications and services.

Commercial IoT platforms (e.g., Google Cloud IoT or Amazon Web Services IoT) additionally introduce a variety of features into the hardware and applications as well. They provide components for frontend and analytics, on-device data processing, and cloud-based deployment. Some of them can handle End-to-End IoT solution implementation from the ground up.

2.2.3.1 OneM2M standard

Most industries are solving their IoT needs on their own. They are addressing specific "vertical" application requirements in isolation from each other despite similar architectures. This created "silo" solutions based on very heterogeneous design, production, data model, and implementation cycle. Such unique solutions often result in vendor-specific hardware. Interoperability is, in general, very limited or non-existent. Development is limited to the system owners who understand the particular API, resulting in high development costs and high costs for support. To overcome this challenge, a consortium with 263 members called OneM2M took a standardization effort. The proposed standard objective is to design a standard that will lead to the development of "horizontal" IoT platforms, that is to say, allowing multiple IoT applications with diverse needs to be sustained while remaining independent of the network and the Things to be connected (Fig. 2.1). This consortium has attracted and actively involved organizations from fields of activity related to M2M such as telemetry, intelligent transport, health, utilities, industrial automation, smart homes.

The oneM2M standard is expected to prevail as the main IoT platform architecture since it enables and facilitates interoperability at different levels. Concretely, the standard is based on the notion of resources following the REST architectural style (with resources in a tree structure) and integrates several communication protocols such as HTTP, CoAP, or MQTT.

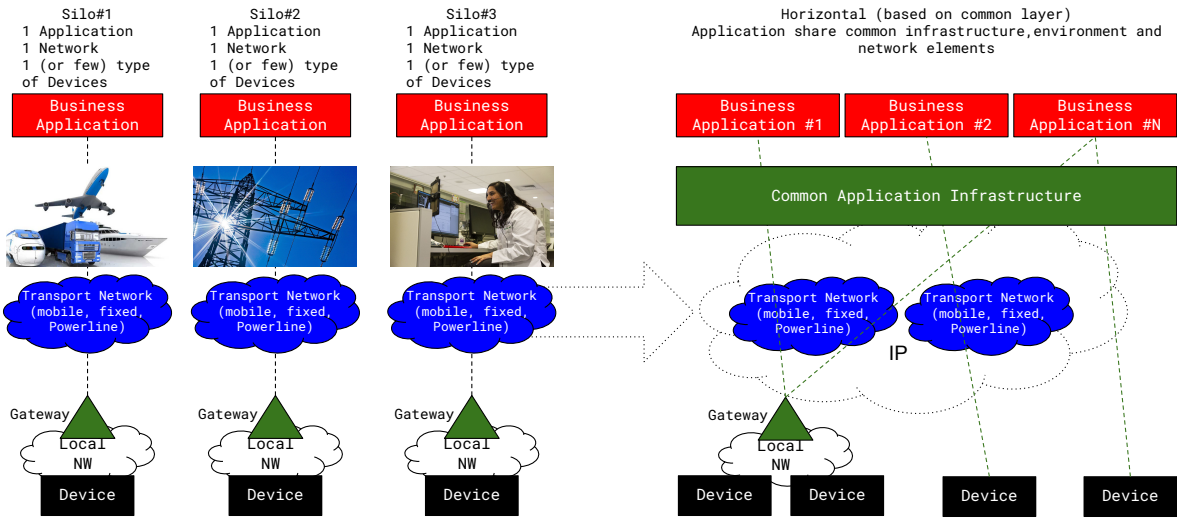


Figure 2.1: IoT challenge: emergence of the common service platform [ETSI 2014]

The oneM2M architecture is made up of four layers.

- The application layer is made up of application entities Application Entity (AE) which represent the applications interacting with the server, gateways, or device.
- The Service layer, called the Common Service Entity (CSE), represents the middleware abstraction layer. There are two types of CSE: Infrastructure Node CSE (IN-CSE) hosted on a Server and Middle Node CSE (MN-CSE) hosted on a Gateway.
- The Network layer encompasses all communication networks.
- The Things layer encompasses all underlying devices.

2.2.3.2 Considered IoT platform: Eclipse OM2M

Eclipse OM2M is an open-source IoT platform compliant with the OneM2M standard developed in JAVA by the LAAS-CNRS. It provides a REST API with open interfaces allowing the development of services and applications independent of the underlying network. Eclipse OM2M platform allows one:

- to support different communication protocols (e.g., HTTP, CoAP, MQTT);
- to interface with remote device management standards (e.g., Open Mobile Alliance Device Management or OMA-DM);
- to integrate existing technologies (e.g., Zigbee, Phidgets).

In this manuscript, we consider an architecture inspired by the vision given by the oneM2M standard. It constitutes our overall infrastructure and is broken down into the following components (Fig. 2.2): IoT applications, IoT Server, IoT Gateways, and devices. The interconnection between applications and the Server, or between the IoT platform entities themselves (server $\Longleftrightarrow$ gateway or gateway $\Longleftrightarrow$ gateway), is assumed to be on IP networks. The interconnection between end Gateways and the device is supposed to rely on short-range technologies (e.g., Zigbee, Bluetooth).

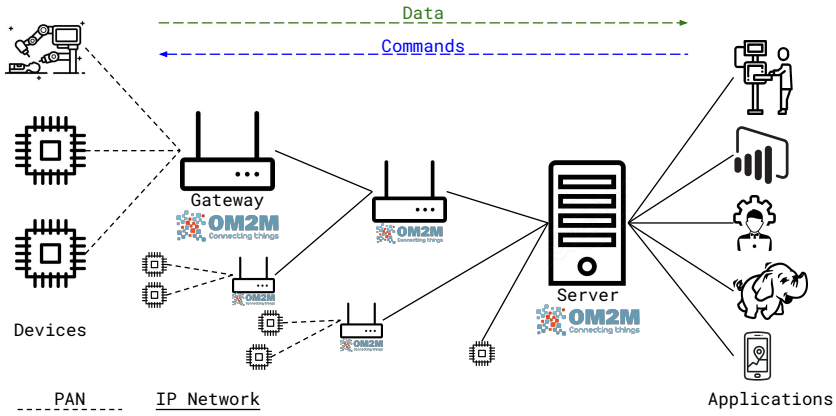


Figure 2.2: OM2M functional architecture

Server A Server is the entry point for communications between applications and devices via a gateway. There is only one Server in an IoT platform and can be deployed on a Cloud and access over the Internet. It implements the IN-CSE of the oneM2M standard.

Gateway A Gateway is the entry point to devices and potentially to other gateways. A gateway acts as a proxy for the devices to interface them with the core network. Therefore, it may be deployed on a physical machine close to the devices, generally limited in resources (e.g., CPU, RAM). Gateways can be attached in sequence and hierarchically up to the Server. The number of gateways varies from one to several. It can be attached to one or more devices and zero or more other gateways. A gateway implements the MN-CSE of the oneM2M standard.

Device An device usually performs metrics capture (e.g., temperature, humidity, heartbeat) and actuation (e.g., camera, motor) operations. Their integration into the IoT platform may require the use of a gateway. The number of devices varies from one to several for each gateway. Each device may be powered with a battery and communicates with the gateway via a specific network technology (see Section 2.2.2.1).

Implementation details The considered IoT platform, Eclipse OM2M is based on a modular architecture implemented through an Equinox OSGi framework, which makes it highly extensible through modules (OSGi plugins) that can be installed during the design or the runtime of the platform. (Fig. 2.3).

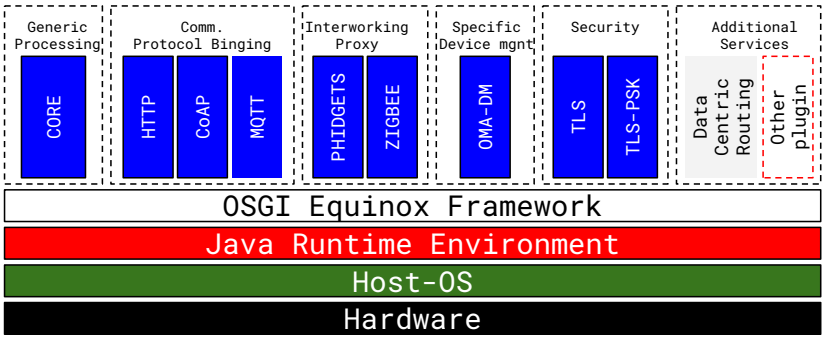


Figure 2.3: OM2M overall internal structure [Alaya 2014]

The Open Services Gateway initiative (OSGi) standard is an open standards organization specified and continues to maintain the OSGi Alliance. The OSGi specification depicts a modular system for the Java programming language that implements a complete and dynamic component model that does not exist in a standalone Java Virtual Machine. In the form of plugins (or bundles) for deployment, applications or components can be remotely installed/ uninstalled, started/ stopped, updated, and without requiring a restart of the modular system. The management of Java packages/classes is specified in detail. Application life cycle management is implemented via APIs that allow remote loading of management policies. A service registry allows bundles to detect the addition of new services or the removal of services and adapt accordingly. The OSGi specifications have evolved beyond the original focus of service gateways. OSGi is used in applications ranging from mobile phones to the open-source Eclipse IDE. Several application areas include automobiles, industrial automation, building automation, PDAs, grid computing, entertainment, fleet management, and application servers.

Each of OM2M plugins (Fig. 2.3) offers specific functionalities, allowing in particular, not only to have an extensible tool but also adaptable, because it is possible thanks to OSGi to start only a certain number of plugins, to stop, to uninstall, or delete others without the need to restart the platform entity.

2.3 NFV-enabled IoT platforms

Today we witness the birth of a NIP that relies on two complementary paradigms, NFV and SDN, to decouple the IoT architecture from its current infrastructure. In the following, we describe these technologies.

2.3.1 Definitions

To understand the NIP some definitions need to be made clear.

Network function Networks are responsible for transporting data from one terminal machine to another terminal machine. To do this, a series of intermediate equipment is often necessary. These devices implement logic that allows them to process the traffic they receive. This logic is called a network function. Traditionally, these functions are performed on *dedicated* equipment, which is designed for this single use. For example, a router is a piece of equipment that implements a Network Function (NF). This NF is generally a routing algorithm that decides the immediate destination of incoming traffic to bring it closer to its final destination.

Virtualization The term virtualization can take several meanings depending on the audience [Kaufmann 1996]. In this manuscript, virtualize means divide/share an entity's resources[2] for multiple users by applying techniques such as time-division multiplexing. For example:

- virtualize a processor consists in distributing the total access time to the processor among several users;
- virtualize a memory consists of logically partitioning this memory for several users;
- virtualize a local network consists (classically) in logically partitioning this local network into several virtual networks (VLANs), then allocating them to users.

The concept of emulation is often confused with virtualization. Although it can be considered a complementary technique, emulation aims to provide users with resources different from those offered by the entity in question. This is possible thanks to an interface (or microcode) that translates the entity's resources into resources for the user (Fig. 2.4). For example:

- emulate a processor consists in proposing a processor with natively non-existent functionalities;
- emulate a memory consists in proposing another type of memory by imitating the behavior of the latter;
- emulate a network (for example satellite) consists in reproducing the behavior of this network in an experimental environment (not satellite).

[2]Anything required for the execution of a program is called a resource. The processor, memory, disk storage, networks are all examples of resources.

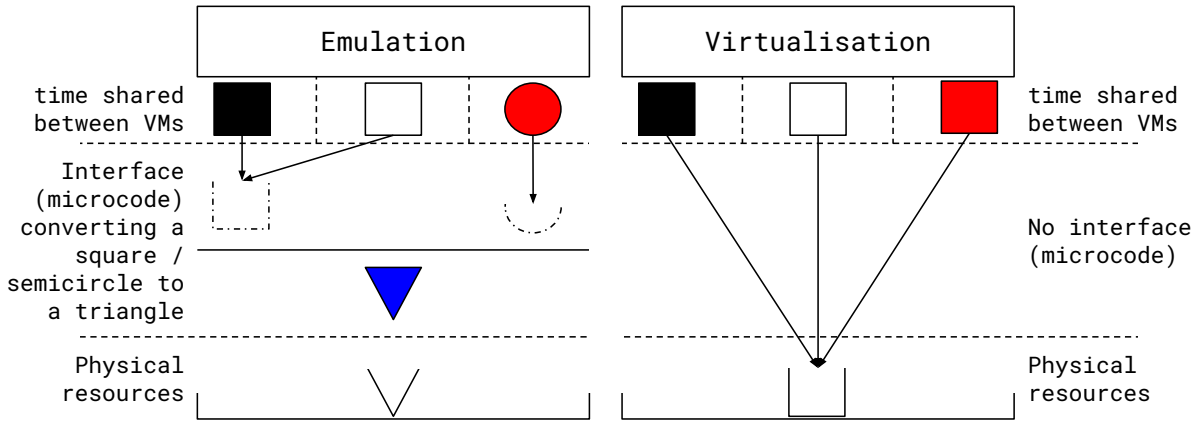


Figure 2.4: Emulation and Virtualisation [Gallard 2008]

Virtualization container Another important term in NFV is that of virtualization container. Suppose for the moment that NFV simply means virtualizing the resources used by network functions. In that case, there are several techniques to achieve this virtualization in practice. Thus, it is recognized that there are two main ways to proceed when it comes to NFV. These two ways are the use of the virtual machine and the use of a container to provide the necessary resources for network functions.

- A virtual machine (VM) is a virtualized computing environment that behaves almost like a physical computer/server. A VM has all the components (processor, memory/storage, interfaces/ports) of a physical computer/server. A VM is generated by a hypervisor (e.g., Kernel-based Virtual Machine or KVM) which partitions its underlying physical resources and allocates a part of them to the managed VMs;

- A container is a virtual environment obtained by limiting and prioritizing the resources allocated to a group of processes (such as CPU, memory, network). A container is generated by a "container engine" such as Docker or Linux Containers (LXC). Note that hypervisors such as Proxmox or Openstack make it possible to generate VMs and containers.

Seen by an application, a container is no different from a virtual machine. However, from the point of view of their structures, these two environments are different. As shown in Fig. 2.5, container groups together an application and its libraries while virtual machine groups together an application, its libraries but also an operating system (called Guest-OS). This difference has two consequences: a virtual machine generally consumes more resources (RAM, CPU, DISK); a container is highly dependent on the underlying operating system (Host-OS).

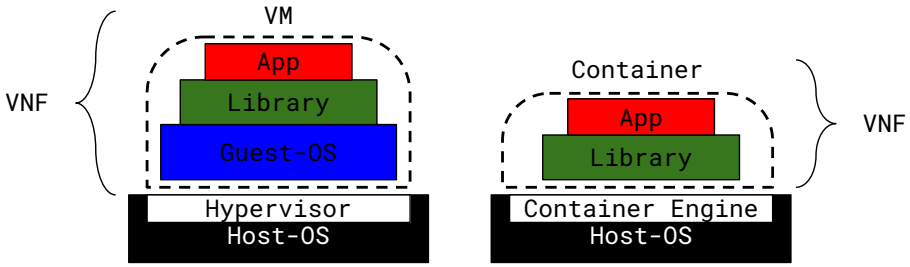


Figure 2.5: Virtual Machine and Container

2.3.2 Enabling technologies

This Section presents two complementary paradigms, NFV and SDN, the purpose of which is to provide flexibility in the management and deployment of initially operator networks, then more generally, communication networks. We focus on the various standards established by international standardization bodies such as the European Telecommunications Standards Institute (ETSI), the Internet Research Task Force (IRTF), and the Open Networking Foundation (ONF). The NFV paradigm proposes to overcome the dependence between network functions (e.g., filtering function) and the physical hardware (e.g., switch) on which they are usually deployed. The SDN paradigm aims to apply a logic of programmability[3] to all elements of a network. For example, a switch can have its packet forwarding logic dynamically programmed rather than applying a predefined, static algorithm.

2.3.2.1 Software-Defined Network

The idea of programmable networks is older than NFV. Already in 1996, and Ipsilon company working group proposed a protocol standard called General Switch Management Protocol (GSMP) for Asynchronous Transfer Mode (ATM) switches [Newman 1998]. This protocol allowed an entity (called a "controller") to establish and terminate connections on a switch; add and remove permissions for point-to-multipoint connections; manage the switch ports, or request configuration information and even statistics. GSMP being limited to ATM technology, in 1998, researchers from the University of Cambridge proposed the Framework Tempest [Van der Merwe 1998]. From the early 2000s, an Internet Engineering Task Force (IETF) working group began working on programmable networks and in 2004 proposed a new architecture called ForCES [Yang 2004] (Forwarding and Control Element Separation). ForCES intends to define a framework and associated protocols to standardize the exchange of information between a control plane and a routing plane. In 2007, at Stanford University, Martin Casado and his team proposed Ethane [Casado 2007], a new architecture for enterprise networks. Ethane enables

[3]In this manuscript, programmability means the capability of a system to accept a new set of instructions that may alter its structure or behavior.

managers to define a single network-wide policy and then apply it directly. Ethane simply combines Ethernet switches with a centralized entity (controller). This controller manages the admission and routing of flows and uses the Ethernet protocol on the switches for routing. In 2008, Nick McKeown and his team proposed an improvement of the Ethane prototype called OpenFlow [McKeown 2008]. Given many unsuccessful attempts in programmable networks, the idea of these researchers was to propose a pragmatic compromise:

- Manufacturers do not need to expose the inner logic of their switches; they have to provide an interface for a protocol (e.g., OpenFlow);
- Network experts and research and development teams will be able to conduct experiments on switches belonging to different manufacturers, such as experimenting with a new routing protocol.

Thanks to this compromise, programmable networks have migrated from laboratories to the industrial world, resulting in laying the first brick of SDN. The primary motivation of the SDN is to allow companies to easily integrate various applications to improve efficiency, reduce the complexity of their network infrastructure, and provide new experiences to their users.

2.3.2.2 Network Function Virtualization

The history of NFV dates back several years. The idea comes from network service providers who have always sought to reduce their production costs. To accelerate the advent of this technology, some of them have come together in an ETSI Industry Specification Group. In October 2012, they published a white paper entitled: "Network Functions Virtualization An Introduction, Benefits, Enablers, Challenges & Call for Action" which introduces and explains the advantages, possible levers, and challenges of the NFV concept [Virtualisation 2012]. At the end of 2012, a discussion on the initial concepts occurred during the ETSI workshop on future networks. After this workshop, the working group expands and proposes an NFV architecture. Along with these efforts of the ETSI group, other groups, especially from academia, have been addressing the issue [Qazi 2014, Shih 2016, Cziva 2017b, Hwang 2015a]. Their thoughts will also be presented in the following sections.

NFV is a concept which aims to allow the implementation of network functions on a virtualized infrastructure such as cloud computing or generic computer hardware [GSNFV 2013]. These functions are intended to be instantiated, configured, moved in various places of the network according to the needs of the operators, thus avoiding the need to install new equipment. The expected benefits include:

- reducing the deployment time of new network services,
- greater automation of network management,

- greater flexibility in terms of the use of network resources,
- cost savings in operation and network hardware investment.

Of course, these benefits must be obtained while maintaining the availability and performance requirements currently recommended in telecommunications networks. Regarding the problem explained in the Chapter 1 (Introduction), intending to ensure End-to-End QoS for the various applications, NFV allows deployment in virtualized environments (e.g., Cloud, Fog) network functions.

NFV is a term that implicitly means to virtualize the resources used by a network function. Thus, virtualizing a network function means virtualize the computing, storage, and network resources used by a network function. This statement is confirmed by the ETSI-NFV working group, which believes that this paradigm does not consist in emulating IT resources for existing network functions but rather in re-implementing these functions to match their deployment environment (Cloud for example). This position can be explained on the one hand by a concern for performance (virtualization being more efficient than emulation [Abramson 2006]) and on the other hand from respect for total independence from the necessary IT resources. Examples of virtualized network functions are listed in Table 2.1.

Niveau dans modèle OSI	Fonction	Produit
2	Switch	OpenVSwitch
3	Router	VyOS
4	Load Balancer	Apache Load Balancer

Table 2.1: Examples of VNFs

In the current NFV infrastructures, NFV provides general networking functions virtualisation tools. SDN orchestrates networking functions for specific purposes, allowing behavior and configuration to be changed and set programmatically. Precisely, when NFV virtualizes the entire infrastructure of a network, SDN centralizes control of the network, creating a network that uses software to build, control, and manage it. Therefore, speaking of NFV, although it is not mentioned, it should be considered that there is an underlying use of SDN. The definition and instantiation of a set of VNF and subsequent “steering” (via SDN) of traffic through them are termed Service Function Chains (SFC).

2.3.3 Need for autonomy: No Silver Bullet!

To quote Frederick P. Brooks, Jr.: “complexity is the business we are in, and complexity is what limits us” [Brooks Jr 1995]. The IoT industry has spent a decade creating an ecosystem of marvelous and ever-increasing complexity. Nevertheless, soon, complexity itself will be the problem. The spiraling cost of managing the increasing complexity of IoT platforms is becoming a significant inhibitor that threatens IoT’s future growth and societal benefits. Managing such

complex systems has grown too costly and prone to error. Managing a myriad of system details is too labor-intensive. People under such pressure make mistakes, increasing the potential of system outages with a concurrent impact on business. Furthermore, testing and tuning complex systems are becoming more difficult. Consider:

- IoT connections will grow 2.4-fold, from 6.1 billion in 2018 to 14.7 billion by 2023. There will be 1.8 IoT connections for each member of the global population by 2023. By 2023, IoT devices will account for 50% of all networked devices (nearly a third will be wireless) [Cisco 2020].

- The rapid growth of data and devices may be outpacing the IT team's capabilities, and manual approaches will not allow keep up. Unfortunately, up to 95% of network changes are still performed manually, resulting in operational costs two to three times the cost of the network [Cisco 2020].

- A significant portion of spending on the IoT (746 billion in 2019 [Ergun 2021]) is associated with maintenance and technical diagnostics due to system failures [Cisco 2020]. Among various system failures, hard failures in hardware, for which the devices age, degrade, and eventually fail, are crucial since they are irrecoverable, requiring maintenance to replace defective parts at high costs.

- Microsoft reports in 2019 and 2020 that complexity and technical challenges are an IoT deal-breaker for 38% of the over $3,000$ decision-makers surveyed. It says that 47% believe there are not enough available skilled workers to build or maintain a network of connected devices [Microsoft 2020].

- For many companies, administrative labor around the IoT service platform life cycle will account for $20-50\%$ of overall operational expenses costs [Jasper 2016].

- SmartThings (the Samsung-owned home platform) experienced 100% of Loss Rate on Monday, March 12, 2018 evening that remained for nearly an entire day for some customers. That is frustrating for people who have SmartThings appliances that rely on the service: door locks, garage doors, lights, and more. This 24 hours incident cost nearly 8 millions[4].

- According to a 2020 survey [ITIC 2020] by the Information Technology Intelligence Consulting (ITIC) on $1,200$ companies of all sizes, respondents most common causes of the worst QoS (i.e., 100% of Loss Rate) included: Human Error (60% of respondents), Software bugs/flaws (40% of respondents), complexity in provisioning/configuring (35% of respondents), Understaffed/Overworked IT Dept. (22% of respondents).

[4]According to Gartner, The average cost of network downtime is around $\$5,600$ per minute. https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

To improve and automate IoT platform operations, installation, dependency management, and performance management to address the above observations, a high degree of autonomy is desired. To achieve this autonomy, high-level decision-making techniques for reasoning in uncertainty must be used. These techniques, if used by humans, can be traced to intelligence. Therefore, one way to achieve a high degree of autonomy is to use high-level decision-making techniques, intelligent methods. In our view, greater autonomy is the goal, and autonomous computing is one way to achieve it. In the Section below, we present the paradigm of autonomous computing.

2.4 Autonomic Computing

The term "autonomic" comes from an analogy to the central nervous system in the human body, which adapts to many situations automatically without any external help. One way to address the problem of managing a complex IT infrastructure is to create IT software and systems that can respond to changes in the IT environment (and, ultimately, the business) so that systems can adapt, heal and protect themselves [Jacob 2004].

2.4.1 Definition

In a report of IBM from 2001 [Horn 2001], Paul Horn describes the growing complexity of the software ecosystem and industry. The development of software requires increasing care to ensure the smooth functioning of such systems. This vision has been discussed in [Kephart 2003] by Kephart et al. They propose an approach based on a living organism that can manage a system and also manage itself. In [Jacob 2004], Autonomic computing is defined as the ability of an IT infrastructure to adapt to change following business policies and objectives. This allows IT professionals to focus on tasks with higher added value, with business rules guiding systems to self-configure, self-repair, self-optimize, and self-protect.

- **Seft-configuration** this feature represents the capability of the system to reconfigure itself depending on the evolution of the monitored system.
- **Self-optimization** the management system needs to optimize itself .
- **Self-healing** when the system has issues, the management system can detect and repair them based on high-level policies.
- **Self-protection** the system can protect itself from malicious attacks and errors that would disable its operation.

Over time, several control loops have been proposed. For instance, the OODA (Observe, Orient, Decide, and Act) loop [Boyd 1987] had been offered by John Boyd (a military strategist) and

applied to the combat operations process. While the Autonomic Computing Monitor, Analyzer, Planner, Knowledge Base (MAPE-K) loop prevails as the oldest and the most popular control loops in IT, recently, ETSI Experiential Networked Intelligence (ENI) control loop was proposed [ETSI 2019]. This loop is inspired by OODA and has four stages (i.e., Sense, Perceive, Learn, and Adapt) that correspond to the Observe, Orient, Decide, and Act stages of the OODA control Loop.

2.4.2 Maturity level

Implementing the MAPE-K loop, is a complex task that requires going through five levels [Jacob 2004]. The five levels, or transition steps, of autonomic maturity are:

1. **Basic** The starting point where most IoT platforms are today, this level represents manual computing in which all platform elements are installed and managed as separate entities. These environments require extensive, highly skilled IT staff who must aggregate and analyze multiple sources of platforms generated data and manage the IT environment from a broad spectrum of individual consoles with multiple interfaces. The highly skilled staff sets up, monitors, and eventually replaces platform elements.

2. **Managed** Supervision techniques and tools are used to collect metrics from the system to detect anomalies, thus helping to reduce the time for collecting and synthesizing information. Human skills are necessary for the analysis of detected anomalies and the execution of corrective actions.

3. **Predictive** At this level, the system monitors and correlates data to recognize patterns and recommend actions that are approved and initiated by IT staff. At the predictive level, the integration of management between several components begins to occur. With the implementation of predictive capabilities, the benefits include the possibility of reducing reliance on excellent skills.

4. **Adaptive** At the adaptive level, not only does the system monitor, correlate and develop action plans, but the system also takes action following established policies. This level allows staff to manage performance against service level objectives. This helps an organization strike a balance between human and system interactions and helps IT infrastructure better handle changing business conditions and improve resiliency.

5. **Autonomic** At the final level, the infrastructure components are well integrated and manage themselves dynamically according to business rules and policies. The autonomous level allows staff to focus on business requirements. Trade policy becomes the primary driver of IT management, and the business benefits from improved agility and resilience.

2.4.3 Architecture

An architecture is proposed to implement an autonomic computing system. Fig. 2.6 shows this architecture, called MAPE-K. The framework is made up of the following components: the Managed Entity and Autonomic Manager.

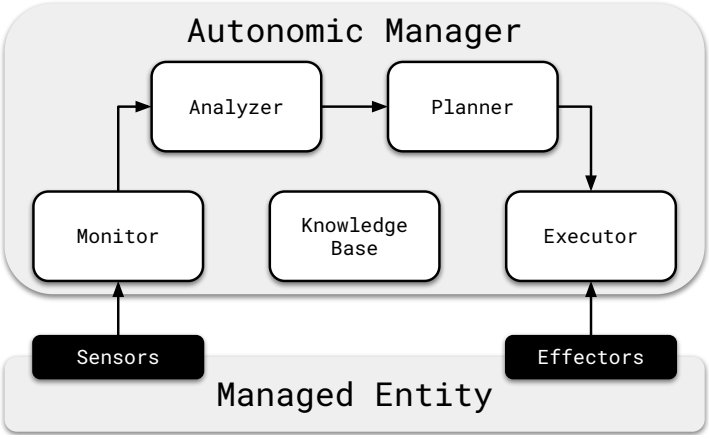


Figure 2.6: MAPE-K loop for Autonomic Computing [Jacob 2004]

Managed Entity The managed entity is the controlled system. The managed resource is a collection of resources, observed and controlled through:

- **Sensors** they represent entities gathering metrics and sending them to the management system.
- **Effectors** these components are in charge of changing the managed system when the autonomic framework detects issues. They perform basic actions on the managed system, following the orders of the management framework.

Autonomic Manager The autonomic manager is a component that implements the control loop. The architecture dissects the loop into five parts. The five parts work together to provide the control loop function.

- **Monitor** this component aggregates the metrics received from the sensors. It has to update the Knowledge Base of the framework when a change is detected.
- **Analyzer** the Analyzer is in charge of finding out the problems in the system. Based on the description of the entities in the system and their current state retrieved by the Monitor. It will infer the Symptoms. This information will send a Request For Change (RFC), a high-level representation of the parameters to change in the system, to the planner.

- **Planner** this component bases its reasoning on the RFC received from the Analyzer. It aims to find a plan of actions to perform on the system to apply the given changes. The choices made by the planner are influenced by the high-level policies defined in the Knowledge Base.

- **Executor** this receives the plan of actions inferred by the planner. It uses this plan to determine the correct actuators to use in the system to perform the actions.

- **Knowledge Base** this component stores the information of the monitored system. It contains a description of the elements of the system, along with their current state. It also possesses high-level policies to apply when a decision has to be taken in the system.

In the sections below, we present our vision of implementing Autonomic Manager for NIP.

2.4.4 Enabling techniques

One of the problems in building such an Autonomic Manager for NIP is the complexity that prevents it from being accurately described by mathematical models and is therefore difficult to control using such existing methods. Soft computing[5] on the other hand, deals with partial truth, uncertainty, and approximation to solve complex problems. To quote Zadeh A Lotfi, who is the pioneer of fuzzy logic: "the guiding principle of soft computing is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, a better rapport with reality" [Zadeh 1993]. Because of its features such as intelligent control, nonlinear programming, optimization, and decision-making support, soft computing has become popular and has drawn research interest from people with different backgrounds [Jang 1997].

It is becoming difficult to control the growing complexity of modern NIP using traditional control systems techniques. For example, many nonlinear and time-variant systems with considerable time delays cannot easily be controlled and stabilized using traditional techniques. One reason for this difficulty is the lack of an accurate model that describes the system. Soft computing is proving to be an efficient way of controlling such complex systems. [Yager 1994] pointed out that soft computing is not a single method, but instead, it is a combination of several techniques, such as fuzzy logic, neural networks, and genetic algorithms. All these methods are not competitive but complement each other and can solve a given problem. It can be said that soft computing aims to solve complex problems by exploiting the imprecision and uncertainty in decision-making processes.

Fig. 2.7 shows the conventional and soft computing-based solution principle. The left diagram shows the traditional hard computing approach where an exact model of the system under investigation is available and traditional mathematical methods are used to solve the

[5]Soft computing is a collection of artificial intelligence-based computational techniques [Choudhury 2016].

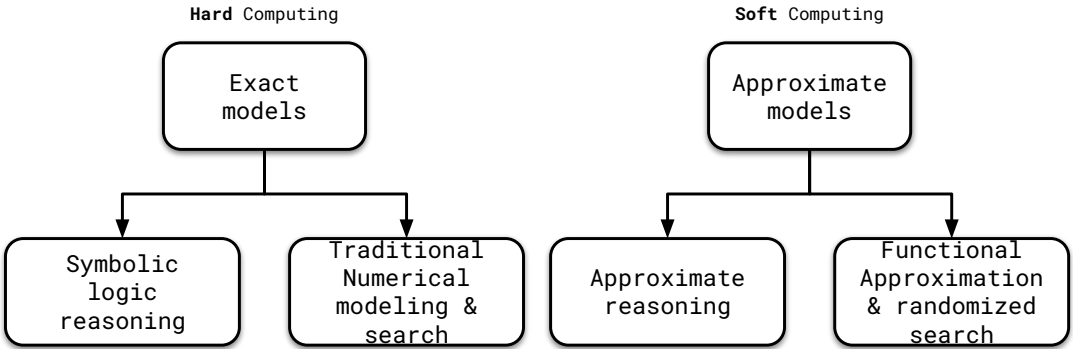


Figure 2.7: Problem-solving approach [Gupta 2013]

problem. The right diagram shows the soft computing approach where only an approximate model of the system may be available, and the solution depends upon approximate reasoning techniques.

One can only build an Autonomic Manager by relying on Soft computing techniques. Below we describe the main artificial intelligence-based computational techniques used in this thesis to cope with such a complex task.

2.4.4.1 Machine Learning

To understand "Machine Learning", one needs to understand what "learning" means in the context of machine learning. A computer program is said to "learn" from experience E for a task T and performance measure P, if its performance P at the task T, improves with experience E. For instance, in "learn to play draughts" for a computer program, the task T is "Play draughts". The performance P is the percentage of games won in a world tournament. The experience E is the opportunity to play against self.

Therefore, ML is the study of computer algorithms that improve automatically through experience. A computer program that learns from experience is called a learning program (a.k.a a learner). The learning process can be divided into four stages: data storage, abstraction, generalization, and evaluation.

1. **Data storage** is the facilities for storing and retrieving huge amounts of data are an important component of the learning process. Computers use hard disk drives, flash memory, random access memory, and similar devices to store data and retrieve data.

2. **Abstraction** is the process of extracting knowledge from stored data. This involves the creation of general concepts on the data as a whole. Knowledge creation is the application of known models and the design of new models. The process of fitting a model to a data set is called training. After the model training is completed, the data is transformed into an abstract form that summarizes the original information.

3. **Generalization** describes the process of transforming knowledge about stored data into a form that can be used for future action. These actions must be performed on tasks similar but not identical to those seen previously. In general, the goal is to discover the properties of the data that will be most relevant for future tasks.

4. **Evaluation** is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then used to measure the improvements in the whole learning process.

In general, ML algorithms can be classified into three types – supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is the ML task of learning a function that maps input to output based on samples of input-output pairs. Each sample in the training set is a pair consisting of an input object (typically a vector) and an output value. In supervised learning, the learner analyzes the training data and produces a function, which can be used for mapping new samples. In the optimal case, the process will correctly determine the output for unseen samples. Both classification and regression problems are supervised learning problems. Numerous supervised learning algorithms are available, each with its strengths and weaknesses.

Unsupervised learning is a type of ML algorithm used to draw inferences from datasets consisting of input data without outputs. In unsupervised learning algorithms, classification or categorization is not included in the observations. There are no output values, so there is no estimation of the functions. The samples given to the learner are not labeled. Therefore the accuracy of the algorithm cannot be assessed. The most popular unsupervised learning method, used for exploratory data analysis to find hidden patterns or groupings in the data, is cluster analysis.

Reinforcement learning is the problem of getting an agent to act in the world to maximize its rewards. A learner is not told what actions to take as in most forms of ML but instead must finds, by testing different actions, which ones generate the most reward. In the most exciting and challenging cases, actions may affect the immediate reward and the following situations and, through that, all subsequent rewards. For instance, consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. One can use a similar method to train computers to do many tasks, such as playing draughts, scheduling jobs in the Cloud, or manage QoS in NIP. Note that reinforcement learning is different from supervised learning. Supervised learning is learning from samples provided by an expert.

2.4.4.2 Evolutionary computation

More than 50 years ago, several innovative researchers at different places in Europe and the United States independently got the idea of mimicking mechanisms of biological evolution to develop robust algorithms for problems of adaptation and optimization. The concept called Evolutionary computation, proposes to utilize the underlying mechanism of natural evolution for optimization problems, resulted in several approaches that have proven their effectiveness and robustness in various applications.

"Evolutionary computation" (EC) is the study of computer algorithms drawing their inspiration from nature. EC uses a form of optimization search. For example, it can start with a population of organisms (the assumptions) and then allow them to mutate and recombine, selecting only those ablest to survive each generation (by refining the assumptions). Such a program is sometimes also referred to as a metaheuristic. The search process involves the same steps:

1. **Initialization** Randomly generate the initial population of individuals.
2. **Evaluation** Evaluate the fitness of each individual in that population with the preferred fitness function.
3. Repeat the following generational steps until a termination condition has been reached (e.g., a solution that satisfies minimum criteria is found):
    a. **Selection** Select the parents (best-fit individuals) for reproduction.
    b. **Variation** Breed new individuals through crossover and random mutation, giving "birth" to the next generation.
    c. **Evaluation** Use the fitness function to gauge the individual fitness of the new individuals
    d. **Recombination** Replace least-fit population with new individuals.

Classical evolutionary algorithms include genetic algorithms, gene expression programming, and genetic programming. Alternatively, distributed research processes can coordinate through swarm intelligence algorithms.

Genetic algorithms (GA) are usually associated with the early work of Holland [Holland 1992], although essentially the same type of algorithm existed much earlier [Fraser 1957]. GA is commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection. In a GA, a population of candidate solutions (aka individuals) to an optimization problem is evolved toward better solutions. Each individual has a set of properties (aka chromosomes or genotype). Chromosomes typically have several fields (called Genes) that might

contain specific sets of values that in turn represent the parameters to be optimized. Mutation in GAs might be as simple as changing a bit in the chromosome or might involve an arbitrarily complicated alteration of one bit into another. Recombination in GAs occurs by selecting and swapping sets of genes from each parent, usually by simply cutting two sequences and exchanging the resulting fragments. Chromosomes are stochastically chosen for replication in the next generation, with a probability distribution that depends directly or indirectly on their fitness values. There are several algorithms to select these parents. The most straightforward strategy, sometimes called "proportional fitness selection", involves scaling the fitness values within a range of zero to one and choosing chromosomes based on those probabilities. The probabilities that determine whether a chromosome will pass to the next generation can be changed. In all cases, mutation and recombination only take place in parents who have been selected for breeding.

Evolution strategies (ES) were invented [Huning 1976] to solve technical optimization problems. Contrary to GA, ES uses problem-dependent representations, and primarily mutation and selection, as search operators. Indeed individuals in ES are described both by "problem-specific variables", which are optimized to solve the target problem, and "strategy parameters", which modify the algorithm's behavior itself. The term "strategy parameter" is given to genes that affect the evolutionary process for a particular individual, usually by specifying a probability distribution or a rate for random processes. A simple strategy parameter might consist of two real numbers representing the mean and standard deviation of the amount by which a gene (a real number) will change when mutated (assuming a normal distribution).

Even if ES looks a lot like Reinforcement Learning, the OpenAI team finds in [Salimans 2017] that ES is faster, easier to implement, and scale in a distributed computational environment does not suffer in case of sparse rewards and has fewer hyper-parameters. ES also discover more diverse solution compared to the traditional Reinforcement learning algorithm.

2.5 Quality of Service

The importance of QoS technologies for computer networks is a constant in the history of networks. Today, QoS is undoubtedly one of the central pieces of the overall computer network technologies. How has QoS come to take such an important place in computer networks? This Section reviews the history of telecommunications network evolution to put this fundamental question underpinning this manuscript in perspective.

2.5.1 Background and Motivation: history repeats itself

Referring to Figure 2.8, there were usually two separate networks in the early days of telecommunications – one for data and one for voice. Each network started with a unique and straight-

forward goal of transporting a particular type of information. The telephone network, which was introduced with Alexander Graham Bell's invention a hundred years ago, was designed to carry voice. The IP network, on the other hand, was designed to move data.

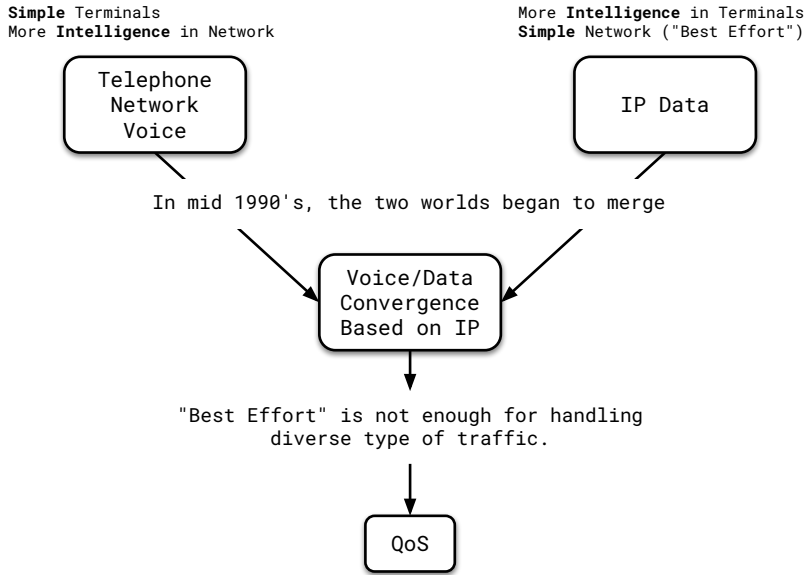


Figure 2.8: Telecommunications network evolution [Park 2004].

In the early times of the telephone network, the terminal was a simple telephone device, a simple analog transducer designed to produce an electric power fluctuating with the speaker's sound pressure. That was all the function the terminal had to perform. In contrast, the network itself was more complex than the terminal. It was endowed with the "intelligence" to provide various types of voice services. A telephone connection is dedicated to one call during the entire period. After the call is terminated, the circuits are used to establish further calls. The circuits used to establish calls are called trunks instead of "loops", which are the lines permanently dedicated to the telephones of individual end-users. In the first telephone network, there were two critical measures of the QoS. The first measure was the probability of call blocking. The probability that a call attempt is blocked due to a lack of an available trunk circuit. The second quality measure was voice quality, once a call attempt was successful and the connection was established. The voice quality depended on the transmission quality of the End-to-End connection during a call, such as transmission loss, circuit noise, echo. The original telephone network was therefore designed with two main objectives. The first was to ensure that enough trunk circuits were provided to make the probability of call blocking reasonable (e.g., 1%). The second was to design the End-to-End network with a transmission plan optimized for voice. Network degradations such as loss, noise, echo, and delay were reasonable. Voice was, and still is, a real-time communications service, and there were no queues in the originating telephone network to store voice signals for later delivery.

The early IP network was a completely different network from the telephone network. First of all, the IP network was designed to carry data. Unlike voice, data was, and still mostly is a non-real-time service. Data are stored in the network and delivered later. When the data was delivered with an error, it could be retransmitted. This service was sometimes referred to as the "store-and-forward". Since the information carried by the IP network was different from that of the telephone network, the design philosophy used for the IP network was also different from that used for the telephone network. In the original IP network, the network was designed to be as simple as possible. The network's primary function was to forward packets from one node to the next. Packets were treated the same way – stored in a single buffer and delivered in a first-in, first-out order. Most of the intelligence was implanted in the terminal device, which was typically a computer. For example, when a packet arrived at its destination with an error, the receiving terminal would send the sending terminal a negative acknowledgment. The packet would retransmit by the sending terminal. The ability to retransmit lost or errored packets was embedded in the terminal. At the same time, the network was unaware of the errored packet. Because the early IP network carried one type of information, "store and forward," non-real-time data, the network was designed to operate in the "best effort" mode. In this mode, all packets are equally treated, and, as a result, the simple design paradigm described above was possible. The main design objective of the IP network was to ensure that the end-user terminal had the requisite intelligence and protocols to ensure reliable data transmission so that the network could remain as simple as possible.

In the mid-1990s, however, the two separate networks started to merge. The word around this time was "voice and data convergence." The idea was to build a single network to carry both voice and data. For more efficient and economical operation, carriers started to plan to consolidate their hodgepodge of separate networks into single "converged" networks. The idea of creating a single converged network for voice and data was no longer an engineering concept. With this convergence, however, a new challenge has arisen. In the converged network, the best effort operation of the earlier IP network is no longer good enough to meet various performance requirements, often conflicting, of various types of information carried by the network. QoS is the technology that provides solutions to this technical problem.

Today IoT platforms find themselves in the same situation – platforms built in silos for particular needs of a specific application. By bringing together these different silos as during the convergence of networks in the 1990s, new approaches should be developed to address the problem of QoS.

2.5.2 Definitions

In this Section, we introduce terms associated with QoS for the understanding of this manuscript. Notwithstanding the long history of discussion, the phrase "quality of service" does not have a universally accepted meaning. The ITU defines QoS as the totality of characteristics of

a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service [Rec 1994]. In this manuscript, QoS is used to describe a set of measurable parameters that can be attached to some identifiable subset of the traffic of IP packets through a given network domain [Carpenter 2002a]. Depending on the reader's context, these characteristics can be defined from two angles: IP network and IoT platform. In Table 2.5.2, the main characteristics [Gozdecki 2003] used in the literature are presented.

IP network point of view	**IoT platform point of view**
Delay: the amount of time it takes a bit (or a packet) to be routed through the network heading to source from a destination	**Latency**: the amount of time it takes a message (e.g. request) to reach the source from a given destination
Jitter: the delay variation over time	**Jitter**: the latency variation over time
Bandwidth: the maximum rate of data transfer across a given path per unit time	**Throughput**: the number of payload messages/bits successfully transferred across a given path per unit time
Bit error rate: the rate of bits/packets with errors that have been transmitted or received per time unit	**Loss Rate**: share of messages not received by the destination per unit time (dropped, lost in transmission or in wrong format)

Table 2.2: QoS from different point of view.

In this manuscript we will adopt a IoT platform point of view using the following characteristics : *Latency*, *Jitter*, *Throughput* and *Loss Rate*.

2.5.3 Historical approaches

Since the problem is as old, it is not surprising that there have been earlier attempts to solve it. The IETF has defined two architecture models: **Int**egrated **Serv**ices and **Diff**erentiated **Serv**ices. The fundamental difference between these architectures is that one (IntServ) was design to *guaranteed QoS* and the other (Diffserv) to *optimized QoS*.

The Integrated Services (IntServ) model is also known as the hard QoS model. It is a model based on traffic flows (i.e., source and destination IP addresses and ports). With the IntServ model, applications ask the network for an explicit reservation per flow. The network devices keep track of all the flows traversing the nodes, checking if new packets belong to an existing flow and enough network resources to accept the packet. By reserving resources on the network for each flow, applications obtain resource guarantees and predictable behavior of the network. IntServ model performs deterministic admission control based on resource requests vs. available resources. The implementation of this model requires IntServ capable routers in the network. It uses Resource Reservation Protocol (RSVP) for End-to-End resource reservation. RSVP enables a host to establish a connection over connectionless IP Internet:

- Applications request some level of service to the network before sending data.
- The network admits or rejects the reservation (per-flow) based on available resources.

- Once cleared, the network expects the application to remain within the requested traffic profile.

The scalability[6] of this model is limited by the fact that exists a high resource consumption on network nodes caused by per-flow processing and associated state. Remember that network nodes need to maintain the reservation state for each flow traversing the node. The fact that RSVP is a soft state protocol with continuous signaling load only aggravates the scalability problem.

The Differentiated Services (Diffserv) model is also known as a soft QoS model. It is a model based on service classes and per-hop behaviors associated with each class. There is no need for an explicit request for resource reservation by applications to the network. Differentiated Services is based on statistical preferences per traffic class. DiffServ allows an end-user to classify packets into different treatment categories or Traffic Classes (TC), each of which will receive a different Per-Hop-Behaviour (PHB) at each hop from the source to the destination. Each network device on the path treats packets according to the locally defined PHB. PHB defines how a node deals with a TC. Network service policies can be specific to an entire QoS domain, some part of a network, or even a single node. DiffServ model implements a statistic, class-based, admission control.

2.5.4 Applications need for QoS: Use Cases

QoS support in IoT platforms is mandatory in a large number of use cases. Each application, however, will be characterized by a different set of QoS need that can vary noticeably among each other. In the following, some IoT use cases that require QoS support to ensure proper operation are presented along with a short characterization of their main requirements.

2.5.4.1 Connected Vehicles

Mobility is a fundamental need in modern society and crucial to economic development. Road-traffic safety and efficiency are the main factors in sustainable transport. Traffic congestion causes substantial economic damage, billions of Euro's in France every year, and the number of vehicles on the road is growing. Each year thousands of people died on roads in the European Union. The number of road accidents and fatalities has decreased, at least in highly developed countries. However, a considerable and sustainable reduction can only be achieved by vehicle communication and coordination. We have already come to rely on vehicle sensors and driver-assistance systems to support us in arriving safely and comfortably at our destination. Through communication, the data exchange among vehicles (V2V communications) and between vehicles and roadside infrastructure (V2I communications), a vehicle turns from an autonomous system

[6]In this manuscript, scalability is the property of a system to handle a growing amount of work by adding resources to the system.

	Advantages	Disadvantages
IntServ	• Good solution for managing flows in small networks. • Intserv enables hosts to request per-flow, quantifiable resources, along end-to-end data paths and to obtain feedback regarding admissibility of these requests.	• Poor scalability. • High resource consumption on the network nodes. • Per flow processing (CPU): signaling & processing load. • Per flow state (memory): to keep track of every flow traversing the node. • Continuous signaling (RSVP is a soft state protocol). • It's very difficult to implement.
DiffServ	• Highly scalable QoS mechanism. • Does not require any resource reservation mechanism on end hosts. • Easy configuration, operation and maintenance. • Support complex traffic classification and conditioning at the edge. • Can aggregate multiple app flows into a limited number of TCs. • Reduced overhead associated to the maintenance of policies on a per flow basis. • Diffserv nodes can process traffic more easily than Intserv devices. • Diffserv is a distributed QoS service model. Resource allocation is distributed among all the routers of a Diffserv domain, allowing for a greater flexibility and efficiency in the routing process.	• Coordination between domains in the QoS end-to-end service. • SPs QoS customization may affect the guaranteed QoS end-to-end service.

Table 2.3: Models for QoS.

into a component of a more efficient cooperative one. The data exchange provides information on a vehicle's vicinity as well as non-visible surroundings. Existing communications systems, such as the radio data system in FM radio, bear high latency and are therefore not suitable for safety applications. Cooperative systems (e.g., WLAN-based V2X communications system) introduced to the market enable direct data exchange among vehicles but do not support all safety applications. Applications for vehicle safety require a very low End-to-End latency of below 10 milliseconds (the time needed for collision-avoidance systems to intervene before a collision occurs). With a bi-directional exchange of data for the negotiation of automatic cooperative-driving maneuvers, a latency of less than 1 millisecond would be needed. In the future, vehicles will detect a highly dynamic object by radar or video, such as a pedestrian, and disseminate this information to neighboring vehicles. In the long term, it is expected that fully automated driving will change individual mobility entirely. Moreover, with small distances between automated vehicles, particularly in platoons or road trains, potentially safety-critical situations need to be detected earlier than human drivers.

2.5.4.2 Smart Manufacturing

The rapid evolution of IoT technologies has recently captured the attention of industrial companies that expect IoT systems to introduce a breakthrough to enhance the efficiency of the manufacturing process. This emerging use case referred to in the literature as Industrial IoT (IIOT for short) represents a significant challenge for IoT platforms. In a typical IIOT ecosystem, sensors and actuators are deployed in a dedicated network inside a factory plant to collect specific data and to assist and control the production process. However, the architecture of an IIOT system is not different from a standard IoT system, the requirements that many industrial manufacturing processes demand represent the main challenge. Stringent QoS requirements in terms of loss rate and latency are mandatory to ensure proper implementation of manufacture automation [Chen 2015]. An example of smart manufacturing applications is a closed-loop control for non-critical processes. In this case, the application requires that the telemetry data and the control commands, from sensors and actuators deployed in the assembly line, respectively, be strictly delivered with latencies of 10 milliseconds [Pister 2009]. When a higher latency is experienced, the whole system enters into an emergency shutdown state, which might cause substantial financial repercussions. Even more stringent requirements characterize emergency signals produced by sensors. They must be dispatched to a powerful central controller with the lowest latency possible. Furthermore, the loss rate of communication is critical since a packet loss may result in products with defects. Enforcement of applications' QoS requirements is more challenging since it is limited to time-related parameters and involves different aspects, such as loss rate.

2.5.4.3 eHealth

Latest technology developments will enable new frontiers for the IoT. Among them, smart health, or eHealth, is a good use case expected to improve our lives significantly by providing new healthcare services such as remote patient monitoring. In this context, QoS are a key requirement [Gama 2008]. In remote health monitoring, for example, through a body sensor network (BSN), the collected data have different relevance, e.g., heart activity data are more important than data on the body temperature. For this reason, the collection and delivery of data must be prioritized accordingly through different QoS requirements. Also, data priority can dynamically change over time depending on the sensor value. To this aim, an IEEE working group has defined QoS requirements for several health applications. For instance, the application that is characterized by the most stringent QoS requirements is electrocardiogram (ECG) monitoring. Such application requires sending bursts of 4 kbit/s of data that must be delivered within a maximum latency of 500 milliseconds for each electrode [Chevrollier 2005].

2.5.5 State-of-the-art

Commonly, IoT platforms have been used as an intermediate level that enriches the data collected from the remote sensors and consumed by the business-level applications. In response to interoperability and vertical fragmentation problem in the IoT, standardization efforts have been made to provide horizontal service platform architectures with common services for applications and devices. However, these standardized service platform architectures (oneM2M, LwM2M, OCF/Iotivity, for example) do not offer practical solutions for QoS management at the platform level. These platforms consider the QoS as the result of the underlying networks [oneM2M 2016, Alliance 2014].

The IoT community has followed two main tracks in its research to improve the situation.

- A first approach is based on the assumption that finding a global solution to the problem in the actual IoT platform is not feasible. This approach consists of trying to optimize the use of the resources available from the platform at a given time. In this context, no latency or throughput guarantees can be obtained, but still, improvements can be achieved. For example, a service differentiation approach (like DiffServ presented in Section 2.5.3) may prioritize and maintain QoS for the most strict applications while offering the best effort to non-sensitive applications. Studies in [Abdullah 2013, Ezdiani 2015, Nastic 2016, Pizzolli 2016, Agirre 2016, Khazaei 2017, Guevara 2017, Santos 2017] adopted such an approach regarding the QoS requirements of applications. Let us recall that all of these studies shared the same advantages and disadvantages of a DiffServ approach (see Table 2.5.3)

- A second family of approaches consists of looking for ways to provide guarantees to users' service from the IoT platform. For example, with an explicit reservation/allocation approach (like IntServ presented in Section 2.5.3), every application that requires some kind of QoS guarantee has to make an individual reservation. That reservation may be granted if the required resources are available. All other applications for which no reservation is made will be served with "Best Effort". Studies in [Tariq 2014, Kim 2017, Bhowmik 2017, Petrov 2018, Mendiboure 2019, Kharb 2019, Shi 2020a, Shi 2020b] rely mainly on the integration of SDN as a enable to implement an IntServ-like approach in IoT platform for QoS management. We recall that all of these studies shared the same advantages and disadvantages of an IntServ approach (see Table 2.5.3)

We group these multiple specific solutions (non-standardized) based on the "track" in to provide QoS to applications (see Fig. 2.9).

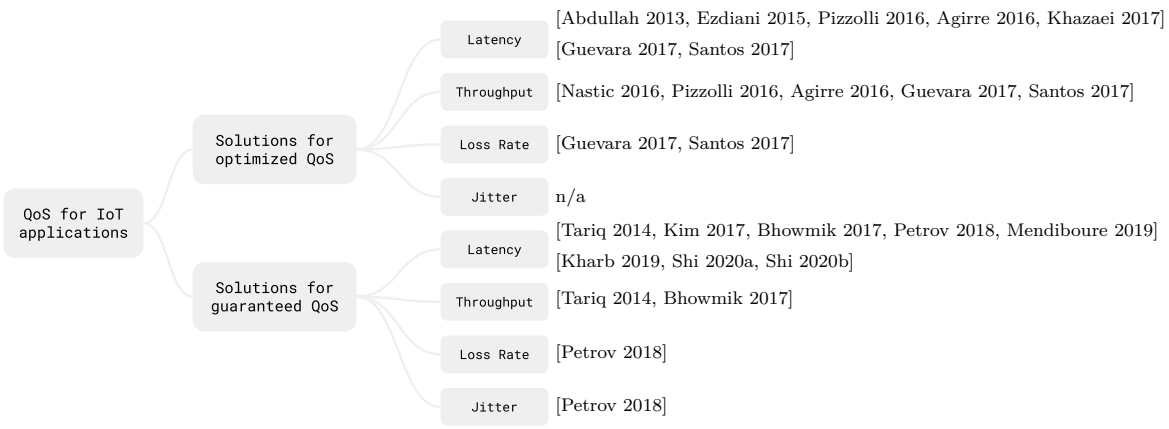


Figure 2.9: State-of-the-art taxonomy.

2.5.5.1 Solutions for optimized QoS

Authors in [Abdullah 2013] propose a message scheduling for IoT platforms to differ emergency messages from non-mission-critical messages. Messages are classified into high priority and best effort. In [Ezdiani 2015], authors present a service differentiation framework based on a scheduling algorithm. Data captured by the devices are assigned different priorities as per the desired QoS requirement by the packet. In [Nastic 2016] Nastic et al. introduce a novel middleware, which provides comprehensive support for multi-level provisioning of IoT Cloud systems. The main features of this IoT platform include i) a generic, lightweight resource abstraction mechanism, which enables application-specific customization of Edge devices; ii) support for automated provisioning of Edge resources and application components in a logically centralized manner, via dynamically managed APIs; and iii) flexible provisioning models that enable self-service, on-demand consumption of the Edge resources. Pizzolli et al. [Pizzolli 2016] introduce Cloud4IoT, a platform offering automatic deployment, orchestration, and dynamic configuration of IoT software components. This platform support data-intensive applications with data processing and analytics and enable plug-and-play integration of new sensor objects and dynamic workload scalability. Overall, the platform is designed to support systems where IoT-based and data-intensive applications may pose specific requirements for low latency, throughput, or data locality. Authors in [Agirre 2016] presented a QoS-based service reconfiguration method to compose the component-based distributed services in IoT platforms. This method provides four functionalities, including monitoring resources, monitoring component-based applications, flexible APIs, and composing a QoS-based mechanism. The registry service status achieves the infrastructure resource management at the service selection level. The monitoring of component-based applications specifies the QoS characteristics to compose a service. In [Khazaei 2017], authors propose and evaluate a hierarchical, programmable, and autonomic IoT platform based on the micro-service models. The proposed platform supports big data processing. The

autonomic management system ensures the overall QoS and optimized resource utilization. Guevara et al. [Guevara 2017] present a classification of services according to application QoS requirements. This is expected to facilitate the decision-making process for the fog scheduler and specifically to identify the timescale and location of resources. Moreover, [Guevara 2017] introduces a mapping between the presented classes of service and the processing layers of the Fog computing reference architecture. Finally, Santos et al. [Santos 2017] propose a model for the resource provisioning in IoT platform dedicated to Smart Cities. The model is executed iteratively to optimizes multiple objectives (such as latency, service migrations, and energy efficiency) and considers cloud-based application QoS requirements and characteristics coming from the wireless network.

2.5.5.2 Solutions for guaranteed QoS

Authors in [Tariq 2014, Bhowmik 2017] presented a publish/subscribe middleware. The proposal relies on SDN technology. The proposed middleware offers an application-aware control capable of enhancing the responsiveness of the control plane while ensuring consistent changes to the data plane with low synchronization overhead even in the presence of network failures. Exploring a more narrow use case (i.e., Connected Vehicle), [Mendiboure 2019] presented a location-aware Pub/Sub middleware with mobility management as additional functionality. The proposed middleware relies on the Openflow protocol and the SDN. To do so, this middleware enabling an efficient SDN-based QoS-aware geographic data dissemination. Adding NFV to SDN, [Petrov 2018] introduces a softwarized 5G architecture for applications with a focus on End-to-End loss rate. [Petrov 2018] presented a mathematical framework to model and quantified the process of applications with strict QoS requirements and the corresponding impact on other applications (i.e., with easy-going requirements). A common drawback of most existing publish/subscribe systems is their dependence on the application layer mechanisms to optimize the publish/subscribe operations. For instance, event routing on a broker network that is organized oblivious to the underlying physical network may result in higher throughput utilization, and higher End-to-End latency since multiple logical links in the broker network may share the same physical links.

2.5.5.3 Current Limitations and positioning

Satisfying the various QoS of the applications is crucial to ensure their optimal operation. The current state regarding this challenge is presented in Table 2.4. This table links the limitations of each of the solutions studied in the taxonomy presented in Fig. 2.9. In this table, the Valid. (Validation) column indicates the method used to assess the validity the proposal. Therefore in this column, “P” means a prototype is used; “S” means the proposal is evaluated in a simulation; and “None” means no validation was conducted.

	Work	Main Contribution	Limit	Valid.
Solutions for optimized QoS	[Abdullah 2013]	QoS-aware message scheduling algorithm	• Deals with only a fixed number (two) types of traffic classes • Do not consider different kinds of scheduling strategies • Only consider Latency	S
	[Ezdiani 2015]	IoT system architecture that handles interoperability	• Very poor reactivity (Reconfiguration of the system is executed every hour according to the result of historic data performance analysis) • Only consider Latency	S
	[Nastic 2016]	Middleware that supports the provisioning of IoT cloud systems	• Do not support post-deployment resource management • Only consider Throughput	P
	[Pizzolli 2016]	PaaS with automatic deployment, orchestration, and reconfiguration capabilities	• Orchestration and reconfiguration is performed by employing a simple threshold-based mechanism • Only consider Throughput and Latency	None
	[Agirre 2016]	QoS-based reconfiguration method for publish/subscribe middleware	• Only consider application-level reconfiguration actions • Only consider Throughput and Latency	P
	[Khazaei 2017]	Autonomic microservice-based IoT platform	• Use a simple threshold-based algorithm with a predefined static threshold (manually tune) taking into account CPU, memory, and network utilization. • Only consider horizontal scaling action • Only consider Latency	P
	[Guevara 2017]	Classification of services according to their QoS requirements	• Only considered Fog-based IoT platforms	None
	[Santos 2017]	Resource provisioning model for the IoT service placement problem in Smart Cities	• Poor scalability due to the long execution time of the used optimization method (integer linear programming). This may work only for relatively static service demands.	S
Solutions for guaranteed QoS	[Tariq 2014, Bhowmik 2017]	SDN-based publish/subscribe middleware	• Non-scalable due to the centralization of the SDN controller • Poor efficiency in bandwidth usage • Unable to react to overload situation in the presence of a dynamic workload • Only consider Throughput and Latency	P
	[Kim 2017]	VNF placement algorithm	• Do not support post-deployment reconfiguration • Only consider Latency	S
	[Petrov 2018]	Softwarized 5G architecture for end-to-end reliability of mission-critical traffic	• Deals with only a fixed number (two) types of traffic classes • Focus on the radio access network (RAN) • Does not consider Throughput	P
	[Mendiboure 2019]	Location-aware SDN-based Pub/-Sub middleware	• Non-scalable due to the centralization of the SDN controller • High computational and communication cost • Only consider Latency	S
	[Kharb 2019]	Fuzzy-based scheduling technique for support service differentiation	• Focus on the LPWAN communications • Only consider Latency	S
	[Shi 2020b, Shi 2020a]	SDN-based publish/subscribe middleware with multiple SDN domains	• Poor efficiency in bandwidth usage and limited scalability • QoS is only guaranteed at the local level (within an SDN controller domain) • Only consider Latency	S

Table 2.4: State-of-the-art limitations

All the limitations of the literature can be summarized as follows.

Poor scalability of solutions for guaranteed QoS IntServ-like solutions provide poor scalability and are only suitable for small networks. Considering that there will be 1.8 IoT devices for each member of the global population by 2023 (14.7 billion devices) and that IoT devices will account for 50% of all networked devices, such an approach is not conceivable.

Incompleteness Current solutions are generally not configurable and provide only a tiny subset of the service differentiation mechanisms needed for control and management of IoT applications traffic. Indeed, to be able to manage traffic in IoT platforms, we must first identify the (group of) targeted traffic (with Classification and Marking mechanisms) and then differentiate the services (with Dropping, Shaping, Scheduling, or Redirecting mechanisms).

Missing to address lack of resources in the IoT When service differentiation mechanisms are proposed, authors always assume the mechanisms already deployed in the IoT platform missing; therefore, an essential characteristic of IoT gateways: "resource is tight". To deal efficiently with the IoT platform resources, the on-the-fly deployment (i.e., when needed) of these mechanisms is essential.

Lack of an overall framework It is necessary to develop an overall QoS management framework to build upon and reconcile the existing scaling (out/in and up/down) mechanisms (for gateways and Server deployed in a virtualized environment) with service differentiation mechanisms (mentioned above). Typically a framework allowing to differentiate the service offered by the IoT platform and adding (or removing) resources (e.g., Computation, Memory, Network) when needed.

Lack cognitive mechanisms The literature lack mechanisms to minimize the role of humans in the control loop and overcome the limitations of manual administration related to the complexity, heterogeneity, and scale of the IoT platform.

Given all these limitations, in this thesis, our scientific positioning is centered on:

1. sustaining End-to-End QoS to IoT Applications in today's IoT platforms with a Diffserv-like approach,
2. handle resource scarcity with a less resource consuming way of deploying network function, and
3. handle the complexity in the QoS management (due to the scale, resource scarcity, and technology heterogeneity) with autonomic computing models.

2.6 Conclusion

This chapter aimed at giving an overview of the technological landscape in which the work presented in this manuscript has been executed.

First, the IoT paradigm has been introduced as the networking Things. One of the characteristic features of IoT ecosystems is heterogeneous technologies calling for a standardized platform.

Second, we present the NIP, which aims to decouple the IoT architecture from its current infrastructure. NIP relay on two complementary paradigms, NFV and SDN, the purpose of which is to provide flexibility in the management and deployment of initially operator networks, then more generally communication networks. Such a platform is so complex that a high degree of autonomy to overcome several challenges.

Third, we present the Autonomous Computing paradigm that provides a blueprint for building autonomous systems. In our view, greater autonomy is the goal, and autonomous computing is one way to achieve it. We also present enabling techniques (i.e., artificial intelligence-based computational techniques) to implements the control loop (i.e., the autonomic manager).

Finally, we present the importance of QoS technologies for computer networks, especially in IoT platforms. We present and discuss the difference of the historical approaches on QoS management: Integrated Services (IntServ) and Differentiated Services (DiffServ). The distinction between these approaches is that IntServ was designed to guaranteed QoS and the Diffserv optimized it. We presented why IoT applications need QoS and why it is incredibly challenging. We review the State-of-the-art limitation of current approaches to sustain QoS for applications.

In the following, we explore these artificial intelligence-based computational techniques to bring planners to life (Fig. 2.10).

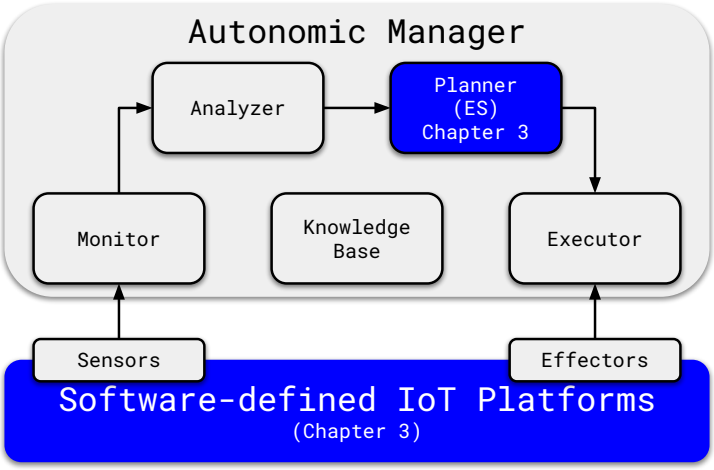


Figure 2.10: Building of the Planner in the Chapters 3.

CHAPTER 3

Joint Optimization of the Scaling Action and the TCFs Deployment

3.1 Introduction

In the NFV environments, autoscaling[1] is used to guarantee a certain level of QoS[2]. Despite the recent efforts made by the industry and academia communities for QoS management in NFV-enabled IoT Platforms (NIPs), we drew the following observations and conclusions. First, the majority of current IoT platform providers, such as Google, AWS IoT Core, or Microsoft Azure IoT, only provide "autoscaling tools" to manage resources provisioned by tenants [Qu 2018]. According to a Microsoft study, [Microsoft 2020], almost all companies involved in IoT have experienced failure of a project at the Proof of Concept stage. The number one reason is the overall high cost of scaling. The autoscaling scheme is automatically triggered when resource usage reaches a given threshold (e.g., CPU usage $> 80\%$). This observation drove the IoT community toward considering the cost minimization of autoscaling as an important research challenge. The second observation is that in those platforms, nodes implement the First-Come-First-Served (FCFS) [oneM2M 2016] as a default traffic control policy. In FCFS, the traffic messages coming from different IoT applications are queued together and served in the order of their arrival. For a given level of resources within a NIP's node, the FCFS processing of IoT applications' traffic traversing this node may lead to the following problem: the resource usage induced by a "greedy" IoT application can lead to QoS degradation for other IoT applications. Therefore, this would trigger a new scaling action to increase the provisioned resources. These first two observations lead to a state that the cost of the provisioned resources is not optimal. To reach an optimal solution, we propose to associate the autoscaling scheme with Traffic Control Functions (TCFs) that take into account the different QoS levels required by the IoT applications. The third observation is that the data centers have held significant Capital Expenditure but have low resource usage. For instance, in 2017 inside Alibaba cluster [Lu 2017], the average CPU utilization per machine was 40%, and maximum maintains about 60%. The average memory utilization per machine was 60% and the maximum about 90%. While at Google[3], the average CPU and memory usage rates in production clusters were only 20% and 40%, respectively, in 2012 [Reiss 2012]. Nearly at the same time, the average CPU utilization rate of Amazon AWS EC2 was only 7% to 17% [Liu 2011]. This observation concludes that there are available resources that one can use to host the TCFs.

The last observation is that in the Cloud-to-Thing continuum (C2TC) [Brogi 2017], the availability and capacity of the resources, namely computation, storage, and connectivity, decrease when moving from the Cloud toward things. Typically, the IoT End Gateways, located close to things, are small devices with limited processing, storage, and connectivity capabilities. Thus, motivated by the above observations and by the promises brought by the existing studies

[1] Autoscaling is a reconfiguration scheme where the number of resources varies automatically based on the load on the platform.

[2] The term QoS refers to the measurement of the overall performance of the NIPs service. We consider the following aspects of the NIPs service: Unavailability, Throughput, and Latency.

[3] No recent information is available today.

[Tootoonchian 2018], one option for the deployment of TCFs within NIPs relies on the use of technologies such as NFV. However, deploying those TCFs only as VMs or OS-level containers (as required by NFV) does not cover the resources and capacities heterogeneity of future networks. In this context, we formulated the following research question:

> *"How to maintain the applications' QoS the closest to their requirements while adapting to the resources' scarcity when moving from Cloud to Things?"*

By answering this question, we seek to build a method that enables End-to-End QoS management in IoT. To fit the decreasing resources' capacities when moving close to the things and make the End-to-End deployment of NFs, an isolation technique (or virtualization solution) that consumes fewer resources is required. Such a solution is one of the contributions of this Chapter that we name Application Network Function (ANF).

Considering these conclusions, our objective is to meet the QoS requirements of IoT applications executed on NIPs and to optimize the costs induced by a classic autoscaling scheme. For this purpose, the approach explored in this Chapter is to take advantage of the different ways of deploying TCFs (i.e., via VNFs or ANFs), while taking into account nodes' resources heterogeneity. In other words, we seek to deploy dynamically (i.e., when the need arises) i) the appropriate TCF (e.g., Dropper, Scheduler), ii) in the appropriate packaging (ANF or VNF), and iii) on the appropriate nodes of the platform (e.g., a Scheduler before a congested node, not after).

The significant contributions of this Chapter are summarized below.

- We introduce the ANF concept, which is based on a minimal level of isolation technique dealing with software execution. The ANFs make possible the deployment of NFs on IoT End Gateways and support reaching the best possible use of available heterogeneous resources capacity C2TC. We design a collection of TCFs that we implement as VNFs and ANFs, with the aim to sustain the QoS level required by the IoT applications. We also provide the performance measurement results to get the quantitative characteristics associated with the different implementation packages (VNFs and ANFs) of the considered TCFs. We study the effects of the traffic arrival rates on the processing time and the resource usage (CPU and RAM) required for executing the TCFs. The performance measurement results are used to solve the multiobjective optimization problem introduced hereafter.

- To achieve optimal deployment of these TCFs on the available nodes in NIPs, we formulate a multiobjective optimization problem. The formulated problem is solved by a planning scheme, named QoS4NIP, that considers both TCFs deployment and scaling actions while optimizing the overall End-to-End QoS.

- We evaluate the benefits in terms of cost-saving of the solutions provided by the QoS4NIP scheme. These benefits are compared to the solutions provided by FCFS (the lazy scheme), the autoscaling scheme, and the two variants of QoS4NIP that do not consider scaling action but only TCFs (the first considers only TCFs deployed as VNFs, and second considers TCFs deployed as VNFs and ANFs). We consider a realistic case study dealing with Connected Vehicles for the validation of our approach. The validation results show that our scheme, QoS4NIP while sustaining the End-to-End QoS for each application in NIPs, achieves the best cost-saving amongst the existing competing approaches.

The remainder of the Chapter is structured as follows. Section 3.2 discusses the state-of-the-Art. Section 3.3 develops the proposed approach and explains the key concepts. Section 3.4 presents the implemented TCFs and the performance evaluations of the implemented VNF and ANF concepts. Section 3.5 is devoted to the QoS4NIP scheme description. Section 3.6 demonstrates the proposed approach effectiveness for the Connected Vehicles case study. The proposed work considered hypotheses are discussed in Section 3.7. Finally, Section 3.9 concludes the Chapter.

3.2 State-of-the-Art

Several fields, such as information-centric networking (ICN), overlay network, and network slicing, consider the use of NFV for the management of QoS. In this Chapter, since we only aim to contribute to this domain for the IoT context, we consider the reference contributions made in the literature. The following sections present a literature review analysis on *overhead minimization* for cost saving in NFV and *runtime optimization* in NFV that are essential aspects of the proposed approach.

3.2.1 Overhead Minimization in NFV

As presented in Section 3.4, in this Chapter, we propose the ANF concept to enable the deployment of NFs over the full C2TC. The existing literature involves research proposals aiming to reduce the massive footprint of today's NFV platforms [Cziva 2017a, Palkar 2015, Riggio 2015, Yasukata 2017, Gallo 2018]. In [Cziva 2017a], the authors present the Glasgow Network Functions (GNF), a container-based NFV platform that runs and orchestrates lightweight container-based VNFs, saving core network utilization and providing lower Latency. Palkar et al. [Palkar 2015] propose a framework (E2) for NFV packet processing. E2 provides a single coherent system for managing NFs while relieving developers from developing per-NF solutions for placement, scaling, fault-tolerance, and other functionalities. Riggio et al. [Riggio 2015] propose a MEC OS that supports lightweight virtualization. Yasukata et al. in [Yasukata 2017] propose HyperNF, a high-performance NFV framework aiming at maximizing server performance when concurrently running large numbers of NFs. HyperNF implements Hypercall-based

virtual I/O, placing packet forwarding logic inside the hypervisor to significantly reduce I/O synchronization overheads. Gallo et al. [Gallo 2018] propose a scalable NFV-based solution as a novel approach that satisfies the stated requirements for user-centric support of IoT devices. The main differences between all these frameworks and our proposal are related to the isolation of NFs. Since isolation is not a mandatory requirement in our context (the ANFs being used are considered trusted because they are only supplied by the NIP operator), ANFs have a more reduced overhead than hypervisor-based NFs.

Furthermore, the virtualization technologies proposed in these studies still have significant memory, and CPU requirements [Nandugudi 2016] for the C2TC. These solutions are not adapted to the common IoT End Gateways capacities. At the same time, their needs for a particular hypervisor prevent them from operating on these gateways.

3.2.2 Runtime Optimization for Cost-saving in NFV

Most of the work for cost saving in NFV consider the *initial planning step* or the VNFs initial development step (i.e., design-time optimization), as described in detail in [Herrera 2016]. However, the few works that deal with the *Runtime Optimization* for cost saving in NFV problematic, radically, consider to automatically scale the resources provisioned to the platforms without human intervention under a dynamic workload, to minimize resource cost while satisfying each application QoS requirement [Qu 2018]. Only considering the autoscaling scheme in these studies without differentiation in the QoS levels leads to a non-optimal scheme and induces high relative costs. Ren et al. propose in [Ren 2018] an adaptive autoscaling algorithm (ASA) using an analytical model to balance the cost-performance trade-off while maintaining an acceptable level of performance for 5G mobile networks. ASA adds (or removes) VNF instances according to the number of user requests waiting. Rahman et al. propose in [Rahman 2018], a proactive Machine Learning (ML)-based approach to perform autoscaling of VNFs in response to dynamic traffic changes. The authors propose an ML-based planner that learns VNF (VMs and Docker containers) scaling decisions and behavior of network traffic load to generate scaling decisions ahead of time. However, the conducted experiments show that such a proposal has a high financial cost. Similarly, [Tang 2015] investigates a reinforcement learning approach for autoscaling on NFV. Exploring a different approach, [Rahman 2020] proposes a negotiation-game-based autoscaling method where tenants and service providers both engage in the autoscaling decision, based on their willingness to participate, different QoS requirements, and financial gain (e.g., cost savings). Also, [Rahman 2020] proposes a proactive ML-based prediction method to perform SFC autoscaling in dynamic traffic scenarios. Searching beyond the autoscaling scheme, Draxler et al. [Draxler 2018] propose JASPER, a fully automated approach for jointly optimizing scaling, placement, and routing for multiple network services, consisting of of of numerous VNFs. JASPER manages various network services that share the same substrate network, dynamically adds or removes services, and

handles workload changes. On a similar line, [Toosi 2019] and [Liu 2017] study how to optimize SFC deployment and readjustment in a dynamic situation. Authors in [Liu 2017] try to jointly maximize the implementation of new users' SFCs and the adaptation of in-service users' SFCs while considering the trade-off between resource usages and operational overhead. Quang et al. in [Quang 2018] extend the SFC deployment and readjustment in a dynamic approach. [Quang 2018] examines VNF migration by providing a model that solves the adaptive and dynamic VNF allocation problem under QoS constraints. Yu et al. [Yu 2017] extended the SFC readjustment in a proactive situation approach. [Yu 2017] considers load balance, energy cost, and resource usages to formulate a multiobjective problem. Contributions in [Ren 2018, Rahman 2018, Tang 2015, Rahman 2020, Quang 2018, Yu 2017] do not consider network IoT End Gateways resource constraints, and this limits the applicability for NIPs. While contributions in [Draxler 2018, Toosi 2019, Liu 2017] authors explicitly acknowledge it in their approaches. In [Cheng 2018], Cheng et al. investigate the issues of network utility degradation when implementing NFV in dynamic networks and design a proactive NFV solution from a stochastic perspective. Unlike existing deterministic NFV solutions that assume given network capacities and static service quality demands, their work explicitly integrates the knowledge of substantial network variations. Targeting End-to-End reliability of mission-critical traffic, Petrov et al. in [Petrov 2018] introduce a softwarized 5G architecture. [Petrov 2018] also proposes a mathematical framework to model the process of critical session transfers in a 5G access network and to quantify their impact (QoS interferences) on other users traffic flows. They implemented, in [Petrov 2018], a hardware prototype to investigate the practical effects of supporting mission-critical data in a 5G NFV-enabled core network.

In summary, the existing literature lacks the attention to NIPs in two perspectives. On the one hand, several [Ren 2018, Rahman 2018, Tang 2015, Rahman 2020, Quang 2018, Yu 2017] studies failed to take the available heterogeneous resources capacity of the C2TC into account. On the other hand, none of the current studies consider the traffic control perspective for cost saving in NIPs. The approach we propose here addresses the shortcomings of the related work mentioned below.

3.3 Key Concepts and Approach Overview

The main originality of our contribution consists of the combination of several changes in the autoscaling approach, with the aim to optimize the cost of QoS management for NIPs. The first change (Section 3.3.1) consists in considering the on-the-fly deployment of the TCFs to sustain applications' QoS in NIPs. The second change (Section 3.3.2) consists in considering the ANFs, in addition to the VNFs, for the TCFs deployment. The last change (Section 3.3.3) deals with the elaboration and implementation of a new planning scheme, QoS4NIP, which jointly optimizes scaling actions and TCFs deployment. QoS4NIP take into account the TCFs

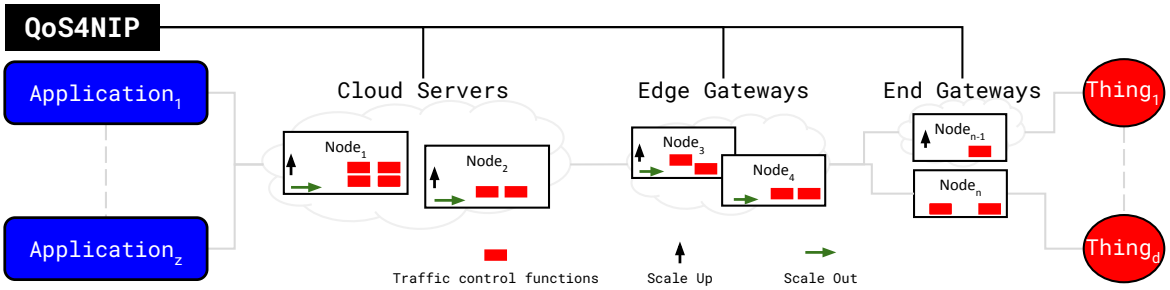


Figure 3.1: Approach overview over the Cloud-to-Thing continuum.

deployment costs and the scaling actions costs.

3.3.1 The Traffic Control Functions

We handle the NIPs that implement the common reference architectures, such as oneM2M [oneM2M 2016]. More specifically, we deal with *stateless* communication (i.e., no stored knowledge of or reference to past requests) between the platform nodes (i.e., Server and Gateways). Such architectural frameworks allow TCFs to be inserted in the platform nodes without disturbing the supported IoT applications. Based on these features, we consider the TCFs proposed at the IP network level by Carpenter et al. in [Carpenter 2002b], that we adapt to the specifics of the IoT traffic context. We manage the QoS in a NIP by implementing and distributing on-the-fly the adequate TCFs on the NIP's nodes. We consider that NIPs' nodes in the Cloud Server and Edge Gateways have nested virtualization capabilities for hosting VNF in VM (e.g., running Docker over Amazon EC2 VMs) [Ren 2017]. Let us remark here that our approach cannot be applied for all platforms, typically multimedia streaming platforms, because of the stateful aspects (i.e., requests are performed with the context of previous requests, and the current requests may be affected by what happened during previous requests) of End-to-End protocols (i.e., Real-time Transport Protocol or RTP and Real-time Streaming Protocol or RTSP) widely used in this context.

3.3.2 The ANFs packaging solution

To fully enable the deployment of TCFs over the C2TC, we consider the solution explored in [Kohler 2000, Decasper 2000], leading to package NFs into software components that one can deploy on-the-fly on NIPs' nodes. We then distinguish, in Fig 3.2, two types of NFs. The first type consists of NFs hosted inside virtual containers (VMs or OS-level containers like Docker). This type of function is commonly called VNF [ETSI 2014]. The second type consists of NFs hosted inside a program as a software component. We call them Application NFs (ANF) in the sequel.

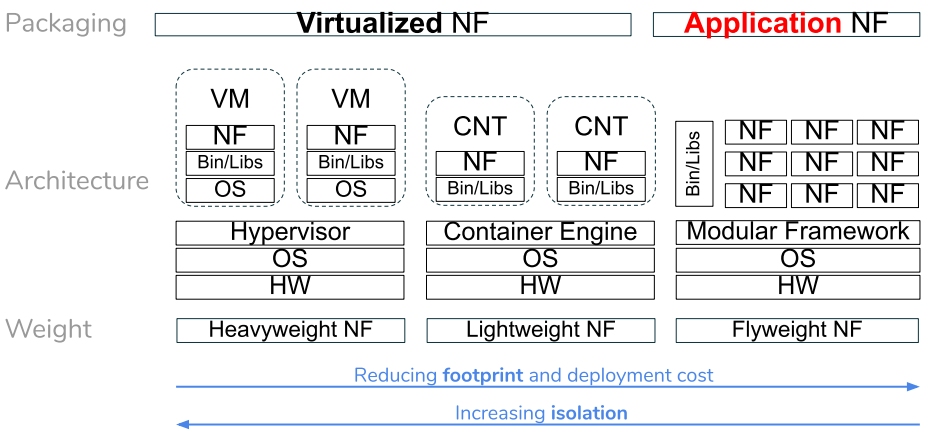


Figure 3.2: Network Functions Instances.

The ANF concept does not give the same isolation as the VNF concept. Isolation is one of the inherent features highlighted in the existing NFV platforms. Isolation allows the NFs to run on the same hardware and not interfere with each other from two standpoints [Panda 2017]: security and performance. OS-level containers and VMs are the two virtualization techniques commonly used to provide isolation. On the one hand, different studies, such as [Nandugudi 2016, Gallo 2018], show that these virtualization techniques generally induce a high usage of resources. On the other hand, the IoT End Gateways, located close to things, are generally small devices with limited processing, storage, and connectivity capabilities. For example, an IoT Gateway, such as a *Dell Edge Gateway 5000*, has an Intel Atom processor clocked at 1.33 GHz with only 2 GB of memory. Deploying NFs as VMs or Os-level Containers does not fit NIPs' heterogeneity of resource capacities. Indeed, deploying a VNF as a standard Linux VM requires a minimum[4] of 256 Megabytes RAM, a 300 MHz x86 processor, and 1.5 GB of disk space. Deploying this VNF as a container requires a minimum[5] of 29 Megabytes of disk space. This requirement sharply limits the amount of VNFs that can be deployed on a node and drastically reduces the number of compliant nodes. For most IoT End Gateways, it is difficult to host multiple instances of such VNFs. Moreover, ANF adapts the NF packaging to the constrained deployment context using specific solutions for each chosen programming and deployment environment. The runtime environment "ANF-host" executes on-the-fly ANFs written in a specific programming language (Java in our case). Some of the characteristics of such a runtime environment are:

- an ANF is held and versioned in a code repository;
- ANF dependencies are explicitly defined;

[4]https://help.ubuntu.com/community/Installation/SystemRequirements
[5]https://hub.docker.com/r/_/ubuntu/

- an ANF can be deployed into development, staging, or production environments without changes;
- an ANF configuration is stored in the environment, for instance, through environment variables;
- backing services, such as data stores, message queues, and memory caches, are accessed through a network, and no distinction is made between local or third-party services;
- an ANF is stateless, and therefore, enables easy scale out.

All these characteristics allow elementary ANFs to be chained, the same way as VNFs, to provide complex services commonly named SFC. For example, when using an OSGi-based modular platform, the ANF is uploaded on the ANF-host through a specific protocol, often HTTP. The OSGi specifications assume an architecture to remotely manage the OSGi framework components relying on a Management Agent (MA). The MA receives, verifies, installs, and configures the ANF according to a "Manifest" file that is similar to the VNFD defined by ETSI-NFV. The method to deploy multiple ANFs (ANFs SFC) implements the "Pre-Calculated Deployment" specified by OSGi [Alliance 2018]. A pre-calculated deployment process is initiated using one of the OSGi subsystem service's install methods. In this case, ANFs SFC (OSGi-based) is an OSGi subsystem deployed as an OSGi Subsystem Archive (.esa) file. An OSGi subsystem comprises resources, including OSGi bundles (ANFs).

3.3.3 The Joint Optimization of the Scaling Action and the TFCs Deployment

Fig 3.1 illustrates the overall approach. In this figure, the IoT applications run on top of the NIP (e.g., on a Cloud Server or a User Device). The presence of a square (red) indicates the deployment of a particular TCF on a node (NFV-I or ANF host). The up-arrow (blue) and the right-arrow (purple) indicate the execution of a scaling action on a node (scale up and scale out, respectively). Scale out means adding more instances to a NIP' node, and scale up means adding more resources to a NIP' node. The overall approach relies on the TCFs deployment on the NIPs' nodes and the scaling action execution to sustain the QoS for the IoT applications.

Nevertheless, the IoT application QoS' fulfillment in NIPs can be seen as a multiobjective optimization problem where objectives are all the different application QoS (e.g., latency, throughput). This problem raises a set of optimal solutions (known mainly as Pareto-optimal solutions) instead of a single optimal solution. A solution is Pareto-optimal if we cannot improve any of the objectives without degrading the others. Without additional subjective preference, all Pareto-optimal solutions are considered equally "good." Classical optimization methods suggest converting the Multiobjective optimization problem artificially to a Single-Objective optimization problem. This usually requires the repetitive use of an algorithm to find multiple

Pareto-optimal solutions. On some occasions, such usage does not even guarantee to find Pareto-optimal solutions [Deb 2002]. In contrast, the population evolution approach of Evolutionary Algorithms (EA) allows an efficient way to find simultaneously multiple Pareto-optimal solutions in a single run [Deb 2002]. This is the most popular approach in the literature. We implement this approach in this Chapter. Additional studies on the Multiobjective EA can be discovered in [Deb 2001].

Moreover, the joint optimization of the TCFs deployment and the scaling action execution is very similar to a Knapsack Problem (a widely known non-deterministic polynomial-time hard problem). For this problem, we propose a meta-heuristic based on the GA that have been proven to constitute an efficient method to provide suitable near-optimal solutions in a short amount of time (see Section 3.5).

For convenience, the used notations in the rest of this Chapter are listed in Table 3.1.

Names	**Meanings**
r	message arriving at a NF
C	set of considered types of services (or traffic classes)
d_t	NF deployment time
p_t	NF processing time
t_r	resource was requested timestamp
t_s	resource was served timestamp
τ	IoT application
z	Total number of IoT applications
n	Total number of nodes
L_{Qos_τ}	Latency required by τ
T_{Qos_τ}	Throughput required by τ
U_{Qos_τ}	Unavailability required by τ
L_{E2E_τ}	End-to-End Latency served to τ
T_{E2E_τ}	End-to-End Throughput served to τ
U_{E2E_τ}	End-to-End Unavailability served to τ
L_{i_τ}	Latency of τ on node i
T_{i_τ}	Throughput of τ on node i
U_{i_τ}	Unavailability of τ on node i
ρ_{i_τ}	Monitored Throughput on node i for τ
δ_i	Monitored Latency on node i
m	Total number of scaling actions
p	Total number of TCFs
A_i	Set of scaling actions supported by the node i
F_i	Set of TCFs supported by the node i
f_q	TCF q benefit
a_c	Scaling actions c benefit
Γ_{i_c}	Cost of scaling action c on node i
cpu_q	TCF q cpu resource usage
ram_q	TCF q ram resource usage
η_i	Sum of the benefits induced by all the supported TCFs and the scaling actions on the node i
ζ_i	Sum of the benefits of the Throughput induced by all the supported scaling actions on the node i

continued ...

... continued

Names	Meanings
ω_i	Weighting factor of the Scheduler on the Throughput on the node i
ϵ_i	Loss factor of the Dropper on the Unavailability on the node i
λ_i	Request arrival rate on node i
$Cost_{E2E}$	End-to-End Cost of the scaling action
$Cost_i$	Cost of all the scaling action execution on node i
RU_{E2E}	End-to-End Resource usage of TCFs
RU_i	Resource usage of TCFs on node i
$H_{i_{cpu}}$	Node i cpu usage
$H_{i_{ram}}$	Node i ram usage
β_i	Node i cpu usage with Scaling action execution and TCFs deployment
γ_i	Node i ram usage with Scaling action execution and TCFs deployment
X_θ	Binary vector describing the describe the application of TCFs or scaling actions to the NIPs' nodes
x_i^j	Binary row of X_θ
T_θ	Integer matrix describing the genes' additional information
t_i^j	Integer row of T_θ
P_t	Pareto front
C_p	Crossover probability
M_p	Mutation probability
N	Population size
l	Chromosome length
T	Maximum number of generations

Table 3.1: Notations

3.4 Network Functions for TCFs in NIPs

The traffic control mechanisms proposed at the IP level by Carpenter et al. in [Carpenter 2002b] inspired the proposed functions. [Carpenter 2002b] introduces DiffServ, an architecture based on a simple model within which the IP traffic that arrives in the network gets assigned to a class of behavior. Each class is uniquely identified by a "Tag" in the IP packets. All the intermediate routers process packets following the behavior associated with their "Tag." For instance, 80% of the bandwidth of a router belongs to packets tagged A and 20% to those tagged B.

A small number of functions can be composed to differentiate the level of service provided to the IoT applications according to their QoS requirements. The traditional functions (i.e., Classifier, Marker, Dropper, Shaper, Scheduler) are split up simply and deployed when needed. For example, we can deploy a dropping function without the shaping function (avoiding its overhead) and vice versa. We added to these functions a Redirector. The Classifier and Marker were merged into a new Classifier capable of marking IoT traffic. We package these functions in NFs (ANF and VNF), deploy, and configure them on-the-fly on the targeted NIPs' nodes (see Fig. 3.1). In these functions, traffic is composed of one or several messages that cross the NIP's nodes (Server, Gateways); and a traffic profile specifies the temporal properties, such as the

rate and the burst size of the traffic. It provides the rules for determining whether a message is in or out of the profile. Usually, this profile is expressed within a Service-level agreement (SLA) between the application (client) and NIP (Service Provider).

This Section presents a) an overview of the TCFs implemented as ANF and VNF to sustain the QoS level to the IoT applications; b) performance evaluations of the VNF and ANF concepts.

3.4.1 Traffic Control Functions Overview

In the remainder of this Chapter, a message arriving at a function is denoted r; C denotes the set of considered traffic classes (or types of services). A class in r is a header called Local Service Level (LSL). Since the TCFs are handling traffic classes, it is possible to group IoT applications with similar QoS requirements in a class. Indeed in a real scenario, this is what should be done. In the following, for the sake of straightforwardness, each IoT application will be assigned a distinct traffic class. Below, we explain each of the considered functions, and we propose the algorithms implementing them on NIPs as TCFs.

Classifier. This function is used to "distinguish" the incoming traffic for further processing. The Classifier allows identifying a message r of an IoT application and updating its header with the appropriate LSL. The Classifier identifies the messages based on their headers' content according to a set of predefined rules, typically some combination of source and destination addresses, content-type, protocols, source, and destination ports fields. Algorithm 1 implements the Classifier. Line 2, the algorithm, first tries to identify the class c of the message r. From lines 3 to 4, the Classifier adds a message header with the associated LSL when it recognizes traffic.

The time complexity of the Classifier (Algorithm 1) is $\mathcal{O}(|C|)$, where $|C|$ denotes the number of elements of the set C. We may even handle a fixed number of classes making $|C|$ a constant in practice.

Algorithm 1: Classifier Network Function

```
   // r: IoT application message
   // C: Traffic classes
   Input: r, C
   Output: r
1  begin
2      c ← ComputeClass(r)
3      if c ≠ {} then
4          r ← AddMark(r,c)
5      return r
```

Dropper. This function allows discarding messages based on their LSL header. The Dropper discards some or all messages in an IoT application traffic to bring this traffic into compliance

with an expected profile. A REST API is used to configure the rejection percentage and the targeted traffic of this function. Algorithm 2 implements the Dropper. Line 2, upon the reception of a message, the Dropper identifies the associated LSL in the message header. Then, from lines 3 to 6, the algorithm calculates the previously rejected percentage for the considered traffic profile. Line 7, the algorithm rejects the message, return *null* when the percentage of the rejected messages is lower than the specified limit in the configured policy. Otherwise, the Dropper forwards the message without any modification. In line 10, the Dropper update associated the rejection percentage.

The time complexity of the Dropper (Algorithm 2) is $\mathcal{O}(|C|)$, where $|C|$ denotes the number of elements of the set C. We may even handle a fixed number of classes making $|C|$ a constant in practice.

Algorithm 2: Dropper Network Function

```
   // r: IoT application message
   // C: Traffic classes
   Input: r, C
   Output: r or null
1  begin
2      c ← GetMessageLSL(r)
3      if c ≠ {} then
4          for k ∈ C do
5              if k = c then
6                  c_past ← GetRejectionPercentage(k)
7                  if c_past < c then
8                      r ← null
9      UpdateRejectionPercentage(k)
10     return r
```

Shaper. This function allows delaying the traffic messages to make them compliant with a defined traffic profile. The Shaper discards some messages if there is not enough space in the buffer to hold the delayed messages. The Shaper uses the LSL to identify the delay time of a message. A REST API is used to configure the delaying time and the targeted traffic of this function. Algorithm 3 implements the Shaper. Line 2, the algorithm tries to identify the LSL of the message in its LSL header. From lines 3 to 7, the Shaper holds the message for the necessary delay time matching the identified profile. Line 8, after the elapsed delay, the function returns the message without modification.

The time complexity of the Shaper (Algorithm 3) is $\mathcal{O}(|C|)$, where $|C|$ denotes the number of elements of the set C. We may even handle a fixed amount of classes making $|C|$ a constant in practice.

Algorithm 3: Shaper Network Function

```
   // r: IoT application message
   // C: Traffic classes
   Input: r, C
   Output: r
1  begin
2      c ← GetMessageLSL(r)
3      if c ≠ {} then
4          for  k ∈ C) do
5              if k = c then
6                  d ← GetDelay(k)
7                  Wait(d)
8      return r
```

Scheduler. This function enables the management of the incoming message sequence according to their LSL headers. The function serves any message with a high LSL before a message with a low LSL. If two messages have the same LSL, then the function serves according to their enqueued order. A REST API is used to configure the associated traffic classes in the queue. Algorithm 4 implements the Scheduler. From lines 1 to 12, the first main procedure enqueues the received message in an internal queue. It delivers this message while it moves to the head of the queue. From lines 13 to 16, the second procedure reorders the messages inside the queue according to their LSL.

The time complexity of the Scheduler (Algorithm 4) is $\mathcal{O}(|C| + |Q| \log |Q|)$, where $|C|$ denotes the number of elements of the set C and $|Q|$ denotes the length of the queue Q. In practice, we may even handle a fixed number of classes making $|C|$ a constant, and then, the time complexity is $\mathcal{O}(|Q| \log |Q|)$.

Redirector. This function enables the interception and the forwarding of traffic messages towards different targets. The routing scheme (at the platform level) is affected by this function since we are using an oneM2M-based [oneM2M 2016] NIP where the routing is at IoT application-level (level 6 of OSI layering). This modification is completely transparent to the IoT application. A REST API is used to configure the new destination and the targeted traffic for this function. Algorithm 5 implements the Redirector. Line 2, the Redirector, identifies the LSL of the message according to its LSL header. From lines 3 to 6, it changes the message's destination (e.g., to another node) according to the corresponding identified LSL. Line 7 it sends the message to its new destination without an LSL and additional modifications.

The time complexity of the Redirector (Algorithm 5) is $\mathcal{O}(|C|)$, where $|C|$ denotes the number of elements of the set C. We may even handle a fixed number of classes making $|C|$ a constant in practice.

Algorithm 4: Scheduler Network Function

```
   // r: IoT application message
   // C: Traffic classes
   // Q: IoT application message Queue
   Input: r, C, Q
   Output: r
1  function Priority-based Scheduler()
2      begin
3          c ← GetMessageLSL(r)
4          if c ≠ {} then
5              for  k ∈ C) do
6                  if k = c then
7                      Q.push(k, r)
8                      while r ¬ Q.peek() do
9                          Wait()
10                     r ← Q.pull()
11         return r
12 function PriortySortQ()
13     while true do
14         if Q ≠ {} then
15             Timsort(Q)
```

Algorithm 5: Redirector Network Function

```
   // r: IoT application message
   // C: Traffic classes
   Input: r, C
   Output: r
1  begin
2      c ← GetMessageLSL(r)
3      if c ≠ {} then
4          for  k ∈ C) do
5              if k = c then
6                  d ← GetRedirectionIP(k)
7                  r ← SendTo(r,d)
8      return r
```

3.4.2 Evaluation of the TCFs packaging (VNF and ANF)

In this Section, we evaluate the deployment time of the TCFs implemented as VNF and ANF. Then, we study the effects of the traffic arrival rates on the processing time and the resource usage (CPU and RAM) required for executing the TCFs. The goal is to get quantitative characteristics associated with the different packaging (ANF and VNF) of the TCFs.

The details of the TCFs implementation are provided in the Appendix A.

Experimental context. The presented performance measurements allow assessing the deployment time, denoted d_t, of the TCFs: as ANF in the considered ANF-host (i.e., IoT

Gateway); and as VNF in the considered NFV-I nodes. In order not to bias the tests by an additional upload time related to network conditions, the TCFs are supposed to be already present in the hosting system as Docker Images for VNFs, and JARs files for ANFs. To collect performance metrics, we implemented monitoring tools based on Java Management Extensions (JMX) technology. In each TCF, we created MBeans objects for processing time, CPU, and RAM remote monitoring.

We characterize the processing time, denoted p_t, associated with each function under the effects of request arrivals. Let a session $\mathcal{S} = (r_1, r_2, \ldots, r_n)$ be a sequence of n requests for resource r_i coming from the same IoT application, and let $t_r(r_i)$ and $t_s(r_i)$, respectively, be the time that resource r_i was requested and the time that resource r_i was served, respectively. The processing time for request r_i in session $\mathcal{S}$ is:

$$p_t(r_i) = t_s(r_i) - t_r(r_i) \tag{3.1}$$

According to [Metzger 2019], the Poisson distribution for modeling the traffic of an IoT application to the Cloud is a good approximation for the scalability analysis. Thus, to simplify, we assume that the arrivals of the IoT traffic in a session follow a Poisson distribution. An event (request arrival) can occur k times (0 to n) in a given interval. The probability P of observing k events in an interval is given by Equation:

$$P(kt) = e^{-\lambda}\frac{\lambda^k}{k!} \tag{3.2}$$

where :

e is Euler's number ($e = 2.71828...$)
k is the number of times a request arrive in an interval and takes values $0, 1, 2, \ldots$
λ is the request arrival rate.

The experimental testbed consists of three host machines: one traffic generator equipped with two CPU and 4 GB RAM, one NFV-I equipped with four CPU and 16 GB RAM, and one ANF-host fitted with one CPU and 4 GB RAM. All the CPUs are CPU Intel Core i7-7500U clocked at 2.70 GHz. The traffic generator produces the IoT traffics according to a Poisson distribution with a request arrival rate of $\lambda \in [1, 50, 100, 150]$ req/s[6] (request size = 1 Mb). The NFV-I is composed of all hardware and software components that build up the environment in which VNFs are deployed and managed using the OpenBaton [Carella 2015] platform. The ANF-host is running an OSGi-based [Alliance 2018] program that can deploy ANFs. The three host machines run with Ubuntu 16.04. The template (*size*) of a VNF/ANF is 1 CPU and 4 Gigabytes RAM. In these experiments, a message is an HTTP request or an HTTP

[6]These different request arrival rates are considered realistic [Banouar 2017]

response. The considered NIP is the Eclipse open-source OM2M [Alaya 2014] that implements the standard oneM2M [oneM2M 2016].

The conducted experiments address the following questions:

(a) How does the TCF type (ANF or VNF) impact the deployment time;

(b) How does λ in Equation (3.2) impact the processing time defined in Equation (3.1);

(c) How does λ in Equation (3.2) impact the CPU and RAM usage;

(d) How does the CPU and RAM saturation impact the TCF performance.

Performance analysis. In the first experiment, we answer the question "(a)" by investigating the *TCF deployment time*. We examine the relationship between the TCF type (ANF and VNF) and their deployment time. Fig. 3.3 shows the results in a logarithmic scale. The deployment time of an ANF with an average weight of 15 Kbits is ≈ 8 ms; the deployment time of a VNF having an average weight of 200 Megabytes is ≈ 520 ms. These results were predictable, but they still had to be quantified.

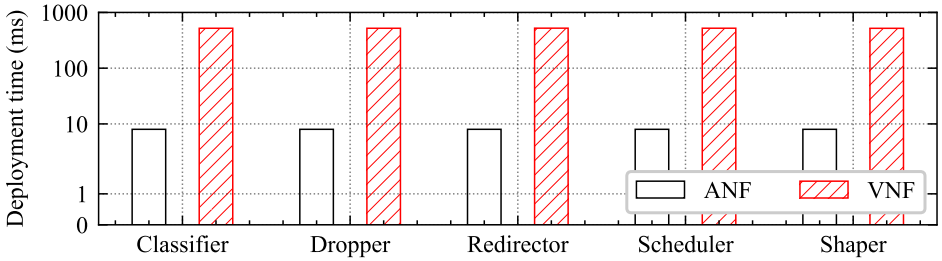


Figure 3.3: Traffic Control Functions deployment time.

The second experiment investigates the *TCF processing time* to answer the question "(b)." We analyze the relationship between the request arrival rate λ and the processing time p_t. We start with each implementation (ANF and VNF) of each TCF facing a session $\mathcal{S}$ of 3000 requests and a $\lambda = 1$. Then, repeatedly, with the same session $\mathcal{S}$ of 3000 requests, we increase λ first to 50, then to 100, and finally to 150. The results (shown in Fig. 3.4 and Fig. 3.5) confirm the expected behavior: the increase of λ leads to the increase of the processing time. For instance, in Fig. 3.4, with $\lambda = 1$, we have a $p_t(min) = 0$ ms, $p_t(median) = 2$ ms, $p_t(max) = 50$ ms for the Dropper Network Function processing time. However, the cumulative distribution function (CDF) of the same TCF facing the same λ differs depending on its type (ANF or VNF). In Fig. 3.4 and Fig. 3.5, the Classifier processing time represents the insertion of the tag (LSL). For instance, in Fig. 3.4, it takes 2 ms to insert a tag for almost 75% of the requests when we handle 1 req/s. Additionally, we handle tags only internally inside an infrastructure node of the NFV network topology (NFV-I). The Classifier calculates the tag in each node according

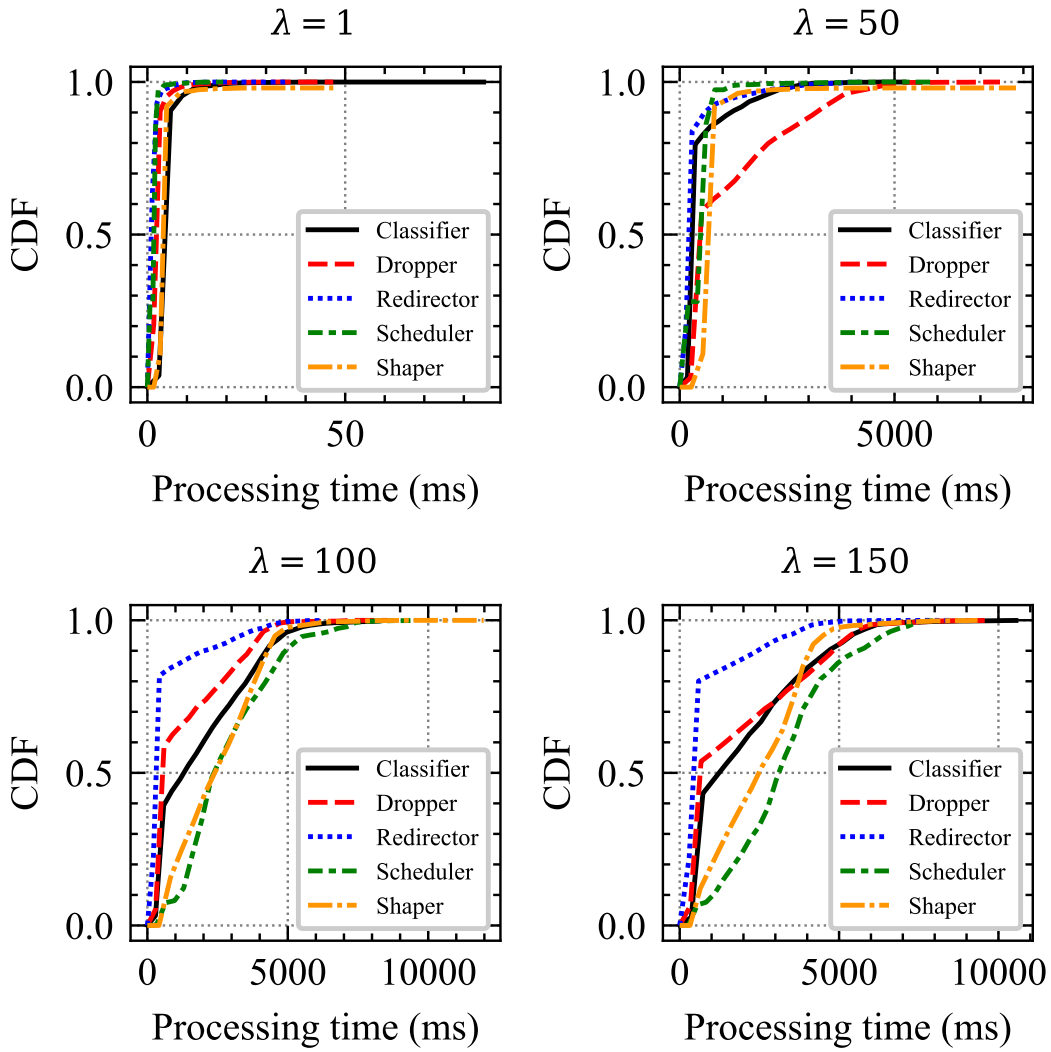


Figure 3.4: [ANF] Traffic Control Functions Processing Time.

to the content of the message headers. The tag is not transmitted outside of the NFV-I node entities and no transmission overhead is then induced by the message exchange between the NFV-I nodes, which is the most significant part of the communication traffic. The same applies to ANF-host.

In the third experiment, we answer the question "(c)" by investigating the *TCF resource usage*. We audit the relationship between the request arrival rate λ and the resource usage (CPU and RAM). We start with each implementation (ANF and VNF) of each TCF facing a session $\mathcal{S}$ of 3000 requests and a $\lambda = 1$. Then, we repeat, with the same session $\mathcal{S}$ of 3000 requests, raising λ first to 50, then to 100, and finally to 150. Using ANF, there is essentially no isolation in the use of resources, so we approximate the ANF resource usage to the whole resource usage of the Java Virtual Machine hosting it, which is the worst situation of resource

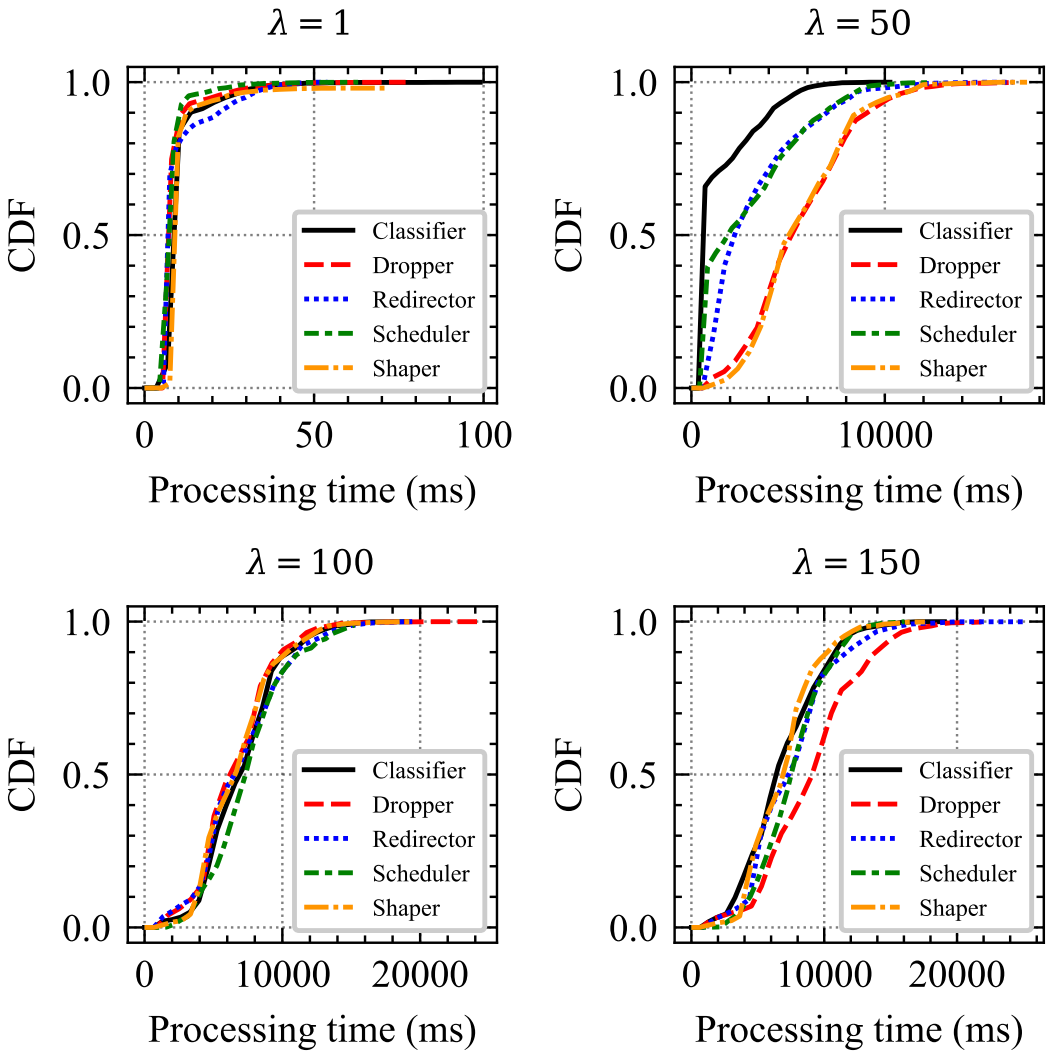
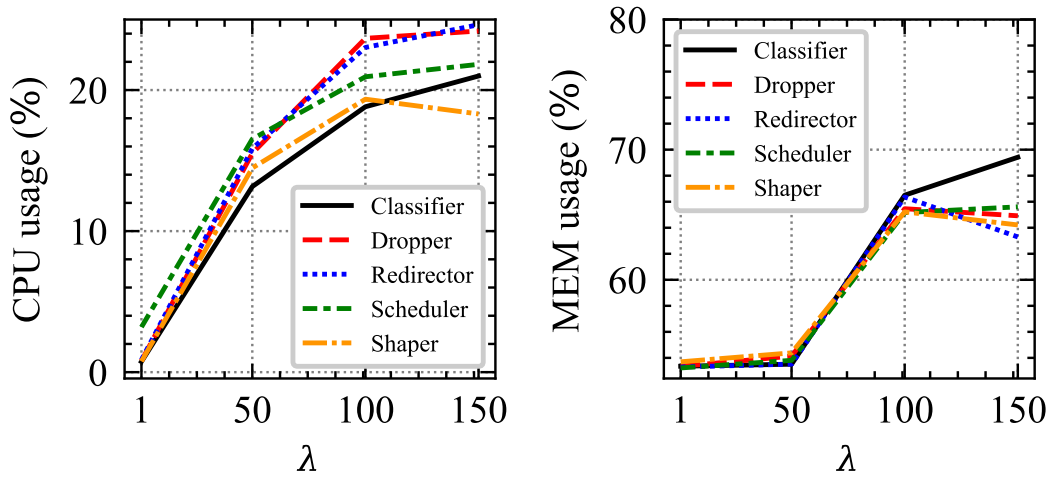


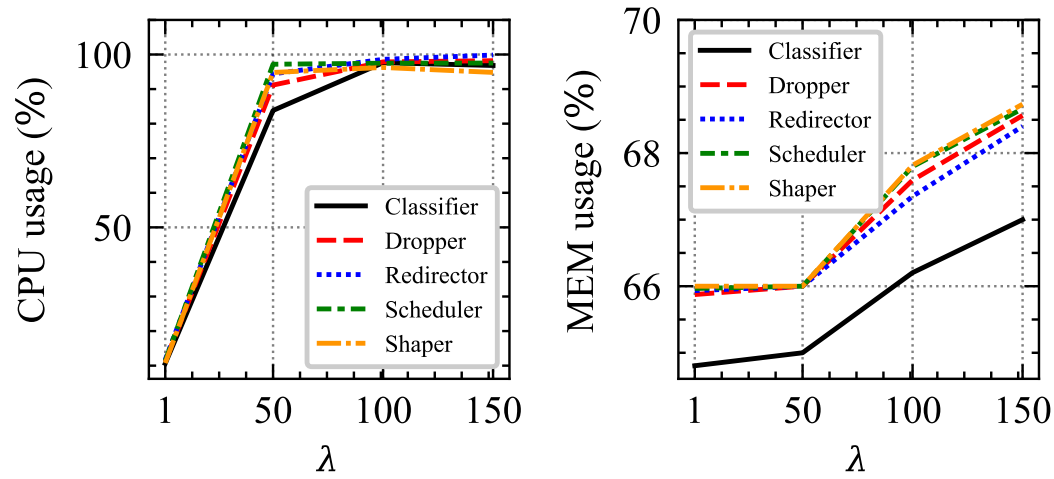
Figure 3.5: [VNF] Traffic Control Functions Processing Time.

usage. For each session of every TCF (ANF, VNF), we measure the average usage of CPU and RAM. Fig. 3.6a and Fig. 3.6b show these average usages. The results show that the TCFs implemented as VNFs consume more CPU compared to ANFs. However, both (ANFs and VNFs) consume the same amount of RAM.

Saturation effect on TCFs: Here, we answer the question "(d)" by exploring the relationship between the resource saturation (i.e., when CPU and/or RAM are utilized over 90%) and the TCF performance. As shown in Fig. 3.4 and Fig. 3.5, the CPU saturation has an important influence on the VNF TCFs performance. From $\lambda = 50$, we can see that, on the one hand, the CPU is used at $\approx 100\%$, and the p_t reaches 5 seconds (Fig. 3.4). On the other hand, the RAM remains slightly congested because the proposed TCFs are almost stateless and computation intensive.



(a) ANF Resource Usage.



(b) VNF Resource Usage.

Figure 3.6: Traffic Control Functions Resource Usage.

By reducing the isolation between NFs, we lose strict resource isolation. However, we decrease the overhead (resource usage, deployment time), reduce the hosting nodes' complexity, and increase the number of hosting nodes. ANFs allow End Gateways with low capacity to accommodate TCFs and therefore to act on (upbound) IoT traffic. Considering, under some circumstances, the strict isolation as a non-mandatory functionality for the deployment of NFs, the concept of ANF completes the global toolset that sustains QoS in NIPs. Our approach aims to dynamically deploy the different TCFs presented in this Section within the platform, according to resources and requirements changes. Given the task's complexity, several TCFs can be considered, but with varying results and deployment opportunities. Section 3.5 presents our contribution, based on combinatorial optimization heuristics, to decide the best combination

of TCFs to deploy appropriately to the current context. Section 3.6 presents the method of evaluating the performance of our contribution, as well as the results obtained.

3.5 Design of QoS4NIP Planner

In this Section, we describe the design of QoS4NIP based on the considerations mentioned above. QoS4NIP considers the different trade-offs for the autonomous management of QoS in NIPs. In Section 3.5.1, we describe the considered system model. In Section 3.5.1, we formulate the QoS model of IoT applications (Latency, Throughput, and Availability), the scaling actions cost model, and the TCFs deployment resource usage model.

As stated in Section 3.3.3, we formulate in Section 3.5.2 a multiobjective optimization problem for efficient planning. We propose in Section 3.5.3 a modelization for the problem resolution (GA-based Constrained Optimization Model). To solve the multiobjective optimization problem, we explore in Section 3.5.4 the evolutionary strategies and the Pareto front. Finally, we present in Section 3.5.5, the QoS4NIP planner algorithm.

3.5.1 System Model

Fig. 3.1 depicts the system model used by the multiobjective optimization algorithm presented in this Chapter. Let the NIP be composed of a set of n TCF (VNF or ANF) hosting nodes that are already provisioned and are parts of the infrastructure. Let consider that each TCF or scaling action is associated with a benefit (estimated a priori). This could be justified; for example, using the classical response time model $R = S/(1-U)$, where S is the node service time, U is the node (resources) utilization. The execution of a scaling action decreases U and therefore decreases R. For simplicity, we call this variation ($R/R_{\text{with scaling action}}$) the "benefit" of the scaling action. This benefit, expressed as a percentage (%), describes how the scaling action influences the QoS on the hosting node. The same applies to the TCFs. For instance, a benefit of 25% means that the TCF or the scaling action reduces the targeted IoT traffic Latency by 25% (to the detriment of other traffics that are not targeted).

The *joint optimization problem* is to find the relevant TCFs to deploy (or remove) on every node of this set and the scaling actions to execute while optimizing the overall E2E QoS (i.e., E2E Latency, Throughput, and Unavailability).

Given a set of z IoT traffics, we compute for each IoT traffic τ: the E2E Latency (denoted L_{E2E_τ}), the E2E Throughput (denoted T_{E2E_τ}), the E2E Unavailability (denoted U_{E2E_τ}). We also compute the resource usage associated with the deployment of the TCFs (RU_{E2E}) and the cost related to the execution of the scaling actions ($Cost_{E2E}$).

E2E Latency model. As per [Stiliadis 1998], we can easily calculate the E2E Latency as the sum of all the local Latencies for IoT traffic τ on the n nodes.

$$L_{E2E_\tau} = \sum_{i=1}^{n} L_{i_\tau} \tag{3.3}$$

In Equation 3.3, we assume a zero-latency for the IoT traffic τ if the benefit η_i (i.e. the sum of the benefits induced by all the supported TCFs and the scaling actions on the node i) is greater than 100. Otherwise, the Latency on the node i is $(1 - \eta_i\%)$ of the monitored Latency.

$$L_{i_\tau} = \begin{cases} 0 & \text{if } \eta_i \geq 100 \\ \delta_i \times (1 - \eta_i\%) & \text{else} \end{cases} \tag{3.4}$$

$$\text{with } \eta_i = \textstyle\sum_{q=0}^{p} f_q + \sum_{c=0}^{m} a_c$$

$$q \in F_i \text{ and } c \in A_i$$

E2E Throughput model. The E2E Throughput is the minimum of all the Throughputs crossed by the IoT traffic τ.

$$T_{E2E_\tau} = min(T_{1_\tau} \dots T_{n_\tau}) \tag{3.5}$$

We assume that ζ_i is the sum of the benefits to the Throughput induced by all the supported scaling actions on the node i. The Throughput on the node i is then the monitored Throughput added to ζ_i, if no Scheduler is deployed or if node i does not support Scheduler deployment. When a Scheduler is on the node i, the Throughput is $\omega_i\%$ of the monitored Throughput added to ζ_i.

$$T_{i_\tau} = \begin{cases} \rho_{i_\tau} - \zeta_i + 1 & \text{if } \nexists \text{ scheduler} \vee \text{scheduler} \notin F_i \\ (\rho_{i_\tau} - \zeta_i + 1) \times \omega_i & \text{else} \end{cases} \tag{3.6}$$

$$\text{with } \omega_i = 1 + \tfrac{1}{100} f_{scheduler} \text{ and } \zeta_i = \tfrac{1}{100} \textstyle\sum_{c=0}^{m} a_c$$

E2E Unavailability model. The E2E Unavailability is the sum of the Unavailability of the n nodes for the IoT traffic τ.

$$U_{E2E_\tau} = \sum_{i=1}^{n} U_{i_\tau} \tag{3.7}$$

We assume that the Unavailability of node i is zero if there is no Dropper deployed or if node i does not support the Dropper deployment. If there is a Dropper deployed on the first node (0), then the Unavailability is the rejection percentage associated with the IoT traffic τ. Otherwise, the Unavailability is the remaining availability multiplied by the rejection % associated with the IoT traffic τ.

$$U_{i_\tau} = \begin{cases} 0 & \text{if } \nexists \text{ dropper} \vee \text{dropper} \notin F_i \\ f_{dropper} & \text{if } i = 0 \\ (1 - \epsilon_i) \times f_{dropper} & \text{if } i > 0 \end{cases} \tag{3.8}$$

$$\text{with } \epsilon_i = 1 - \tfrac{1}{100} \textstyle\sum_{i'=0}^{i-1} U_{i'_\tau}$$

Scaling action execution's E2E Cost model. The $Cost_{E2E}$ is the sum of all the costs associated with the scaling actions execution on the node i ($Cost_i$).

$$Cost_{E2E} = \sum_{i=1}^{n} Cost_i \tag{3.9}$$

where

$$Cost_i = \begin{cases} \sum_{c=0}^{m} \Gamma_{i_c} & \text{if node } i \text{ support all} \\ & \text{the scaling actions} \\ \infty & \text{else} \end{cases} \tag{3.10}$$

TCFs deployment's E2E resource usage model. The RU_{E2E}, is the sum of all the resource usage associated with the deployment of TCFs on the nodes.

$$RU_{E2E} = \sum_{i=1}^{n} RU_i \tag{3.11}$$

Where

$$RU_i = \begin{cases} \sum_{q=0}^{p} (cpu_q(\lambda_i) + ram_q(\lambda_i)) & \text{if} \beta_i\% \leq 1 \\ & \wedge \gamma_i\% \leq 1 \\ \infty & \text{else} \end{cases} \tag{3.12}$$

$$\text{with } \beta_i = (\textstyle\sum_{q=0}^{p} cpu_q(\lambda_i) + H_{i_{cpu})} \times \sum_{c=0}^{m} a_c$$

$$\text{and } \gamma_i = (\textstyle\sum_{q=0}^{p} ram_q(\lambda_i) + H_{i_{ram})} \times \sum_{c=0}^{m} a_c$$

The CPU and RAM usage of the TCF depend on the request arrival rate λ on the node i.

3.5.2 Multiobjective Problem Formulation

We formulate in this Section a multiobjective optimization problem for efficient planning of the TCFs (proposed in Section 3.4) and scaling actions execution in the multi-constraint NIP set-up. Our goal in the formulated problem is to minimize the **ratio** between the IoT traffic's QoS requirement and the QoS provided (E2E Latency, E2E Availability, and E2E Throughput) by the NIP. The k-objectives problem is formulated as:

$$\begin{aligned} &\text{minimize} && F = l_1, \ldots, l_z, t_1, \ldots, t_z, u_1, \ldots, u_z \\ &\text{subject to} && l_\tau \leq 1, \forall \tau \in [1, ..., z], \\ &&& t_\tau \leq 1, \forall \tau \in [1, ..., z], \\ &&& u_\tau \leq 1, \forall \tau \in [1, ..., z] \end{aligned} \tag{3.13}$$

Where we have

$$l_\tau = \frac{L_{E2E_\tau}}{L_{Qos_\tau}} \tag{3.14}$$

$$t_\tau = \frac{T_{Qos_\tau}}{T_{E2E_\tau}} \tag{3.15}$$

$$u_\tau = \frac{U_{E2E_\tau}}{U_{Qos_\tau}} \tag{3.16}$$

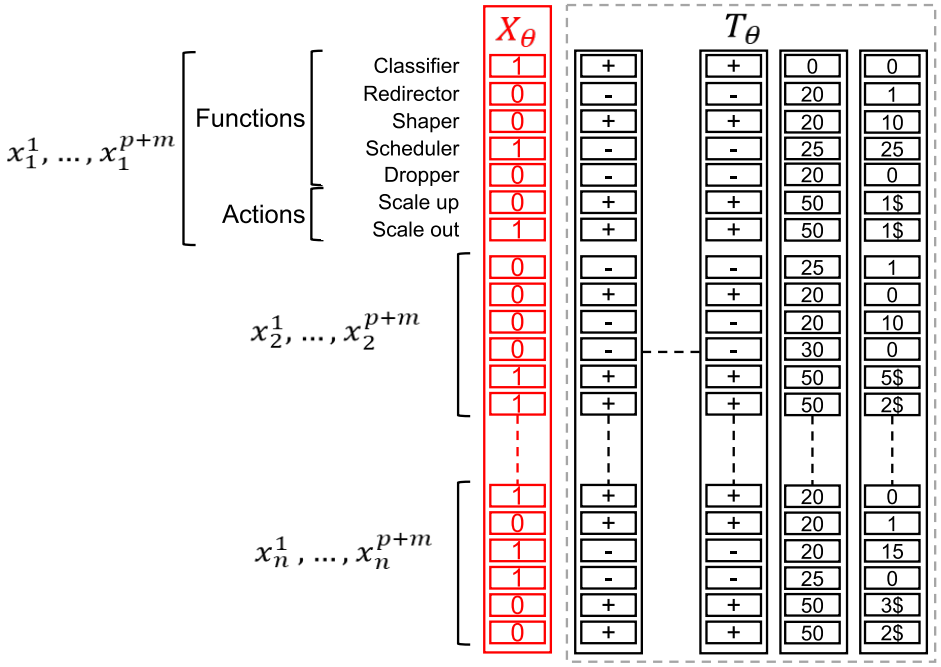
3.5.3 GA-based Constrained Optimization Model

In this Section, we define the "individuals" structure (chromosome). A chromosome is a solution that combines the execution of scaling actions and TCFs deployment to sustain QoS. Additionally, we consider the following genetic operators: mutation and crossover.

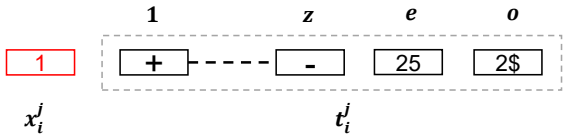
Genotype. The solutions are represented in a way that they can be easily understood and manipulated. We define a chromosome as a binary vector X_θ to describe the application of TCFs or scaling actions to the NIP's nodes (See Fig. 3.7a). Each X_θ is associated with an integer matrix T_θ that contains the TCF or scaling actions' additional information.

$$X_\theta = [x_1^1, \ldots, x_1^{p+m}, \ldots, x_n^1 \ldots x_n^{p+m}] \tag{3.17}$$

Particularly, each decision variable $x_i^j = 1$ if and only if the j^{th} TCF or scaling action is applied to NIP's node i. Each decision variable x_i^j is associated with a configuration vector



(a) Chromosome X_θ.



(b) Illustration of x_i^j and its associated t_i^j.

Figure 3.7: Genotype Representation.

named t_i^j. The vector t_i^j is represented in Fig. 3.7b and contains the following information:

- $t_i^j[1 \ldots z]$: denotes the decision variable x_i^j effect on all IoT traffics.

$$t_i^j(\tau) = \begin{cases} +1 & \text{if } x_i^j \text{ improves the QoS of } \tau \\ -1 & \text{else} \end{cases} \tag{3.18}$$

- $t_i^j[e]$: denotes the proportion of the decision variable x_i^j effect. For instance, $t_i^j[e] = 25$ means that the decision variable x_i^j can reduce (if $t_i^j[\tau] > 0$) or increase (if $t_i^j[\tau] < 0$) the IoT traffic τ Latency by 25%.
- $t_i^j[o]$: denotes information to each the decision variable x_i^j.
  - In the Redirector gene, $t_i^j[o]$ denotes the number of hops for the class IoT traffic.
  - In the Shaper gene, $t_i^j[o]$ denotes the delay time for the class IoT traffic.

- In the Scheduler gene, $t_i^j[o]$ denotes the scheduling rate for the class IoT traffic.
- In the scale up and scale out genes, $t_i^j[o]$ denotes the cost in USD.

Genetic operators. We consider the classic operators that are enough to create and maintain the genetic diversity by combining existing solutions into new solutions and to select between solutions:

- Bit-flip – acts independently on each bit in a solution and changes the value of the bit (0 to 1 and vice versa) with probability M_p, where M_p is a parameter of the operator. The most commonly prescribed value for this parameter is $M_p = 1/l$.
- Tournament Selector (Selection) – selects an individual from a population of individuals by running several "tournaments" among a few individuals randomly chosen from the population. This operator selects the winner of each tournament (the one with the best fitness) for crossover.
- Half Uniform Crossover (HUX) – swaps the half of the non-matching bits of two solutions according to a probability C_p. For this purpose, HUX first calculates the number of different bits (Hamming distance) between the parents. Half of this number is the number of bits exchanged between parents to form the two children.

3.5.4 Evolutionary Strategies and Pareto Front analysis

The proposed model above is the starting point in the implementation of a Genetic Algorithm to optimize the QoS parameters of IoT traffics (Latencies, Throughputs, Availabilities), resource usage of TCFs, and cost of scaling actions. In this Section, firstly, we present the adopted evolutionary strategy to compare individuals. Secondly, we offer a discussion on the choice of the solution in the Pareto front to apply.

Evolutionary strategy. We adopted the evolution strategy for QoS4NIP planner based on the Hyper-volume calculated from Pareto fronts found by the main algorithms in the literature and compatible with the formulated problem. We consider the Non-dominated Sorting Genetic Algorithm II (NSGAII [Deb 2002]), III (NSGAIII [Deb 2014]), and the Strength Pareto Evolutionary Algorithm 2 (SPEA2 [Zitzler 2001]). The Hyper-volume indicator measures the volume of the dominated portion of the objective space. It is of exceptional interest, as it possesses a highly desirable feature called strict Pareto compliance. This feature means that whenever one approximation completely dominates another approximation, the Hyper-volume of the former will be higher than the Hyper-volume of the latter.

The largest Hyper-volume was obtained by NSGAII, as shown in Fig. 3.8. The outperformance of NSGAII on NSGAIII is explainable since our problem is type Knapsack Problems

(KP). As clearly demonstrated in [Ishibuchi 2016], on multiobjective KP, NSGAII outperformed NSGAIII. NSGAII will be used for validation purposes in the rest of this Chapter. The reader may see [Deb 2002] for further details about the NSGAII algorithm.

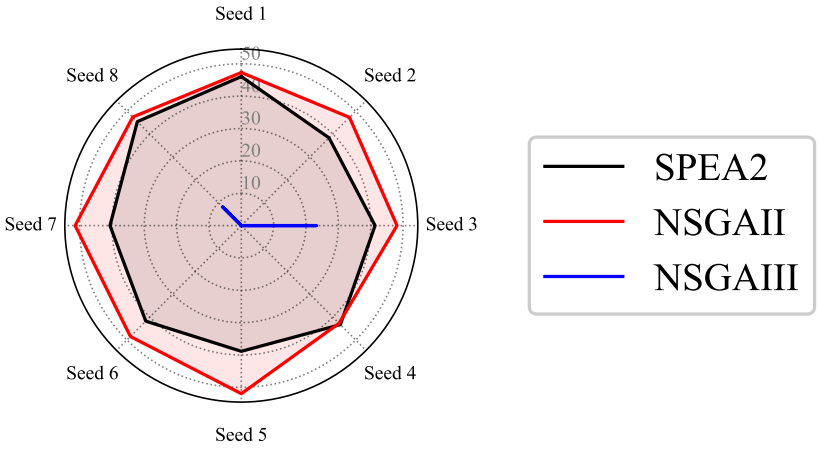


Figure 3.8: Hyper-volume measure on the formulated problem with $C_p = 100\%$; $M_p = 100\%$; N = 200, $l = 28$.

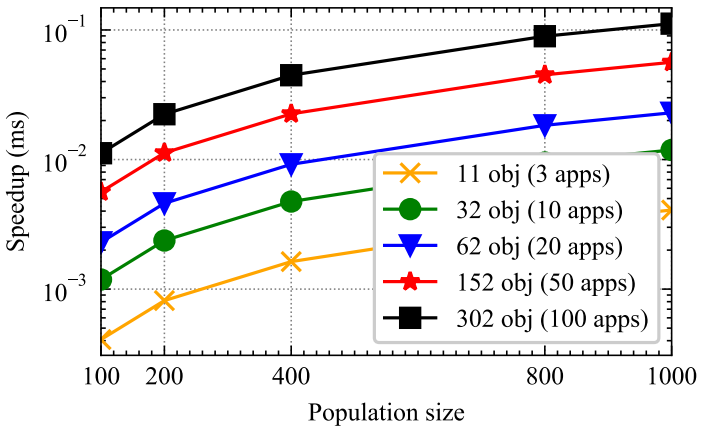


Figure 3.9: Speedup achieved by the NSGAII-based evolutionary strategy on the formulated problem for 3, 10, 20, 50 and 100 IoT traffics (on a Processor Intel (R) Core (TM) i7-7500U CPU @2.70GHz).

Discussion on the choice of the applied solution. As earlier stated, the presence of multiple objectives in a problem, in principle, gives rise to a set of optimal solutions. None of these Pareto-optimal solutions can be considered better than the others in the absence of additional information. In our context, once the GA finds a Pareto front, a choice must be made to apply a unique solution to the NIP. We recommend three methods of selection.

The first method is the *Random Selection*, which consists of choosing a solution randomly from the Pareto front. This method is a proper selection since each solution, X_θ, of the Pareto front has the same probability of being applied to the NIP.

The second method is the *QoS objectives-based Selection*. This method consists of selecting a solution to apply to the NIP based on the ranking or weighting of the QoS. For instance, some non-dominated solutions in the Pareto front lead to request losses for some IoT traffics. The choice will be towards the solution that discards nothing, even if it proposes higher Latencies ($\leq L_{qos}$).

The last method of selection is the *Non-QoS objectives-based Selection*. The selection of the solution to be applied to the NIP is based on Non-QoS criteria, such as the number of scaling actions and TCFs required by the solution (Complexity-based), or the cost and resource usage associated with each solution (Cost-based). The following case study on Connected Vehicles will use this last method (Cost-based).

3.5.5 The QoS4NIP Planner Algorithm

The general workflow of the QoS4NIP planner (NSGAII-based) presented in Algorithm 6 is as follows.

Algorithm 6: QoS4NIP Planner

```
   // N: Population size
   // T: Maximum number of generations
   // X_θ: Solution
   Input: N; T
   Output: X_θ
1  begin
2      Set t = 0 Initialize P_0 and set Q_0 = ∅.
3      while t < T do
4          Calculate fitness for P_t and assign rant based on Pareto dominance
5          Perform selection on P_t to fill the mating pool
6          Apply crossover and mutation operators to obtain the offspring population Q_t
7          Select the best N non-dominated solution from P_t ∪ Q_t by the two-step procedure to form P_{t+1}
8          Set t = t + 1
9      Set j = 0
10     while j < N do
11         Calculate and save RU_E2E[j] for P_t[j]
12         Calculate and save Cost_E2E[j] for P_t[j]
13         Set j = j + 1
14     indexes ← arg min_{j=1...N}{Cost_E2E(P_t[j])}
15     index ← arg min_{j∈indexes}{RU_E2E(P_t[j])}
16     X_θ ← P_t[index]
17     return X_θ
```

From lines 2 to 4, the population is initialized randomly, where every individual's structure is as proposed in Fig. 3.7a. Then, the fitness value of every solution in the current population is computed using Equations 3.3, 3.5, 3.7 and the monitoring information (cf. Equations 3.4 and

3.6). All the individuals of the current population with penalties values are discarded. Once the fitness is assigned, the population is sorted according to the non-domination individual. Line 5, the Tournament selector, is applied to the entire population to determine the fittest individuals of the current population placed into the mating pool. Line 6, new solutions, called offspring, are generated by applying Bit-Flip Mutation and Half Uniform Crossover to the mating pool. Line 7, based on the values provided by the ranking scheme, the best individuals from the combination of the current population P_t and the offspring pool Q_t are detected. Those with a lower value (min) or higher crowding distance are saved in the following population P_{t+1}. The crowding distance mechanism is used to preserve the diversity of solutions. It estimates the volume of the hyper-rectangle defined by two nearest neighbors [Deb 2002]. Suppose some candidate solutions are of the same rank, and not all can enter the following population. In that case, the less crowded individuals from a given rank are selected to fit the future population. From lines 9 to 13, the Pareto Front's scaling action cost and resource usage are calculated. From lines 14 to 17, using the selection method described in Section 3.5.4, the cheapest solution (X_θ) is returned. The heuristic time complexity is $\mathcal{O}(MN^2)$, where M is the number of objectives, and N is the population size. The plots in Fig. 3.9. have been drawn in logarithmic scales. They show the speedup in ms as a function of population size.

3.6 Evaluations in a Connected Vehicles Case Study

Most of the data required by Connected Vehicles can be transferred using short-distance communications. However, numerous use cases depend on information that is not obtainable within proximity. For these longer communication paths, the cellular network could be a potential solution for communication between vehicles and vehicles to the network itself, so-called vehicle-to-network (V2N) communication. As shown in Fig 3.10, we consider three realistic V2N IoT traffics with different QoS requirements [Boban 2018]. We carried out simulations to evaluate the effectiveness of the proposed approach against others.

3.6.1 Compared Schemes

The relative performance comparison of the proposed scheme (QoS4NIP) has been carried out against four other schemes. The first is the standard First-Come-First-Served (FCFS)-based approach. Unlike the proposed scheme, this approach does not emphasize maintaining QoS requirements. The second is the autoscaling scheme. To not bias the results, we compared our scheme with the autoscaling approach without considering a particular implementation in the literature while trying to show its limits. To do this, the compared autoscaling scheme is obtained by tuning our planning algorithm to use the scaling actions only and switching off VNF and ANF deployment.

In our comparison, we also want to distinguish the costs induced by VNF usage versus ANF usage, independently of autoscaling. For this purpose, we implemented the two other schemes as two variants of QoS4NIP, wherein one variant only deploys VNFs, while the latter, additionally, considers using ANFs on the IoT End Gateways. In the following, the FCFS, the autoscaling scheme, and the considered variants of QoS4NIP are referred to as FCFS, AS, QoSEF, and QoSEFe, respectively.

3.6.2 Simulation Setup and Evaluation Parameters

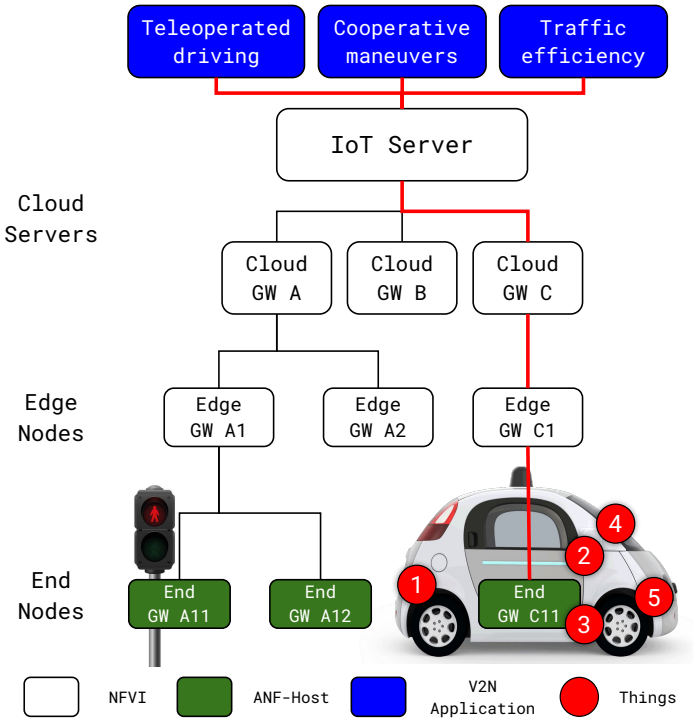


Figure 3.10: Considered topology for the case study.

Table 3.2 shows *Teleoperated driving*, *Cooperative maneuvers* and *Traffic efficiency* QoS requirements. All these V2N IoT applications communicate with the actuators and sensors in the vehicle through "IoT Server", Cloud "GW C", Edge "GW C1" and End "GW C11." In each test case, the platform is modeled by a snapshot, s_0, where no TCF is deployed, and no scaling action is executed. We implemented all the compared schemes (AS, QoS4NIP, QoSEF, QoSEFe) in Python using the multiobjective Evolutionary Algorithms library Platypus [Hadka 2017].

Using the results of our previous work [Ouedraogo 2018b, Ouedraogo 2018a], we show the *a priori* benefit of each TCF presented in Table 3.3. The scaling actions benefits are considered, as shown in the work [Hwang 2015b]. For the resource usage parameters (CPU and RAM), we rely on the performance model of ANF and VNF presented in Section 3.4.

V2N Application	Description	QoS Requirements		
		T	L	A
Teleoperated driving	An external operator drives the vehicle using a live-stream video.	25	20	99
Cooperative maneuvers	A set of vehicles communicating and behaving as a system for performing coordinated actions.	10	100	99
Traffic efficiency	Optimization of traffic parameters (traffic lights, speed limit, etc.).	10	1000	90

Table 3.2: Representative V2N applications. T= Throughput in req/sec (request size = 1Mb); L= Latency in ms; A= Availability in %.

The four considered schemes for comparison (AS, QoSEF, QoSEFe and QoS4NIP) are initialized with the snapshot of the FCFS scheme, s_0 (presented in Table 3.4), corresponding to a number of objectives = 9, $N = 200$, $C_p = 100\%$, and $M_p = 1$; with $n = 4$, $p = 5$, $m = 2$, and $l = 28$ (i.e. $n \times (p + m)$). The Server and the Cloud "GW C" are experiencing CPU and RAM bottlenecks - their resource usage ($H_{i_{ram}}$ and $H_{i_{cpu}}$) are 90%. The Edge "GW C1" and the End "GW C11" are experiencing low resource usage - their resource usage ($H_{i_{ram}}$ and $H_{i_{cpu}}$) are 10%. The resource usage of the VNFs is [20-30]%. The resource usage of the ANFs is [5-10]%. The scale up and scale out cost per node is fixed to 0.3 USD (corresponding to an "AWS r4.large" price in march 2020).

	Benefit	Description
Classification	0%	Deploy a classifier on an node
Redirection	0%	Deploy a redirector on an node
Scheduling	35% [Ouedraogo 2018b]	Deploy a scheduler on an node
Shaping	35% [Ouedraogo 2018b]	Deploy a shaper on an node
Dropping	41% [Ouedraogo 2018b]	Deploy a dropper on an node
Scale out	50% [Hwang 2015b]	Replicate an node
Scale up	50% [Hwang 2015b]	Double resources of an node

Table 3.3: Benefits parameter settings

	QoS offered to applications at s_0					
	Teleoperated driving		Cooperative maneuvers		Taffic efficiency	
	L (ms)	T (req/sec)	L (ms)	T (req/sec)	L (ms)	T (req/sec)
Server	5	25	30	10	100	10
Cloud "GW C"	10					
Edge "GW C1"	5	20				
End "GW C11"		25				

Table 3.4: Initial snapshot s_0 parameter settings

3.6.3 Evaluation Metrics

The reconfiguration plan, we refer to here, are those proposed by the solutions associated with the different schemes. The evaluation metrics used to assess the proposed approach are defined

as follows:

- E2E Actions Cost and E2E Resource Usage: respectively, the End-to-End costs computed from Equation (3.9) and the End-to-End resource usage to sustain the QoS computed from Equation (3.11).
- E2E Latency: End-to-End solutions Latency of *Teleoperated driving, Cooperative maneuvers and Traffic efficiency*, computed from Equation (3.3).
- E2E Availability: End-to-End solutions Availability of *Teleoperated driving, Cooperative maneuvers, and Traffic efficiency* is 1 minus the End-to-End solutions Unavailability (denoted U_{E2E_τ}), computed from Equation (3.7).
- E2E Throughput: End-to-End solutions Throughput of *Teleoperated driving, Cooperative maneuvers and Traffic efficiency*, computed from Equation (3.5).

3.6.4 Observations

This part discusses the results we obtained. We compare the E2E Actions Cost, the E2E Resource Usage, the E2E Latency, the E2E Availability, and the E2E Throughput in the FCFS scheme with the results obtained from the schemes AS, QoSEF, QoSEFe, and QoS4NIP.

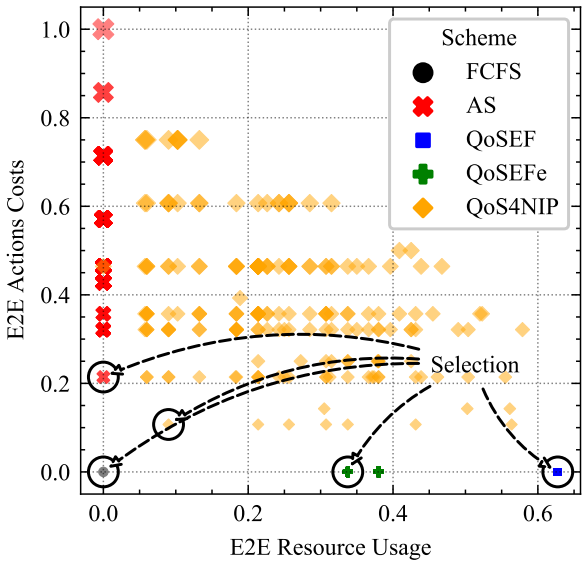


Figure 3.11: Relative E2E Reconfiguration Cost and Resource Usage.

The E2E Actions Cost and E2E Resource Usage. Fig. 3.11 shows the obtained Pareto Front. The associated cost in the FCFS scheme is 0 because no TCF is deployed, and no action is currently performed on the considered NIP set-up. In the AS scheme, the cost ranges from

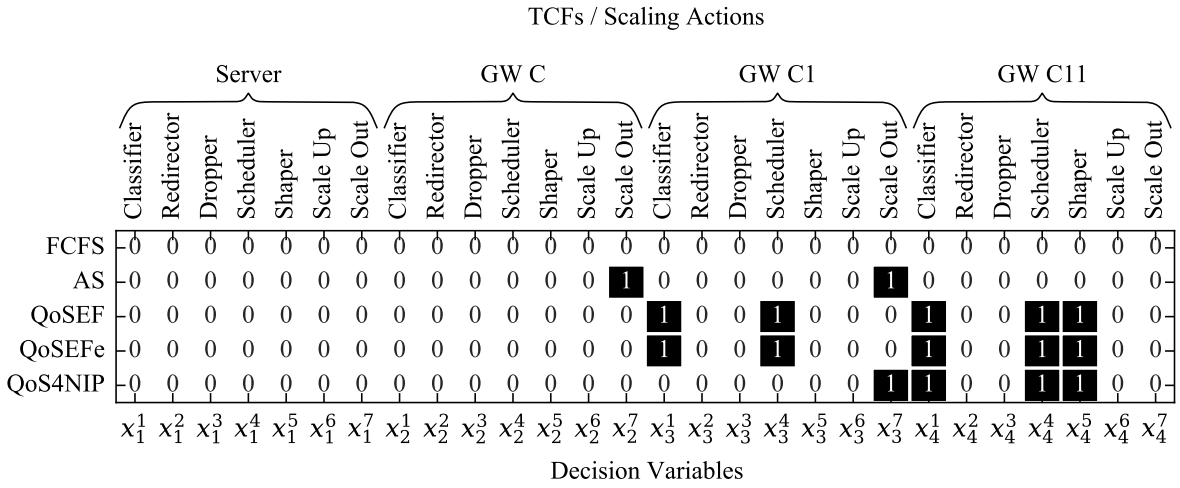


Figure 3.12: Selected E2E Reconfiguration Plan (X_θ).

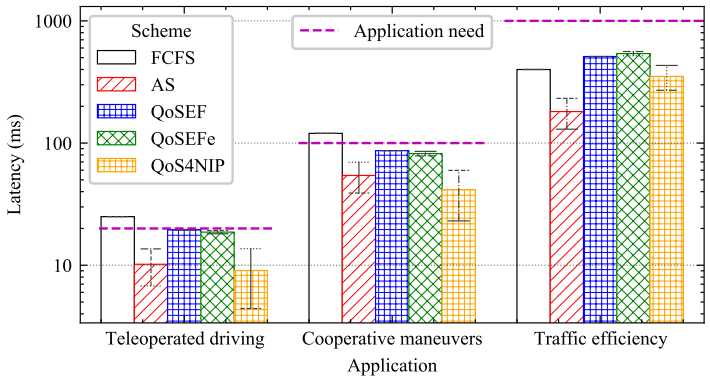


Figure 3.13: E2E Latencies.

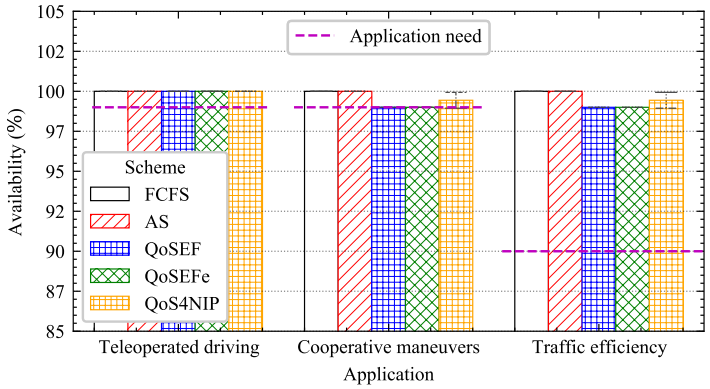


Figure 3.14: E2E Availability.

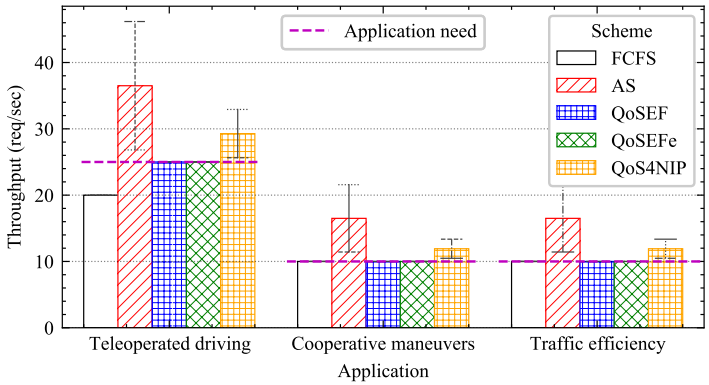


Figure 3.15: E2E Throughputs.

0.21 to 1.0 (0.25 to 1.2 in USD), and the resource usage remains 0 since no TCF is currently deployed on the considered NIP set-up. The QoSEF scheme does not induce any cost, and resource usage ranges from 0.62 to 0.63. In the QoSEFe scheme, using ANFs, resource usage has been reduced to range 0.34 to 0.38. In the QoS4NIP scheme, by combining the AS scheme and the QoSEFe scheme, the resource usage is between 0.1 and 0.58, and the cost is between 0.1 and 0.76 (i.e., 0.12 and 0.912 in USD).

The Cost-based solution selection, discussed in Section 3.5.4, is applied for the FCFS, AS, QoSEF, QoSEFe and QoS4NIP schemes. In general, the $(Cost_{E2E}, RU_{E2E})$ are, respectively $(0,0)$, $(0.21,0)$, $(0,0.62)$, $(0.34,0)$, and $(0.1,0.1)$. The cost of the AS scheme is about two times higher than QoS4NIP (i.e., we have 50% financial cost-saving). Fig. 3.12 shows the selected E2E reconfiguration plan of each scheme. The FCFS scheme is to do nothing. The AS scheme performs scale Out Cloud "GW C" and Edge "GW C1". The QoSEF scheme deploys the following VNFs: on the Edge "GW C1" a Classifier, a Scheduler; on the End "GW C11" a Classifier, a Scheduler and a Shaper. The QoSEFe scheme deploys the following ANFs: on the End "GW C11", a Classifier, a Shaper, a Scheduler; and, on the Edge "GW C1", two VNFs: a Classifier, and a Scheduler. The QoS4NIP scheme performs scale Out on the Edge "GW C1". It deploys, on the End "GW C11", the following ANFs: a Classifier, a Shaper, and a Scheduler.

QoS provided by the optimized reconfiguration plans. Fig. 3.13 shows the provided E2E Latencies by the optimized reconfiguration plan of QoS4NIP versus other schemes. In the FCFS scheme, the E2E Latency for *Teleoperated driving* is 25 ms and 120 ms for *Cooperative maneuvers*, which does not meet their requirements, 20 ms and 100 ms, respectively. Only in this scheme, the *Traffic efficiency's* required E2E Latency is reached (400 ms $\leq$ 1000 ms). In the other schemes (AS, QoSEF, QoSEFe, QoS4NIP), the E2E Latencies required by the IoT traffics are sustained. However, we observe that the AS scheme provides much more than what is required by the IoT traffics. For instance, for *Teleoperated driving*, the AS scheme provided 10 ms E2E Latency, which is less than what is supported by the IoT traffic, and that is where

we see that the QoSEF, QoSEFe, and QoS4NIP schemes do better. Only by differentiating the processing between the IoT traffics, the schemes QoSEF and QoSEFe make it possible to answer the required E2E Latencies of all the IoT traffics. The result is an increase in the E2E Latency of the *Traffic efficiency* ($\approx$ 600 ms), which always remains under the tolerable E2E Latency limit (under 1000 ms). The proposed QoS4NIP scheme provided the best E2E Latencies, except for *Traffic efficiency*, where the AS scheme provided low E2E Latency (200 ms).

Fig. 3.14 plotted the proposed E2E Availability by the optimized reconfiguration plan of QoS4NIP versus other schemes. In the FCFS and AS schemes, the E2E Availability is 100%. The fact that these schemes do not deploy droppers explains this value. However, in schemes QoSEF, and QoSEFe, the E2E Availability is 90% for *Cooperative maneuvers* and *Traffic efficiency*, due to the use of a dropper (rejecting 10% of the targeted traffic). In QoS4NIP, the E2E Availability is 99% for *Cooperative maneuvers* and *Traffic efficiency*, due to the use of a dropper (rejecting 1% of the targeted traffic). *Teleoperated driving* being of the most demanding in QoS, its traffic is not dismissed. The E2E Availability provided by all schemes always remains under the tolerable threshold.

Fig. 3.15 shows the provided E2E Throughput of QoS4NIP versus other schemes. E2E Throughput required by *Teleoperated driving* is not met in the FCFS scheme. In the AS scheme, the provided E2E Throughput is much higher than the IoT traffic's requirement, which is not a cost-optimal plan. The schemes based on differentiation (QoSEF, QoSEFe, and QoS4NIP) use schedulers and provide the closest E2E Throughput regarding the IoT traffic's requirements. For instance, the QoS4NIP scheme provided to the *Teleoperated driving* an E2E Throughput of 25 req/s which is required.

We can conclude from these simulations that the available resources can limit the QoSEF and the QoSEFe scheme's effectiveness in the NIP set-up. The AS scheme is effective but has not optimal costs. The QoS4NIP seems to be the best way to enable QoS for NIPs by taking advantage of the service differentiation and the autoscaling combination to overcome the above limitations of both schemes separately considered.

3.7 Considered hypotheses

We make the following considerations about the problem at hand. First, the NIP's nodes in the Cloud/Edge are VMs and can be easily scaled (up and out). The NIP's nodes at the network End (End Gateways) are mainly hardware nodes. In rare cases, an End Gateway can be a VM located in a data center. Since the VMs are in the Cloud/Edge, the physical server's available capacity is supposed unlimited, as considered in the literature. For example, some IaaS providers are now proposing Cloud/Edge joint offers, where the limited capacity in the Edge data center is mitigated through continuous offloading to Cloud data centers. Oppositely, we consider that all reconfiguration plans generated by QoS4NIP must respect the limitation of resource

capacities inside the NIP's nodes. Second, we consider only the NIP-level QoS regardless of the underlying IP network performance (consisting of routers and switches). Thereby, the system model does not consider the network-level Latency. Third, the scaling decisions are considered binary since QoS4NIP aims to minimize the scaling cost. "Zero," meaning no scaling action is necessary, and "One" meaning a scaling action is unavoidable. This allows us to be accurate in our comparison by considering the "lowest boundary" of the autoscaling approach with a minimal cost of "one new instance" at once (the non-compressible cost). Finally, we assume that only one instance of any TCF can simultaneously run on a NIP's node, and when a scale out is applied to a NIP's node, the associated TCF will be deployed both on the initial instance and on the new replicate. We did not consider applying different TCFs during the scale out for the following reasons. We aim to maintain consistency in handling IoT traffic. When a node is scaled out, a load-balancer is deployed upstream of the node's instances. Upon the arrival of a request, this load-balancer redirects this request to any node instances with no distinction. Applying different TFCs to instances would lead to an inconsistency problem for the IoT traffic handled by that node. That would result in different processing rules for requests arriving at the same (scaled-out) node. Considering such a direction will break the standard management rules for resource scaling. Indeed, we assume that using the standard management rules for the scaling of the nodes, executing different TFCs in instances would be technically not sound: the scaling manager can delete any instances regardless of the TCFs executed. For these reasons, we consider, in our contribution, that any instance of a scaled-out node will process the arriving requests as decided by QoS4NIP regardless of the number of running instances. Considering the same TCFs in all instances of a given node allows us to be in line with the standard scaling approaches that proceed by deploying identical instances when scaling out a given node and by removing any instance when scaling-in.

3.8 Integration in the Autonomic Manager

In a real scenario, as described in Section 3.6, the proposed planner, called QoS4NIP, is located on top of the NIP's monitoring system. This follows the autonomic architecture model of [Kephart 2003]. QoS4NIP is invoked periodically and takes the monitoring information as inputs. The output of QoS4NIP is a reconfiguration plan represented by a binary vector. The configuration enforcement component [Kephart 2003] performs this reconfiguration and considers the current configuration. For instance, when the QoS4NIP reconfiguration plan includes deploying a given TCF on a particular NIP's node, and if this given TCF is already deployed, nothing happens. Otherwise, the TCF will be deployed. The same applies when the QoS4NIP reconfiguration plan does not include the deployment of a given TCF on a particular NIP's node (this TCF will be removed). QoS4NIP handles a scaled out/up node, virtually, as a *unique* node with i) resized resource in case of scaling-up, and ii) combined resources in case of

scaling-out.

3.9 Conclusion

We have proposed in this Chapter a new cost-effective approach combining the advantages of the Traffic Control Functions (TCFs) deployed as NFs and the autoscaling of the virtualized processing resources. 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 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 [oneM2M 2016]. We showed by emulating different scenarios of the domain of Connected Vehicles that the classical systematic scaling can be avoided while fitting the required End-to-End QoS requirements for both common and potentially critical IoT traffics. We considered the different QoS parameters (Latency, Throughput, and Availability) and the Cloud resource usage cost that we handled in a multiobjective optimization approach. We implemented TCFs that we deployed as Network Functions (NFs), which are appropriate to the capacity limits of the NIPs' nodes. We implemented a scheme, QoS4NIP, that efficiently combines the scaling actions and traffic management.

In the next Chapter, we take a deeper look at the bottleneck identification in IoT platforms. The aim is to analyze the root causes of the degradations to orientate the planner to search for a solution properly. The logic for answering this question will be implemented in the analyzer (see Fig. 3.16).

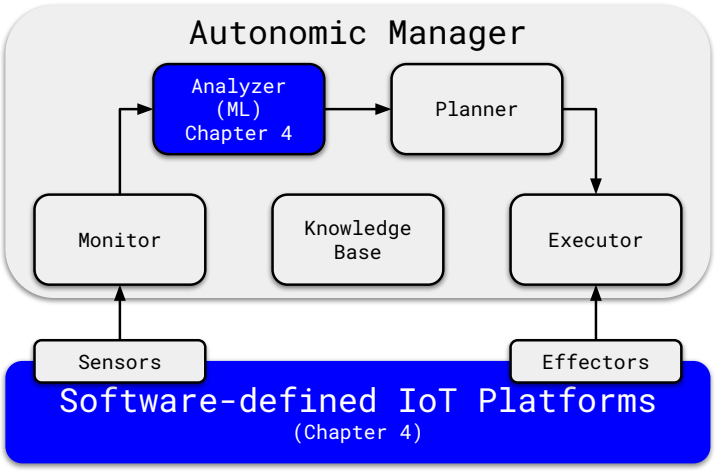


Figure 3.16: Building of the Analyzer in the Chapters 4.

CHAPTER 4

Adaptive Performance Analysis

4.1 Introduction

In general, meeting the strict QoS requirements of IoT applications through effective performance diagnosis remains an inescapable challenge [White 2017]. Indeed, the integration of IoT Platforms, traditionally vertical to shared horizontal platforms, gives rise to performance bottlenecks, challenging to detect and mitigate. 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[1] in terms of performances (e.g., CPUs satura-

[1]A bottleneck is a resource or an application component that limits the performance of a system [Gregg 2013]. [Malkowski 2009b] 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.

tions) associated with the resource of the NIP responsible for the assumed violation. This second step is known as the performance analysis step. This Chapter focuses on this second step, when a violation has already been detected using, for instance, methods presented in [Qiu 2018, Schmidt 2018, Li 2018, Yu 2019]. 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 observations. First, there are over 80 types of metrics available to monitor in an NIP deployed on a public cloud such as AWS (using EC2 VMs[2]). Second, these metrics induce not negligible monitoring overhead[3].

In an ideal scenario, the overhead of collecting data increases with a constant value per access. Following [Waller 2014], 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 [Wang 2018]. In this context, and considering a maximum overhead not to be exceeded (i.e., monitoring overhead 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 for an allocated monitoring overhead budget?"*

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

The significant contributions of this Chapter are summarized below.

- We model the problem of Multiple Bottlenecks Identification (MBI) 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.

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

[3] Monitoring overhead is the amount of additional usage of resources by monitored execution of a program compared to a regular (unmonitored) execution of the program. In this case, resource usage encompasses the utilization of CPU, memory, I/O. Monitoring overhead concerning execution time is the most commonly used definition of overhead

- We build a virtualized platform prototype implementing the experimental testbed to gather a training dataset. We also design the testbed to provide a training set that is representative of the real-world situation.

- We develop different supervised ML algorithms to identify the bottlenecks. We numerically evaluate these MBI models using the collected data.

- We implement the proposed SOMS to find which metrics should be considered for the efficiency of the NIP analysis while optimizing the performance of the MBI model, not to label as positive a sample that is negative and evaluate its performance.

The remainder of the Chapter is structured as follows. Section 4.2 presents an adaptive performance analysis use case to be considered and evaluated. Section 4.3 discusses the related work. Section 4.4 details the system model. Section 4.5 describes the proposed methodology to tackle the multiple bottlenecks identification problems in NIPs with an allocated monitoring overhead budget. Section 4.6 presents the experimental setup. Section 4.7 is devoted to the evaluation of the proposed approach. Section 4.8 describes how the proposed approach is implemented in a real scenario. Finally, our work results, its limits, and future work are discussed in the Conclusion Section.

4.2 Motivating use case

We present here 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. The monitoring overhead can be translated into a financial cost (i.e., the number of metrics observed proportional to the number of messages transmitted by second). In the Cloud-to-Thing continuum, [Brogi 2017], the availability and capacity of the resources, namely computation, storage, and connectivity, decrease when moving from the Cloud toward Things. Typically, the IoT End Gateways, located close to Things, are small devices with limited processing, storage, and connectivity capabilities. In this work, we consider the monitoring overhead is inversely proportional to available resources when moving from the Cloud toward Things (i.e., from the Cloud Server to the IoT End Gateways). We consider 3 situations where the allocated overhead budget fluctuates in time: *unlimited* budget, *modest* budget, and *austere* budget. As depicted in Fig. 4.1, we define the following scenarios based on the Chicago taxi trips dataset provided by the City of Chicago's open data portal[4]. 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

[4]Chicago data portal. https://data.cityofchicago.org

is severely limited in the third scenario (austere overhead budget between 0h-5h). Below we describe each scenario.

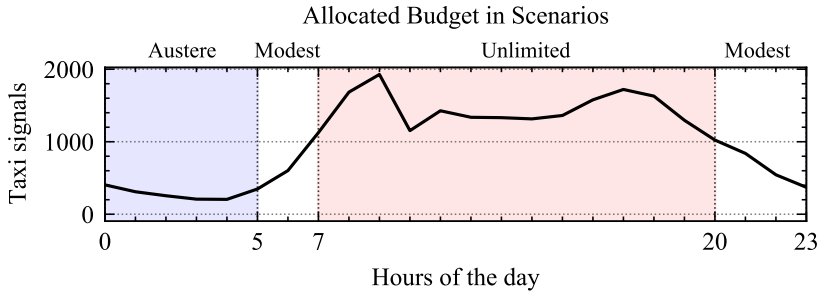


Figure 4.1: Chicago Millennium Park taxi signal counts by hour of the day for Monday, February 06, 2017.

- *Unlimited budget scenario:* We first investigate the case where the overhead budget is Unlimited. This scenario occurs during the rush hours in Fig. 4.1 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. 4.1). 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 selecting 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 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 lesser efficiency than in the previous scenario. This scenario occurs during the hours where the taxi signal number is lower than five hundred (see Fig. 4.1). 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 make adjustments accordingly to the monitoring budget.

4.3 State-of-the-Art

Several fields, such as traditional IP Networks [Yan 2012], Cloud Computing [Weng 2018], and Big Data [Zhou 2018], consider the multiple bottlenecks identification problem. Moreover, regarding NFV, most of the existing works consider the *fault detection problem* or the *fault recovery problem* in the fault management framework (see [Solé 2017] for more detail). Nevertheless, few works deal with the *fault localization problem* (i.e. bottlenecks identification problem). In this Chapter, since we only aim to contribute to this domain for the IoT context, we consider the reference contributions made in the literature. In the following, we present a literature review analysis on NFV-enabled IoT Platforms, including IoT which is an essential aspect of the proposed work.

Sauvanaud et al. propose, in [Sauvanaud 2016] and [Sauvanaud 2018], an approach to detect the 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 [Gonzalez 2017] 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 [Cui 2017] 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 [Cotroneo 2017b] 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 [Cotroneo 2017a] 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 [Cotroneo 2018] 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 [Zhang 2018] 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 [Mariani 2018] 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 compensates 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 [Pfitscher 2019] 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 [Tola 2019] 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 [Bouattour 2020] 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.

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

In that direction, our contribution's main originality consists of combining several changes in the traditional approach to handle bottlenecks identification problem. The first change (Section 4.5.1) consists of considering that multiple bottlenecks may arise among several resources in NIPs. The second change (Section 4.5.2) consists of considering adapting the monitored metrics 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.

4.4 System Model

In this Section, we propose a model for the considered system. For convenience, Table 4.1 lists the main notations.

Names	Meanings
B	Number of possible bottlenecks
C_t	Observation cycle
$\mathbf{D}$	Multi-bottleneck training set
F_k	Flow k of messages
h	Hypothesis to optimize
m	Number of samples
M_k	Message on F_k
N_k	Set of network functions composing a Path_k
O_k	Monitoring overhead of every performance metric
$o_{p,n}$	Value of the monitoring overhead associated to the metric p on nf$n$
p	Performance metrics p
P	Number of performance metrics
Ψ	Optimization criterion
S	Set of metrics without a metric $\theta_{p,n}$
Θ_k	Decision variable regarding which performance metrics is actually monitored
$\theta_{p,n}$	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
$x_{p,n}$	Mean value of the time series associated to the metric p on nf$n$ during a Ct
Y	True Bottlenecks
$\hat{Y}$	Diagnosed Bottlenecks
fn_j	False negative of the j-bottleneck
fp_j	False positive of the j-bottleneck
nf$n$	NF n in N_k
Path_k	Path k
tn_j	True negative of the j-bottleneck
tp_j	True positive of the j-bottleneck

Table 4.1: Notations

4.4.1 NIP Model

In our work, we handle the NIPs that implement the common reference architectures, such as oneM2M [oneM2M 2016]. We consider that NFV-I in the Cloud/Fog/Edge node Virtualized

Network Functions (VNF), Application Network Function(ANF) [Ouedraogo 2020] and Physical Network Functions (PNF) offering the NIP service to the IoT Application and IoT devices.

Fig. 4.2 depicts the NIP model used in this Chapter. 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, oneM2M). To facilitate the presentation of the performance analysis system, we define an 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; $\text{Path}_k = \{nf_1, nf_2, \ldots, nf_{N_k}\}$. 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.

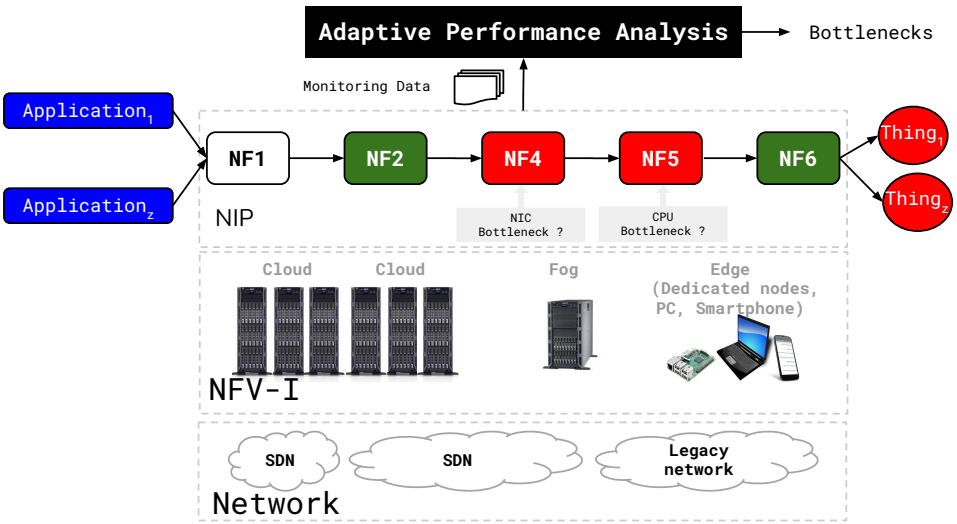


Figure 4.2: System Model

4.4.2 Performance Monitoring Model

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

$$X_k = \begin{array}{c} \begin{matrix} nf_1 & nf_2 & \cdots & nf_{N_k} \end{matrix} \\ \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P,1} & x_{P,2} & \cdots & x_{P,N_k} \end{pmatrix} \end{array} \tag{4.1}$$

where:
$x_{p,n}$= is the mean[5] value of the time series associated to the metric p on node n during a cycle C_t.

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

$$\Theta_k = \begin{array}{c} \begin{matrix} nf_1 & nf_2 & \cdots & nf_{N_k} \end{matrix} \\ \begin{pmatrix} \theta_{1,1} & \theta_{1,2} & \cdots & \theta_{1,N_k} \\ \theta_{2,1} & \theta_{2,2} & \cdots & \theta_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{P,1} & \theta_{P,2} & \cdots & \theta_{P,N_k} \end{pmatrix} \end{array} \tag{4.2}$$

where :

$$\theta_{p,n} = \begin{cases} 1 & \text{if the performance metric } p \text{ on nf}_n \text{ is monitored} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

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

$$O_k = \begin{array}{c} \begin{matrix} nf_1 & nf_2 & \cdots & nf_{N_k} \end{matrix} \\ \begin{pmatrix} o_{1,1} & o_{1,2} & \cdots & o_{1,N_k} \\ o_{2,1} & o_{2,2} & \cdots & o_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ o_{P,1} & o_{P,2} & \cdots & o_{P,N_k} \end{pmatrix} \end{array} \tag{4.4}$$

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

[5]For simplicity, we consider the mean value among a wide variety of others statistics extracted from the time series.

4.5 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 [Malkowski 2009b]. Indeed several works, such as [Battré 2010, Malkowski 2009a], consider an approach allowing to analyze multiple bottlenecks in a single run. We propose to explore this approach in this work.

Our proposed method is intended to overcome the limitations described in Section 4.7. 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 employed. Based on an MLC Algorithm, a feature selection wrapper algorithm (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 [John 1994], 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

$$g(\Theta_k) > g(S) \tag{4.5}$$

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

$$\begin{gathered} g(\Theta_k) = g(S) \text{ , and} \\ \exists S' \subset S \text{ , such that } g(\Theta'_k) > g(S') \end{gathered} \tag{4.6}$$

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

$$\forall S' \subseteq S, g(\Theta'_k) \leq g(S') \tag{4.7}$$

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 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. 4.3, the proposed methodology is as follows.

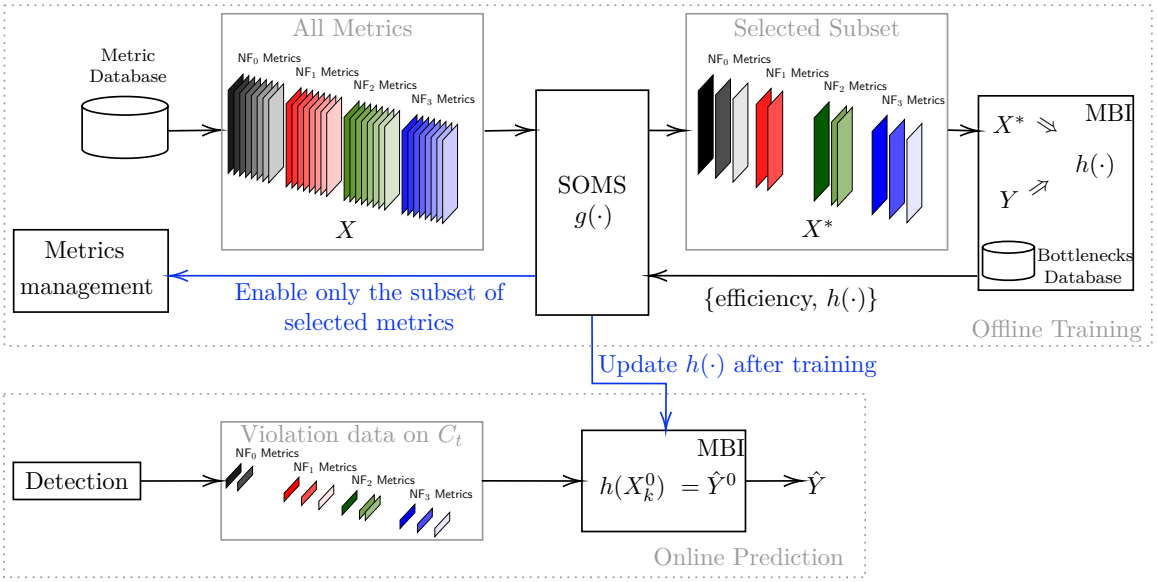


Figure 4.3: Adaptive Performance Analysis Method

Off-line Training In a supervised learning approach, there is a training step. In this step, the Adaptive Performance Analysis infers two functions $g(\cdot)$ (e.i. SOMS) and $h(\cdot)$ (e.i. MBI) from the training dataset. In the SOMS Algorithm (see Algorithm 7) each new subset is used to train and test a MBI model. Training a new model for each subset is computationally intensive but provides the best performance [Jović 2015] and is the only approach directly applicable to multilabel dataset [Tsoumakas 2009]. 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 MBI.

Online Prediction Once the optimal subset is found in the training step; the predictions are made online. When the *Monitoring* (see Chapter 2) component catches a QoS violation,

the corresponding data on the violation is gathered, and the Online MBI is invoked to identify the Bottlenecks.

4.5.1 Multiple bottlenecks identification (MBI)

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. 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 MBI 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 MBI model (see Section 4.7.1).

4.5.2 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 SOMS. The proposed SOMS Algorithm select a subset of relevant metrics for a given overhead budget. Formally, SOMS solve the following optimization problem:

$$\begin{aligned} \text{optimize} \quad & g = \frac{1}{m} \sum_{i=1}^{m} \Psi(Y^i, h(X_k^i \odot \Theta_k)) \\ \text{subject to} \quad & \omega_{\text{admin}} \geq \omega \end{aligned} \tag{4.8}$$

where:

- ω_{admin} is the overhead budget tolerated by the NIP administrator for a flow F_k.
- ω (see Eq. 4.9) is the total monitoring overhead for a flow F_k,

In Eq. 4.8:

$$\omega = \sum_{p=1}^{P} \sum_{n=1}^{N_k} (\Theta_k \odot O_k)_{p,n} \tag{4.9}$$

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 [Natarajan 1995]. For this problem, we propose a heuristic based on the Forward Sequential Selection search strategy [Reunanen 2006] that has been proven to constitute an efficient method to provide suitable near-optimal solutions in a short amount of time (see Section 4.6). This strategy follows a wrapper approach [Kohavi 1997]. The general work-flow of the SOMS Algorithm is presented in Algorithm 7.

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 line 7 to 10, add the p metric on node n, evaluate the combination. In line 12, find the best combination. From line 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 metric 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 4.9 with Θ_k and O_k. From line 36 to 39, can one afford the selected metrics? If yes, cross-validate the MBI model $h(\cdot)$ (see Section 4.5.1) with the combination of metrics and return the score; if no, return a penalty score.

4.6 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.

Algorithm 7: Simple Overhead-sensitive Metrics Selection

```
   // h: MBI 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
 1 begin
 2   Θ_k ← 0_{P,N_k}
 3   k ← 0
 4   S_b ← ∅
 5   while r < P × N_k do
 6     S_r ← ∅
 7     foreach {(p,n) | Θ_{k_{p,n}} = 0} do
 8       Θ*_k ← Θ_k
 9       Θ*_{k_{p,n}} ← 1
10       S_r(p,n) ← evaluate(X,Y,Θ*_k)
11     r ← r + 1
12     (p,n) ← arg max S_r(·)
13     if S_r(p,n) ≥ evaluate(X,Y,S_b(r)) then
14       Θ_k ← S_b(r)
15     else
16       Θ_{k_{p,n}} ← 1
17       S_b(r) ← Θ_k
18       backtracking ← True
19       while r > 2 and backtracking=True do
20         S_r ← ∅
21         foreach {(p,n) | Θ_{k_{p,n}} = 1} do
22           Θ*_k ← Θ_k
23           Θ*_{k_{p,n}} ← 0
24           S_r(p,n) ← evaluate(X,Y,Θ*_k)
25         (p,n) ← arg max S_r(·)
26         if S_r(p,n) < evaluate(X,Y,S_b(r−1)) then
27           r ← r − 1
28           Θ_{k_{p,n}} ← 0
29           S_b(r) ← Θ_k
30         else backtracking ← False
31     if S_b(r) = penalty then break
32   return S_b
33
34 function evaluate(X,Y,Θ_k)
35   Compute ω from Equation 4.9 with Θ_k and O_k.
36   if ω_admin ≥ ω then
37     s ← crossValidate(h,X[:,vec(Θ_k)],Y)
38   else s ← penalty
39   return s
```

4.6.1 Testbed

We deployed on a virtualized platform a prototype implementing the experimental testbed (see Fig. 4.4) consists of nine node machines: Applications (Apps) node, Devices (Devs) node, NF1 (SRV) node, NF2 (GW1) node, NF3 (GW11) node, and NF4 (GW111) node.

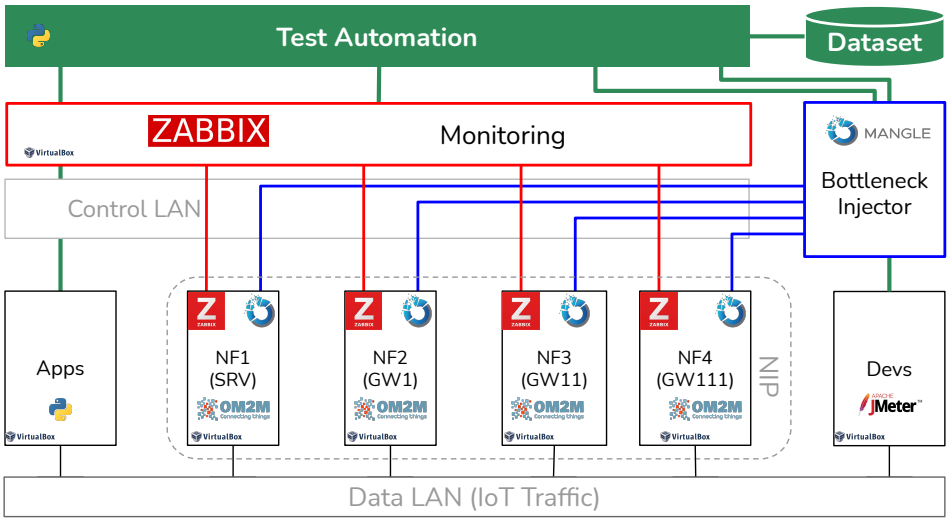


Figure 4.4: Experimental Setup.

Table 4.2 describes the resources allocated to each node.

NF	vCPU	RAM (GB)	Disk (GB)	Match
Applications (Apps)	1	0.5	10	AWS "T2.micro" instance
Devices (Devs)	1	0.5	8	Raspberry Pi 1 Model B computer
NF1 (SRV)	2	2	15	AWS "T2.medium" instance
NF2 (GW1)	1	1	10	AWS "T2.micro" instance
NF3 (GW11)	1	1	10	AWS "T2.micro" instance
NF4 (GW111)	1	0.5	8	Raspberry Pi 1 Model B computer

Table 4.2: Experimental testbed resources description

The testbed is composed of Virtual Machines (VMs) running on Ubuntu server 16.04. A JMeter[6] Server is running in the Devices (Devs) node and produces the IoT workload with a request arrival rate of 20 requests per second. The considered IoT Platform is the Eclipse open-source OM2M[7]. The NIP nodes communicate through the Data LAN. The monitoring data are collected by the Zabbix[8] open-source monitoring software. The bottlenecks injection

[6]Apache JMeter is an Apache project that can be used as a load testing tool for analyzing and measuring various services' performance, focusing on web applications (https://jmeter.apache.org).

[7]The Eclipse OM2M project, initiated by LAAS-CNRS, is an open-source implementation of oneM2M and SmartM2M standard (https://www.eclipse.org/om2m)

[8]Zabbix is an open-source monitoring software tool for diverse IT components, including networks, servers, virtual machines (VMs), and cloud services (https://www.zabbix.com).

and remediation are performed by VMware Mangle[9]. 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.

4.6.2 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 duplication, 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 Algorithm 8 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. 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 Algorithm 8.

Algorithm 8: 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
```

This Algorithm (8) is 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

[9]Mangle enables you to run chaos engineering experiments seamlessly against applications and infrastructure components to assess resiliency and fault tolerance (https://vmware.github.io/mangle).

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.

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

Table 4.3: Injected Bottlenecks during the campaign

Table 4.3 describing 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.

4.6.3 Overview of Multilabel Dataset

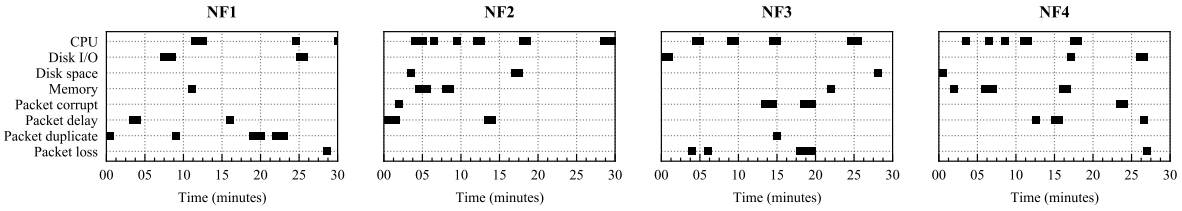


Figure 4.5: Thirty-minute sample of injected bottlenecks per NF (NF1, NF2, NF3, NF4).

As presented in Fig. 4.5, multiple bottlenecks were injected in the considered testbed. The campaign last 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. 4.6.

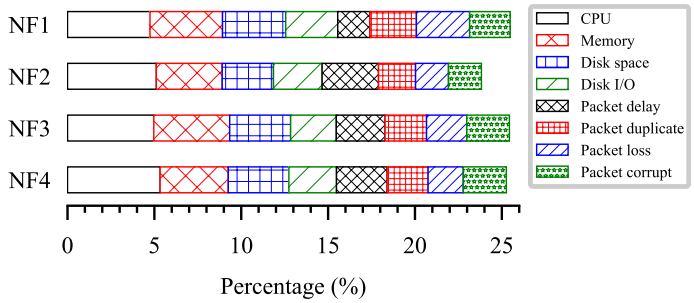


Figure 4.6: Bottlenecks frequency in the dataset per by NF.

4.7 Evaluation

4.7.1 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 standard in one 2×2 table, as depicted in Table 4.4.

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

Table 4.4: Confusion Matrix

Table 4.4 the abbreviations tp, fp, fn, and tn denote the number of respectively true positives, false positives, and false, and true negatives. The terms "True Positive", "False Positive," "True Negative," and "False Negative" refer to the presence or absence of bottlenecks and the correctness of the classification. The same definitions are used throughout the Chapter. For each j-bottleneck the tp_j, fp_j, fn_j, and tn_j are defined as follows.

$$\text{tp}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 1 \text{ and } Y_j^i = 1) \tag{4.10}$$

$$\text{fp}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 1 \text{ and } Y_j^i = 0) \tag{4.11}$$

$$\text{fn}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 0 \text{ and } Y_j^i = 1) \tag{4.12}$$

$$\text{tn}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 0 \text{ and } Y_j^i = 0) \tag{4.13}$$

As stated in the motivation Section, in this work, we are interested in a MBI 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 the NIP has the identified bottleneck in the case of a positive test. 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.

$$\Psi_{\text{Precision}} = \frac{1}{B} \sum_{j=1}^{B} \frac{tp_j}{tp_j + fp_j} \tag{4.14}$$

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 MBI model has to predict on average without missing any true one.

$$\Psi_{\text{Subset accuracy}} = \frac{1}{m} \sum_{i=1}^{m} 1(\hat{Y}^i = Y^i) \tag{4.15}$$

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).

$$\Psi_{\text{Coverage Error}} = \frac{1}{m} \sum_{i=1}^{m} \max_{j \in Y_i} r_i(j) \tag{4.16}$$

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.

$$\Psi_{\text{Specificity}} = \frac{1}{B} \sum_{j=1}^{B} \frac{tn_j}{tn_j + fp_j} \tag{4.17}$$

$$\Psi_{\text{Sensitivity}} = \frac{1}{B}\sum_{j=1}^{B}\frac{tp_j}{tp_j + fn_j} \tag{4.18}$$

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 helpful 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.

$$\Psi_{\text{AUC}} = \int_{x=0}^{1} \Psi_{\text{Specificity}}((1 - \Psi_{\text{Sensitivity}})^{-1}(x))\,\mathrm{d}x \tag{4.19}$$

Below, we present the MBI and the SOMS evaluations.

4.7.2 Multiple bottlenecks identification (MBI)

There are two main approaches [Zhang 2013] 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 latter's objective 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. There are algorithms based on nearest neighbors in the algorithm adaptation approach, such as ML-kNN. 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 MBI model based on the Ψ_{AUC}. As in the literature, we use 75% of the collected data for training the different MBI 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 [Szymański 2017]. In Fig. 4.7 (a) - (d) four curves are shown. The diagonal line (Random Classifier) shows the performance of a random

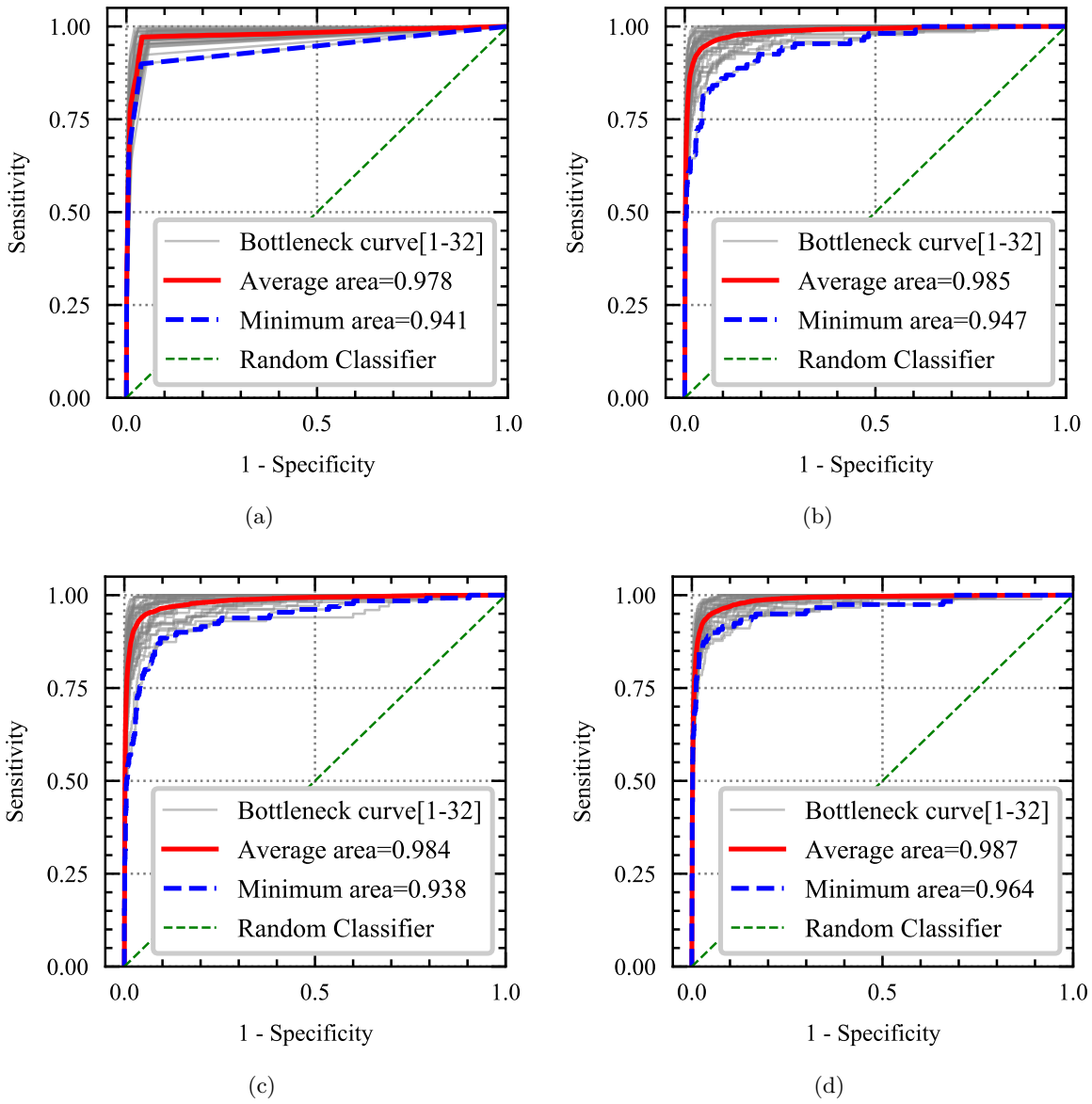


Figure 4.7: MLC Receiver operating characteristic and AUC (Ψ_{AUC}). (a) ML-kNN; (b) Binary Relevance; (c) Classifier Chain; (d) Label Powerset.

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 of AUC. The third Curve (Average Area) corresponds to the average 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. 4.7 (d). Label Powerset is a problem

transformation approach that transforms a multilabel problem into a multiclass problem with one multiclass classifier trained on all unique label combinations found in the training data.

	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**

Table 4.5: Multi-label Classifiers Performance Comparison (with Hyper-parameter optimization)

Additionally, as presented in table 4.5, 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 Chapter. The reader may see in [Tsoumakas 2009] for further details about the Label Powerset algorithm.

In Fig. 4.8 we present a deeper look into the Label Powerset Algorithm performance. Fig. 4.8 (a) shows the bottlenecks identification *precision* grouped by NF and Fig. 4.8 (b) shows the NF identification *precision* grouped by bottlenecks type. In Fig. 4.8 (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. 4.8 (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 SOMS Algorithm in the Adaptive Performance Analysis use case described in Section 4.2.

4.7.3 Simple Overhead-sensitive Metrics Selection (SOMS)

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 MBI model's ability to minimize the false positives (i.e., the precision). Let assume, for evaluation purpose, that the monitoring overhead increases by 0.5 moving from the Cloud to Things – the monitoring overhead is set to

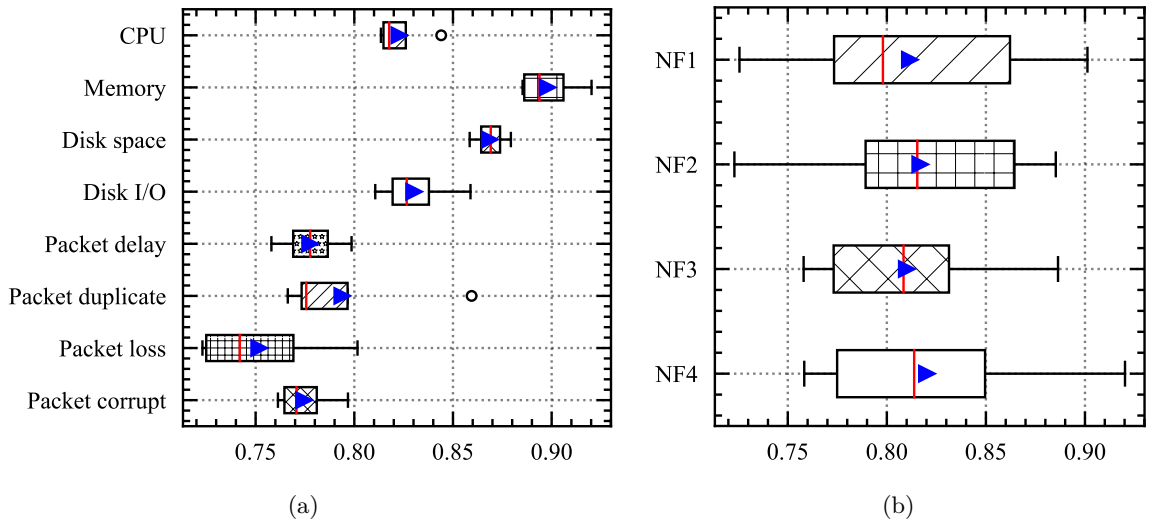


Figure 4.8: Label Powerset Model precision ($\Psi_{\text{Precision}}$). (a) Bottlenecks identification precision grouped by NF; (b) NF identification precision grouped by Bottlenecks.

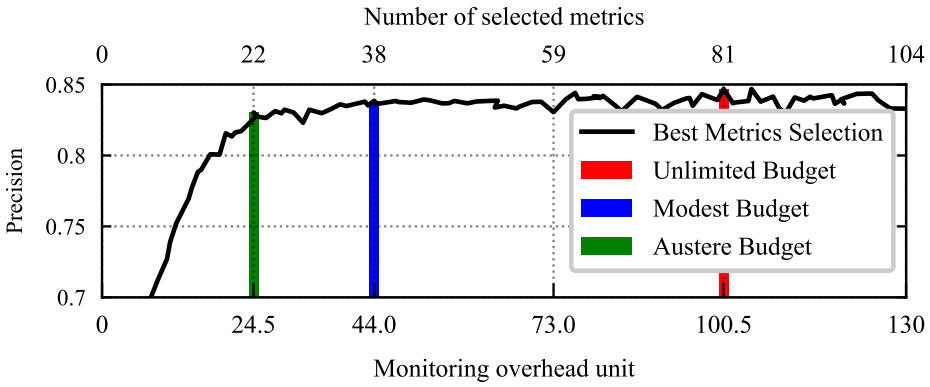


Figure 4.9: SOMS Algorithm precision.

0.5 on NF1, 1 on NF2, 1.5 on NF3, 2 on NF4. As stated in Section 4.2, 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. We use 75% of the collected data for Algorithm training and 25% for the evaluation.

In Fig. 4.9, SOMS Algorithm removes or adds metrics at the time based on the MBI 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 shown on the first x-axis and the second x-axis. When all the metrics are selected, the *precision* of the MBI 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. 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. The best subset metric compatible with this budget is 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 MBI model. This is explained by the fact that some (irrelevant) metrics act as noise on the model and degrading its performance.

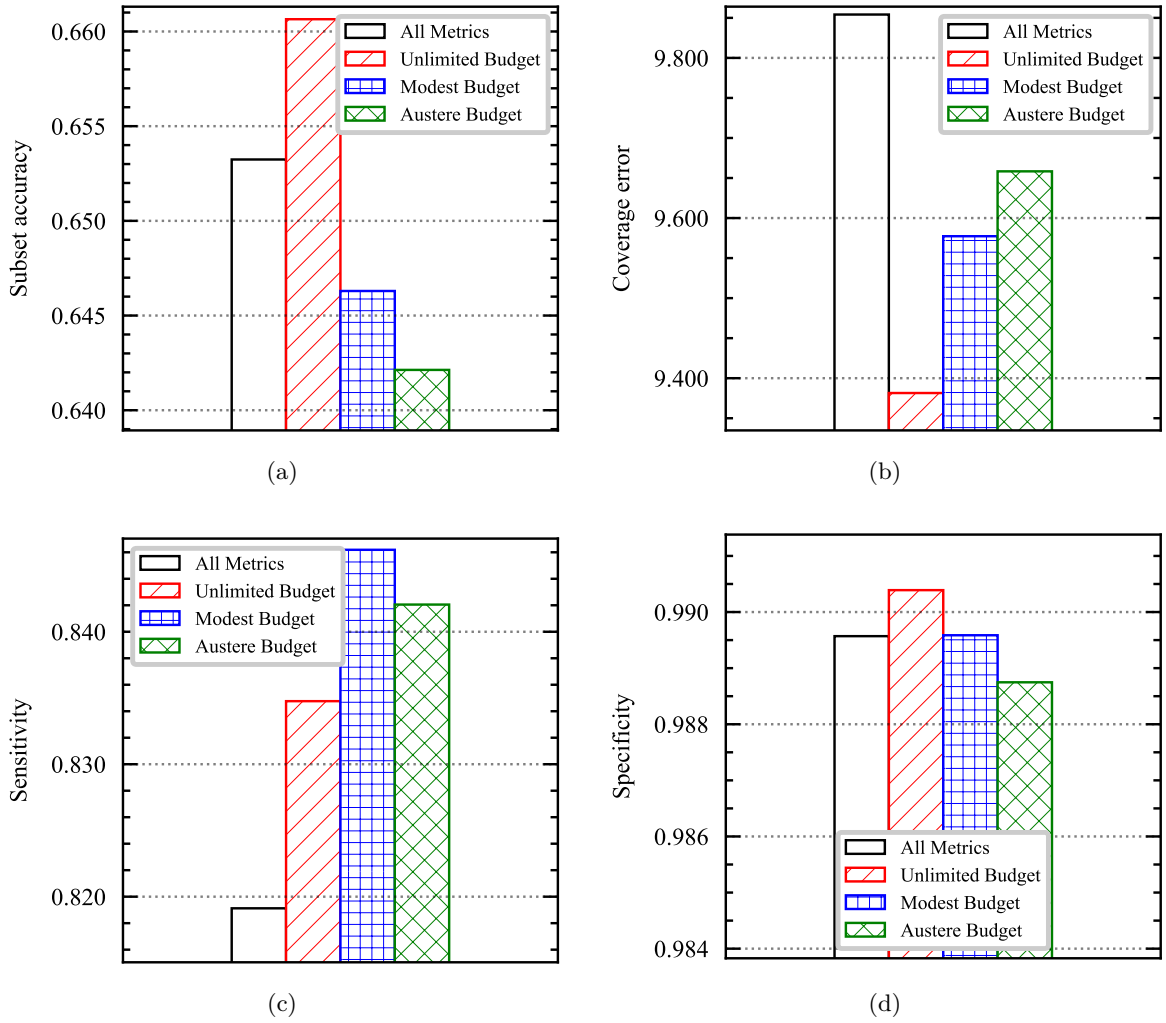


Figure 4.10: Performance in different scenarios. (a) Subset accuracy; (b) Coverage error; (c) Sensitivity; (d) Specificity.

We present an in-depth look at the performance associated with different scenarios. As

Fig. 4.10 shows, in addition to the MBI 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. 4.10 (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 MBI model *Subset accuracy*. In the Modest Budget and the Austere Budget scenarios, the *Subset accuracy* is 0.64. In Fig. 4.10 (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. 4.10 (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. 4.10 (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.

4.7.4 Discussion

As earlier stated, our goal in this Chapter is to build an Adaptive Performance Analysis method that optimizes the 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 MBI and the SOMS. The MBI model is used for the multiple bottlenecks analysis, and the SOMS model is used for the metric selection optimization. Regarding the MBI 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 MBI model.

To achieve the metric selection regarding a monitoring overhead budget, we have proposed SOMS (a feature selection heuristic). SOMS optimize the MBI 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. 4.10). 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. Nevertheless, our

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 (MBI 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 [Tsoumakas 2010]. The SOMS Algorithm computational complexity is upper bounded by $\mathcal{O}(2^{P \times N_k})$ [Doak 1992].

Our numerical results show that 81 metrics give the maximum precision (84%) of the MBI model. Up to 83% can be achieved even with a relatively limited metrics subset of 22 metrics. Regarding these experimental results, it is possible to conclude that our approach gives valuable information to make decisions about the NIP bottlenecks to improve the QoS.

4.8 Integration in the Autonomic Manager

As presented in Chapter 2, the following components interact with the Autonomic Manager The *Monitoring* component [Kephart 2003] that collects the details from the managed NIP via monitoring agents (Sensors). The details include data such as topology information, QoS, and performance metrics. The Autonomic Manager retrieves and stores these collected data for analyzing purposes. The *planner* component [Kephart 2003] provides the mechanism to schedule and perform the necessary changes to the NIP. Once the *planner* has generated an adaptation plan, some actions may need to be taken to modify the state of one or more NIP nodes. The following components interact within the Autonomic Manager. The *Knowledge base* component stores the data. The knowledge base includes topology information, historical logs, metrics, IoT applications information, and the allocated overhead budget. The *Monitoring* component uses simple models, such as time-series forecasting, to detect the violations on IoT applications' QoS. The *Monitoring* component is continuously invoked. The output of *Monitoring* component is performance data associated with a QoS violation. The *Adaptive Performance Analysis (Analyzer)* components analyze the non-trivial dependency in the provided data to analyze the bottlenecks causing a detected violation. This component is invoked by the *Monitoring* component when it detects a QoS violation (see Section 4.5). The *Planner* component (see [Ouedraogo 2020]) determines the set of candidate actions to recover from identified bottlenecks. The *planner* is in charge of increasing (or decreasing) the number of metrics to be observed in the NIP. Let us remark that such functionality is not implemented by the current version of the QoS4NIP planner. This component is invoked by *Analyzer* component every time the selected metrics subset is updated.

4.9 Conclusion

To summarize, we have proposed in this Chapter 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. 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 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 [oneM2M 2016]. 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 valuable insights into how one can adaptively analyze performance bottlenecks in NIPs (i.e., determine the proper metric subset to collect) while efficiently controlling the induced monitoring overhead. The following Chapter concludes the thesis and summarizes the major contributions while highlighting future research directions and perspectives.

CHAPTER 5

Conclusion and Perspectives

5.1 Conclusion

5.1.1 Summary

In recent years, the IoT has evolved at an exceptional speed, making it possible to connect a large number of heterogeneous things (such as sensors, actuators, smartphones, applications). One of the important aspects of this IoT is the IoT platform (a.k.a middleware), the objective of which is to connect remote devices to user applications and manages all the interactions between the hardware and the applications. Today, there are many proprietary solutions on the market, which remain very specific to their manufacturer and application area. This makes the applications very dependent on hardware and software (e.g., sensors of a particular brand, specific development environments), and therefore difficult to deploy and maintain. This induces a "vertical" fragmentation of the IoT solutions offered. Fortunately, many initiatives have led to the specification and implementation of several "horizontal" platform solutions. If heterogeneity seems to be resolved at the platform level, that of QoS remains an open problem until today. Besides, such a platform is so complex that a high degree of autonomy is needed to overcome several challenges.

Considering the current limitations (discussed in Chapter 2) on the QoS management in IoT platforms, we addressed in this thesis the lack of an approach that can, autonomously, handle the scale and resource scarcity of today's IoT platforms and sustain QoS to IoT Applications.

We investigated a general approach that consists in designing, developing, and experimenting with behavioral models for autonomous management of QoS in the IoT platform: i) taking advantage of the technological opportunities offered in the Cloud-enabled infrastructures (i.e., the dynamic deployment of network functions, programmable networks), ii) taking advantage

of the technological opportunities offered by the dynamic deployment of software components, iii) and following autonomous computing concepts.

5.1.2 Thesis Contributions

The summary of the thesis contributions is presented below.

- Beyond and in addition to the classic concept of VNF, in our first contribution, we proposed the concept of ANF, which is based on a software isolation technique that consumes fewer resources. ANFs allow the deployment of network functions in resource-constrained environments, typically on end gateways of IoT platforms. They also lead to optimal use of available resources. On this basis and to maintain at the best level the QoS required by IoT applications, we have designed a set of IoT TCF implemented as VNF and ANF.

- To achieve an optimal deployment of these TCFs, our second contribution consisted in the formulation of a multi-objective optimization problem. The proposed and the implemented solution takes into account both the deployment of TCFs and scaling actions, intending to optimize the QoS of IoT applications. We investigate GA to solve this problem. The proposed algorithm relies on the bottlenecks (such as CPU, RAM) of the platform nodes, first provided manually by a human administrator.

- In a third contribution, we then turn to the automated identification of these bottlenecks. To do this, we proposed an adaptive identification approach that considers the cost associated with the monitoring of the IoT platform. Indeed, it is not desirable that the overhead generated by the monitoring system itself causes QoS problems in the IoT platform. We modeled the problem of identifying multiple bottlenecks by a multi-label classification problem. Different supervised learning algorithms have been studied to solve this problem. Finally, we proposed an algorithm for selecting metrics to monitor in IoT platforms according to the costs they generate.

5.2 Perspectives

During the thesis, we have faced various challenges. Future research directions can be summarized as follows.

5.2.1 Short-term research directions

In the short term, we are considering the following avenues of research.

- **Implementation of a proof of concept prototype:** All the proposed algorithms in this thesis were proven and evaluated through either theoretical analysis, partial prototyping, or extensive simulations. Although the reaction time of the different components for the MAPE-K loop, taken into account in the design stages and guided the selection of families or types of algorithms in all of our contributions. However, these means are insufficient to prove the real performances. Therefore, we plan to implement the algorithms into a real MAPE-K loop.

- **ANF large-scale experimentation testbed:** The limited number of ANF-hosts prevent us from a large-scale measurement campaign of the proposal's experiments. A real-world deployment in a broader scale environment would need to deploy a large number of ANF-hosts. Today, such resources are not yet available, unlike the NFV-I that can be deployed at a significant scale by provisioning a high number of VM (e.g., Amazon EC2 VMs). A potential future work to solve this issue is deploying an open crowd-sourced testbed for large-scale experimentation.

5.2.2 Medium term research directions

In the medium term, we are considering the following avenues of research.

- **Traffic Control Functions parameter configuration:** The current QoS4NIP planner in Chapter 4 considers only the optimization of NFs (VNF/ANF) chaining to be deployed and scaling actions. It does not go further into finding the optimal parameter configuration for all these actions (scaling the NF with different sizes, adapting the loss rate within the Shaper, adapting the timeout limit, the queue reservation rate, and the other parameters for the other functions). It would also be interesting to extend the current planner to configure the TCFs parameters optimally.

- **Metrics Selection Problem Formulation:** A 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 [Morales 2019]), and take advantages of the benefits of each of these distinct algorithms while mitigating their weaknesses to identify known bottleneck as well as an unknown bottleneck. Considering the injected bottleneck types investigated in our experiments, it was 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.

5.2.3 Long-term research directions

In the long term, we are considering the following avenues of research.

- **Distributed QoS management:** Huge chunk of current research focus is on centralized control loops. Modern IoT platforms are inherently distributed with components spanning multiple physical domains (servers or datacenters). Data collection across such domains is often impractical or difficult due to potential system overheads, proprietary, and privacy regulations. This implication calls for a decentralized approach that fits naturally with such platforms.
- **Multi-level QoS management:** Current efforts must extend towards the QoS management in IoT platforms at different OSI levels considering the complexity of today's infrastructure and application. For example, it should be possible to identify bottlenecks from a set of higher and lower level application service components through the virtualization layer to system resource bottlenecks. Similarly, reconfiguration planning should focus on the application layer and be extended to the transport and network layers. Promising future research would be to implement the proposed solution in this thesis to handle the full OSI stack.

APPENDIX A

Appendix

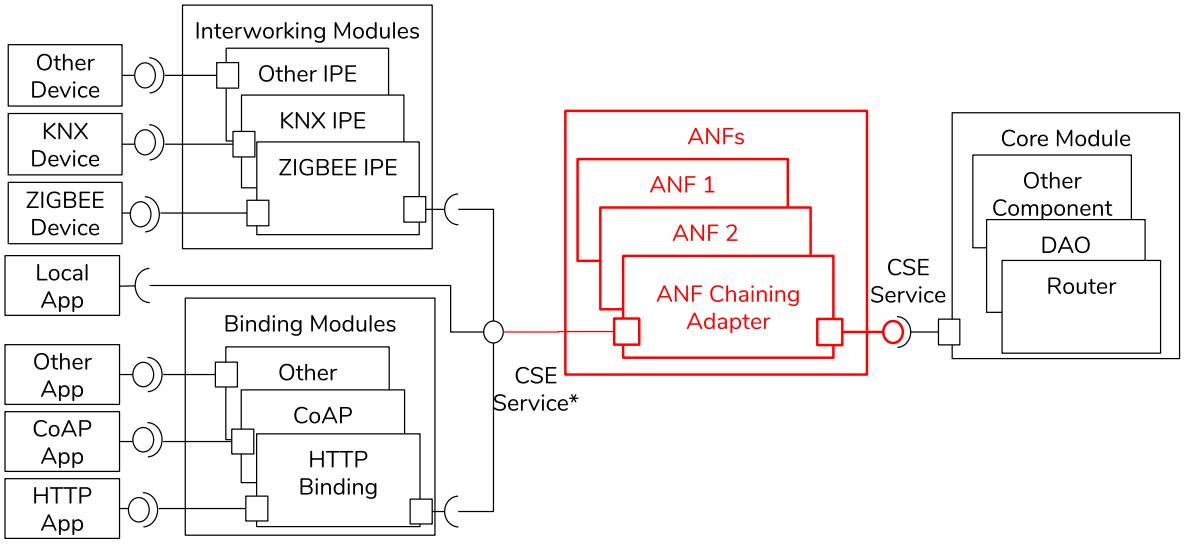
VNFs Implementation Details in Docker

We develop a prototype of the traffic functions in Java 8. Two service layer protocols are supported in this implementation: Constrained Application Protocol (CoAP) and Hypertext Transfer Protocol (HTTP). Moreover, we based the service layer protocols implementation on public optimized Open-Source libraries: Californium (https://www.eclipse.org/californium) for CoAP and Apache HTTP (https://httpd.apache.org) for HTTP. After the compilation of source code, the binaries of the TCFs are built into Docker images (Ubuntu 16.04). The associated VNF packages are created and onboarded in the ETSI-MANO OpenBaton and ready to be deployed as VNFs. IP traffic redirection, when necessary, is done using Software-defined networking (SDN) by adding Openflow rules on the NFV-I interconnection switches via the NFV-I SDN controller REST API.

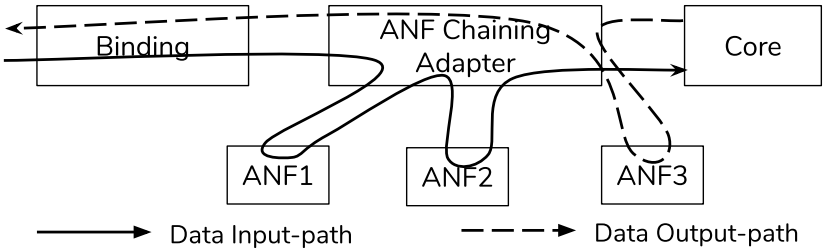
ANFs Implementation Details in Eclipse OM2M

OM2M nodes are developed following a modular architectural style based on the OSGi standard [Alliance 2018]. Thanks to this implementation, it is possible to integrate our ANFs as OSGi Bundles. Our integration approach is achieved so that the OM2M node maintains its modular design and operates without these new ANFs. An OM2M node (in-cse or mn-cse) is composed of the following components: *Core*, *Binding*, *Persistence*, and *Interworking Proxy Entity (IPE)*. The *Core* component is responsible for processing generic requests and responses (i.e., protocol-agnostic messages). It implements features such as Registration, Discovery, Re-routing, Notifications. The *Binding* components act as translators of protocol-specific messages to generic messages and vice versa. A *Binding* component is necessary for every supported protocol (i.e., HTTP, CoAP). The *Persistence* components are responsible for implementing the data storage strategy. There is an interface component and supported storage locations (in-memory, file, or server databases). Similar to the *Binding* components, they provide a translation of generic messages into non-IP (i.e., Bluetooth, ZigBee, Z-Wave) messages and vice versa.

To achieve this integration, we had to consider two options: (1) to re-implement the *Binding* components and *Interworking Proxy Entity* components of a node to add a new interface to be used for the communication with ANFs. Such a modification would have resulted in a new version of Eclipse OM2M, or (2) to use the OSGi feature "Proxying Service" [Alliance 2018],



(a) Internal structure of an OM2M node integrating ANFs.



(b) ANFs chaining in an OM2M node.

Figure A.1: Seamless integration in the OM2M IoT platform.

which allows us to intermediate an OSGi service. We have chosen the second option, which enables the integration of ANFs without affecting the oneM2M [oneM2M 2016] standard being implemented through Eclipse OM2M. Furthermore, this option has the advantage of not changing any element of the current implementation of OM2M. As shown in Fig. A.1a, the main element of this architecture is the "ANF Chaining adapter." This component is specified following a design pattern [Gamma 1995]. It intermediates the OSGi service between the *Core* component and the *Binding* components. Depending on its configuration, it also decides to pass the request message through zero or several ANFs before reaching the *Core*. The same applies to the response message. We implemented a Management Agent (MA) that receives and installs ANF files (JAR). We also implemented an ANF deployment manager that deploys ANFs on a remote node. The deployment manager also configures ANFs dynamically, including the "ANFs Chaining adapter," a particular ANF. An example is illustrated in A.1b. After implementing OSGi compatible source code, we generate the JAR (Java ARchive) associated with each TCFs. The generated JARs are ready to be deployed as ANFs. More details of the architecture of

implementation, integration, and deployment of the TCFs into the Eclipse OM2M can be found in [Ouedraogo 2018b].

Monitored metrics

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

Sources of implemented works

1. **ANF and VNF performance measurement** The performance measurement results to get the quantitative characteristics associated with the different TCFs implementation packages are available at github.com/couedrao/QoS4NIP.

2. **QoS4NIP Algorithm** The Python source of the proposed planning scheme algorithm is available for download at github.com/couedrao/QoS4NIP.

3. **Multi-bottlenecks dataset** The experiment dataset are available at github.com/couedrao/APA4NIP.

4. **APA4NIP Algorithms** The Python source of the proposed analyze algorithms is available at github.com/couedrao/APA4NIP.

This page intentionally left blank.

Journals

1. **Ouedraogo, Clovis Anicet**, Samir Medjiah, Christophe Chassot, Khalil Drira, and Jose Aguilar. "Adaptive Performance Analysis in IoT Platforms." IEEE Transactions on Network and Service Management **(Under review)**.

2. **Ouedraogo, Clovis Anicet**, Samir Medjiah, Christophe Chassot, Khalil Drira, and Jose Aguilar. "A Cost-Effective Approach for End-to-End QoS Management in NFV-enabled IoT Platforms." IEEE internet of things journal (2020).

3. Morales, Luis, **Ouedraogo, Clovis Anicet**, José Aguilar, Christophe Chassot, Samir Medjiah, and Khalil 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 13, no. 3 (2019): 199-219.

Conferences

4. **Ouedraogo, Clovis Anicet**, Samir Medjiah, Christophe Chassot, and Jose Aguilar. "Flyweight Network Functions for Network Slicing in IoT." In 2018 International Conference on Smart Communications in Network Technologies (SaCoNeT), pp. 31-36. IEEE, 2018.

5. **Ouedraogo, Clovis Anicet**, Samir Medjiah, Christophe Chassot, and Khalil Drira. "Enhancing middleware-based IoT applications through run-time pluggable Qos management mechanisms. application to a oneM2M compliant IoT middleware." Procedia computer science 130 (2018): 619-627.

6. **Ouedraogo, Clovis Anicet**, Samir Medjiah, and Christophe Chassot. "A modular framework for dynamic qos management at the middleware level of the iot: Application to a onem2m compliant iot platform." In 2018 IEEE International Conference on Communications (ICC), pp. 1-7. IEEE, 2018.

7. Banouar, Yassine, **Ouedraogo, Clovis Anicet**, Christophe Chassot, and Abdellah Zyane. "QoS management mechanisms for enhanced living environments in IoT." In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1155-1161. IEEE, 2017.

Demos

8. Raissi, Fatma, **Ouedraogo, Clovis Anicet**, Sami Yangui, Frederic Camps, and Nejib Bel Hadj-Alouane. "Paving the way for autonomous cars in the city of tomorrow: A prototype for mobile devices support at the edges of 5g network." In International Conference on Service-Oriented Computing, pp. 481-485. Springer, Cham, 2018.

9. **Ouedraogo, Clovis Anicet**, El-Fadel Bonfoh, Samir Medjiah, Christophe Chassot, and Sami Yangui. "A prototype for dynamic provisioning of qos-oriented virtualized network functions in the internet of things." In 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), pp. 323-325. IEEE, 2018.

Bibliography

[Abdullah 2013] Saima Abdullah et Kun Yang. *A QoS aware message scheduling algorithm in Internet of Things environment.* 2013 IEEE Online Conf. Green Commun. OnlineGreenComm 2013, pages 175–180, 2013. (Cited in pages 37, 38 and 40.)

[Abramson 2006] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu et John Wiegert. *Intel Virtualization Technology for Directed I/O.* Intel technology journal, vol. 10, no. 3, 2006. (Cited in page 21.)

[Agirre 2016] Aitor Agirre, Jorge Parra, Aintzane Armentia, Elisabet Estévez et Marga Marcos. *QoS Aware Middleware Support for Dynamically Reconfigurable Component Based IoT Applications.* Int. J. Distrib. Sens. Networks, vol. 12, no. 4, page 2702789, apr 2016. (Cited in pages 37, 38 and 40.)

[Alaya 2014] M Ben Alaya, Yassine Banouar, Thierry Monteil, Christophe Chassot et Khalil Drira. *OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability.* Procedia Computer Science, vol. 32, pages 1079–1086, 2014. (Cited in pages xi, 16 and 59.)

[Alliance 2014] Open Mobile Alliance. *RESTful Network API for Quality of Service Requirements.* Rapport technique, Alliance, Open Mobile, 2014. (Cited in page 37.)

[Alliance 2018] OSGi Alliance. *Osgi service platform, enterprise specification, release 7, version 1.0.* OSGi Specification, 2018. (Cited in pages 3, 51, 58 and 113.)

[Ashton 2009] Kevin Ashton*et al. That 'internet of things' thing.* RFID journal, vol. 22, no. 7, pages 97–114, 2009. (Cited in page 10.)

[Banouar 2017] Yassine Banouar. *Gestion autonomique de la QoS au niveau middleware dans l'IoT.* PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2017. (Cited in pages 2 and 58.)

[Battré 2010] Dominic Battré, Matthias Hovestadt, Björn Lohrmann, Alexander Stanik et Daniel Warneke. *Detecting bottlenecks in parallel dag-based data flow programs.* In 2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers, pages 1–10. IEEE, 2010. (Cited in page 90.)

[Bhowmik 2017] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler et Kurt Rothermel. *High performance publish/subscribe middleware in software-defined networks.* IEEE/ACM Trans. Netw., vol. 25, no. 3, pages 1501–1516, 2017. (Cited in pages 37, 38, 39 and 40.)

[Boban 2018] Mate Boban, Apostolos Kousaridas, Konstantinos Manolakis, Josef Eichinger et Wen Xu. *Connected Roads of the Future: Use Cases, Requirements, and Design Considerations for Vehicle-to-Everything Communications.* IEEE Veh. Technol. Mag., vol. 13, no. 3, pages 110–123, 2018. (Cited in page 71.)

[Bouattour 2020] Hedi Bouattour, Yosra Ben Slimen, Marouane Mechteri et Hanane Biallach. *Root Cause Analysis of Noisy Neighbors in a Virtualized Infrastructure.* IEEE Wirel. Commun. Netw. Conf. WCNC, vol. 2020-May, pages 10–15, 2020. (Cited in page 86.)

[Boyd 1987] John R Boyd. *Organic design for command and control.* A discourse on winning and losing, 1987. (Cited in page 23.)

[Brogi 2017] Antonio Brogi et Stefano Forti. *QoS-aware deployment of IoT applications through the fog.* IEEE Internet of Things Journal, vol. 4, no. 5, pages 1185–1192, 2017. (Cited in pages 44 and 83.)

[Brooks Jr 1995] Frederick P Brooks Jr. The mythical man-month: essays on software engineering. Pearson Education, 1995. (Cited in page 21.)

[Carella 2015] Giuseppe Antonio Carella et Thomas Magedanz. *Open baton: a framework for virtual network function management and orchestration for emerging software-based 5g networks.* Newsletter, vol. 2016, 2015. (Cited in page 58.)

[Carpenter 2002a] B. E. Carpenter et K. Nichols. *Differentiated services in the Internet.* Proceedings of the IEEE, vol. 90, no. 9, pages 1479–1494, 2002. (Cited in page 33.)

[Carpenter 2002b] Brian E Carpenter et Kathleen Nichols. *Differentiated services in the Internet.* Proceedings of the IEEE, vol. 90, no. 9, pages 1479–1494, 2002. (Cited in pages 49 and 53.)

[Casado 2007] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown et Scott Shenker. *Ethane: Taking control of the enterprise.* ACM SIGCOMM computer communication review, vol. 37, no. 4, pages 1–12, 2007. (Cited in page 19.)

[Chen 2015] Z Chen et C Wang. *Use Cases and Requirements for using Track in 6TiSCH Networks.* In draft. IETF, 2015. (Cited in page 36.)

[Cheng 2018] Xiangle Cheng, Yulei Wu, Geyong Min et Albert Y. Zomaya. *Network Function Virtualization in Dynamic Networks: A Stochastic Perspective.* IEEE Journal on Selected Areas in Communications, pages 1–1, 2018. (Cited in page 48.)

[Chevrollier 2005] Nicolas Chevrollier et Nada Golmie. *On the use of wireless network technologies in healthcare environments.* In Proceedings of the fifth IEEE workshop on

Applications and Services in Wireless Networks (ASWN 2005), pages 147–152, 2005. (Cited in page 36.)

[Choudhury 2016] Balamati Choudhury et Rakesh Mohan Jha. Soft computing techniques, page 9–44. Cambridge University Press, 2016. (Cited in pages 4 and 26.)

[Cisco 2020] U Cisco. *Cisco annual internet report (2018–2023) white paper*, 2020. (Cited in page 22.)

[Cotroneo 2017a] Domenico Cotroneo, Luigi De Simone et Roberto Natella. *NFV-bench: A dependability benchmark for network function virtualization systems.* IEEE Trans. Netw. Serv. Manag., vol. 14, no. 4, pages 934–948, 2017. (Cited in page 85.)

[Cotroneo 2017b] Domenico Cotroneo, Roberto Natella et Stefano Rosiello. *A Fault Correlation Approach to Detect Performance Anomalies in Virtual Network Function Chains.* Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE, vol. 2017-October, pages 90–100, 2017. (Cited in page 85.)

[Cotroneo 2018] Domenico Cotroneo, Luigi De Simone et Roberto Natella. *Dependability Certification Guidelines for NFVIs through Fault Injection.* Proc. - 29th IEEE Int. Symp. Softw. Reliab. Eng. Work. ISSREW 2018, pages 321–328, 2018. (Cited in page 85.)

[Cui 2017] Yiqian Cui, Junyou Shi et Zili 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, pages 1965–1978, aug 2017. (Cited in page 85.)

[Cziva 2017a] R. Cziva et D. P. Pezaros. *Container Network Functions: Bringing NFV to the Network Edge.* IEEE Communications Magazine, vol. 55, no. 6, pages 24–31, 06 2017. (Cited in page 46.)

[Cziva 2017b] Richard Cziva et Dimitrios P Pezaros. *Container network functions: bringing NFV to the network edge.* IEEE Communications Magazine, vol. 55, no. 6, pages 24–31, 2017. (Cited in page 20.)

[Deb 2001] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms, volume 16. John Wiley & Sons, 2001. (Cited in page 52.)

[Deb 2002] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal et TAMT Meyarivan. *A fast and elitist multiobjective genetic algorithm: NSGA-II.* IEEE transactions on evolutionary computation, vol. 6, no. 2, pages 182–197, 2002. (Cited in pages 52, 68, 69 and 71.)

[Deb 2014] K. Deb et H. Jain. *An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With*

Box Constraints. IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pages 577–601, Aug 2014. (Cited in page 68.)

[Decasper 2000] Dan Decasper, Zubin Dittia, Guru Parulkar et Bernhard Plattner. *Router plugins: a software architecture for next-generation routers*. IEEE/ACM transactions on Networking, vol. 8, no. 1, pages 2–15, 2000. (Cited in page 49.)

[Doak 1992] J. Doak. *An evaluation of feature selection methods and their application to computer security*. Davis: University of California, Department of Computer Science, 1992. (Cited in page 106.)

[Draxler 2018] Sevil Draxler, Holger Karl et Zoltan Adam Mann. *JASPER: Joint Optimization of Scaling, Placement, and Routing of Virtual Network Services*. IEEE Transactions on Network and Service Management, vol. 15, no. 3, pages 946–960, sep 2018. (Cited in pages 47 and 48.)

[Ergun 2021] Kazim Ergun, Raid Ayoub, Pietro Mercati, Dancheng Liu et Tajana Rosing. *Energy and QoS-Aware Dynamic Reliability Management of IoT Edge Computing Systems*. In Proceedings of the 26th Asia and South Pacific Design Automation Conference, pages 561–567, 2021. (Cited in page 22.)

[ETSI 2014] ETSI. *Network Functions Virtualisation (NFV); Architectural framework. 2014*. Group Specification, 2014. (Cited in pages xi, 14 and 49.)

[ETSI 2019] ETSI. *ETSI GS ENI 005 Experiential Networked Intelligence (ENI); System Architecture*. ETSI GS ENI, 2019. (Cited in page 24.)

[Ezdiani 2015] Syarifah Ezdiani, Indrajit S. Acharyya, Sivaramakrishnan Sivakumar et Adnan Al-Anbuky. *An IoT Environment for WSN Adaptive QoS*. Proc. - 2015 IEEE Int. Conf. Data Sci. Data Intensive Syst. 8th IEEE Int. Conf. Cyber, Phys. Soc. Comput. 11th IEEE Int. Conf. Green Comput. Commun. 8th IEEE Inte, no. December, pages 586–593, 2015. (Cited in pages 37, 38 and 40.)

[Fraser 1957] Alex S Fraser. *Simulation of genetic systems by automatic digital computers I. Introduction*. Australian Journal of Biological Sciences, vol. 10, no. 4, pages 484–491, 1957. (Cited in page 29.)

[Gallard 2008] Jérôme Gallard, Adrien Lebre, Geoffroy Vallée, Christine Morin, Pascal Gallard et Stephen Scott. *Refinement Proposal of the Goldberg's Theory*. PhD thesis, INRIA, 2008. (Cited in pages xi and 18.)

[Gallo 2018] M. Gallo, S. Ghamri-Doudane et F. Pianese. *CliMBOS: A Modular NFV Cloud Backend for the Internet of Things*. In New Technologies, Mobility and Security (NTMS),

2018 9th IFIP International Conference on, pages 1–5. IEEE, 2018. (Cited in pages 46, 47 and 50.)

[Gama 2008] Óscar Gama, Paulo Carvalho, José A Afonso et PM Mendes. *Quality of service support in wireless sensor networks for emergency healthcare services.* In 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pages 1296–1299. IEEE, 2008. (Cited in page 36.)

[Gamma 1995] Erich Gamma. Design patterns: elements of reusable object-oriented software. Pearson Education India, 1995. (Cited in page 114.)

[Gonzalez 2017] Jose Manuel Navarro Gonzalez, Javier Andion Jimenez, Juan Carlos Duenas Lopez et Hugo A.G. Parada. *Root Cause Analysis of Network Failures Using Machine Learning and Summarization Techniques.* IEEE Commun. Mag., vol. 55, no. 9, pages 126–131, 2017. (Cited in page 85.)

[Gozdecki 2003] Janusz Gozdecki, Andrzej Jajszczyk et Rafal Stankiewicz. *Quality of service terminology in IP networks.* IEEE communications magazine, vol. 41, no. 3, pages 153–159, 2003. (Cited in page 33.)

[Gregg 2013] Brendan Gregg. Systems performance: enterprise and the cloud. Pearson Education, 2013. (Cited in page 81.)

[GSNFV 2013] ETSI GSNFV. *Network functions virtualisation (NFV); use cases.* V1, vol. 1, pages 2013–10, 2013. (Cited in page 20.)

[Guevara 2017] Judy C. Guevara, Luiz F. Bittencourt et Nelson L.S. Da Fonseca. *Class of service in fog computing.* 2017 IEEE 9th Latin-American Conf. Commun. LATINCOM 2017, vol. 2017-Janua, pages 1–6, 2017. (Cited in pages 37, 38, 39 and 40.)

[Gupta 2013] Puja Gupta et Neha Kulkarni. *An introduction of soft computing approach over hard computing.* International Journal of Latest Trends in Engineering and Technology (IJLTET), vol. 3, pages 254–258, 2013. (Cited in pages xi and 27.)

[Hadka 2017] D Hadka. *Platypus: A free and open source python library for multiobjective optimization.* Available on Github, vol. https://github. com/Project-Platypus/Platypus, 2017. (Cited in page 72.)

[Herrera 2016] Juliver Gil Herrera et Juan Felipe Botero. *Resource allocation in NFV: A comprehensive survey.* IEEE Transactions on Network and Service Management, vol. 13, no. 3, pages 518–532, 2016. (Cited in page 47.)

[Holland 1992] John Henry Holland*et al.* Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992. (Cited in page 29.)

[Horn 2001] Paul Horn. *Autonomic computing: IBM's perspective on the state of information technology.* IBM, 2001. (Cited in page 23.)

[Huning 1976] Alois Huning. *Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution*, 1976. (Cited in page 30.)

[Hwang 2015a] Jinho Hwang, K K_ Ramakrishnan et Timothy Wood. *NetVM: High performance and flexible networking using virtualization on commodity platforms.* IEEE Transactions on Network and Service Management, vol. 12, no. 1, pages 34–47, 2015. (Cited in page 20.)

[Hwang 2015b] Kai Hwang, Xiaoying Bai, Yue Shi, Muyang Li, Wen-Guang Chen et Yongwei Wu. *Cloud performance modeling with benchmark evaluation of elastic scaling strategies.* IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 1, pages 130–143, 2015. (Cited in pages 72 and 73.)

[Ishibuchi 2016] H. Ishibuchi, R. Imada, Y. Setoguchi et Y. Nojima. *Performance comparison of NSGA-II and NSGA-III on various many-objective test problems.* In 2016 IEEE Congress on Evolutionary Computation (CEC), pages 3045–3052, July 2016. (Cited in page 69.)

[ITIC 2020] ITIC. *ITIC 2020 Global Server Hardware, Server OS Reliability Report*, 2020. (Cited in page 22.)

[ITU-T 2012] ITU-T. *Recommendation ITU-T Y. 2060: Overview of the Internet of Things.* Rapport technique, ITU-T, 2012. (Cited in page 10.)

[Jacob 2004] Bart Jacob, Richard Lanyon-Hogg, Devaprasad K Nadgir et Amr F Yassin. *A practical guide to the IBM autonomic computing toolkit.* IBM Redbooks, vol. 4, no. 10, pages 1–268, 2004. (Cited in pages xi, 23, 24 and 25.)

[Jang 1997] Jyh-Shing Roger Jang, Chuen-Tsai Sun et Eiji Mizutani. *Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [Book Review].* IEEE Transactions on automatic control, vol. 42, no. 10, pages 1482–1484, 1997. (Cited in page 26.)

[Jasper 2016] Cisco Jasper. *The hidden costs of delivering iiot services: Industrial monitoring & heavy equipment*, 2016. (Cited in page 22.)

[John 1994] George H John, Ron Kohavi et Karl Pfleger. *Irrelevant features and the subset selection problem.* In Machine Learning Proceedings 1994, pages 121–129. Elsevier, 1994. (Cited in page 90.)

[Jović 2015] Alan Jović, Karla Brkić et Nikola Bogunović. *A review of feature selection methods with applications.* In 2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO), pages 1200–1205. Ieee, 2015. (Cited in page 91.)

[Kaufmann 1996] Laurence Kaufmann. *Qu'est-ce que le virtuel ? (Pierre Levy).* Réseaux, volume 14, n°76, 1996. Le temps de l'événement II., 1996. (Cited in page 17.)

[Kephart 2003] Jeffrey Kephart, D Chess, Craig Boutilier, Rajarshi Das et William E Walsh. *An architectural blueprint for autonomic computing.* IBM White paper, pages 2–10, 2003. (Cited in pages 23, 78 and 106.)

[Kharb 2019] Seema Kharb et Anita Singhrova. *Fuzzy based priority aware scheduling technique for dense industrial IoT networks.* J. Netw. Comput. Appl., vol. 125, pages 17–27, 2019. (Cited in pages 37, 38 and 40.)

[Khazaei 2017] Hamzeh Khazaei, Hadi Bannazadeh et Alberto Leon-Garcia. *SAVI-IoT: A Self-Managing Containerized IoT Platform.* In 2017 IEEE 5th Int. Conf. Futur. Internet Things Cloud, pages 227–234. IEEE, aug 2017. (Cited in pages 37, 38 and 40.)

[Kim 2017] Sanghyeok Kim, Sungyoung Park, Youngjae Kim, Siri Kim et Kwonyong Lee. *VNF-EQ: dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV.* Cluster Comput., vol. 20, no. 3, pages 2107–2117, 2017. (Cited in pages 37, 38 and 40.)

[Kohavi 1997] Ron Kohavi, George H John*et al. Wrappers for feature subset selection.* Artificial intelligence, vol. 97, no. 1-2, pages 273–324, 1997. (Cited in page 93.)

[Kohler 2000] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti et M Frans Kaashoek. *The Click modular router.* ACM Transactions on Computer Systems (TOCS), vol. 18, no. 3, pages 263–297, 2000. (Cited in page 49.)

[Li 2018] Zhijing Li, Ben Y. Zhao, Haitao Zheng, Zihui Ge, Ajay Mahimkar, Jia Wang, Joanne Emmons et Laura Ogden. *Predictive analysis in network function virtualization.* Proc. ACM SIGCOMM Internet Meas. Conf. IMC, pages 161–167, 2018. (Cited in page 82.)

[Liu 2011] Huan Liu. *A measurement study of server utilization in public clouds.* In 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pages 435–442. IEEE, 2011. (Cited in page 44.)

[Liu 2017] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu et Zuqing Zhu. *On Dynamic service function chain deployment and readjustment.* IEEE Transactions on Network and Service Management, vol. 14, no. 3, pages 543–553, sep 2017. (Cited in page 48.)

[Lu 2017] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu et Tongxin Bai. *Imbalance in the cloud: An analysis on alibaba cluster trace.* In 2017 IEEE International Conference on Big Data (Big Data), pages 2884–2892. IEEE, 2017. (Cited in page 44.)

[Malkowski 2009a] Simon Malkowski, Markus Hedwig, Deepal Jayasinghe, Junhee Park, Yasuhiko Kanemasa et Calton 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, pages 1–10. IEEE, 2009. (Cited in page 90.)

[Malkowski 2009b] Simon Malkowski, Markus Hedwig et Calton Pu. *Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks.* In 2009 IEEE International Symposium on Workload Characterization (IISWC), pages 118–127. IEEE, 2009. (Cited in pages 81 and 90.)

[Mariani 2018] Leonardo Mariani, Cristina Monni, Mauro Pezze, Oliviero Riganelli et Rui Xin. *Localizing Faults in Cloud Systems.* Proc. - 2018 IEEE 11th Int. Conf. Softw. Testing, Verif. Validation, ICST 2018, pages 262–273, 2018. (Cited in page 86.)

[McKeown 2008] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker et Jonathan Turner. *OpenFlow: enabling innovation in campus networks.* ACM SIGCOMM computer communication review, vol. 38, no. 2, pages 69–74, 2008. (Cited in page 20.)

[Mendiboure 2019] Leo Mendiboure, Mohamed Aymen Chalouf et Francine Krief. *A SDN-Based Pub/Sub middleware for geographic content dissemination in internet of vehicles.* IEEE Veh. Technol. Conf., vol. 2019-Septe, pages 1–6, 2019. (Cited in pages 37, 38, 39 and 40.)

[Metzger 2019] Florian Metzger, Tobias Hobfeld, Andre Bauer, Samuel Kounev et Poul E. Heegaard. *Modeling of Aggregated IoT Traffic and Its Application to an IoT Cloud.* Proc. IEEE, vol. 107, no. 4, pages 679–694, 2019. (Cited in page 58.)

[Microsoft 2020] Microsoft. *IoT Signals Report*, 2020. (Cited in pages 22 and 44.)

[Morales 2019] Luis Morales, Clovis Anicet Ouedraogo, José Aguilar, Christophe Chassot, Samir Medjiah et Khalil 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, pages 199–219, 2019. (Cited in page 111.)

[Nandugudi 2016] A. Nandugudi, M. Gallo, D. Perino*et al. Network function virtualization: through the looking-glass.* Annals of Telecommunications, vol. 71, no. 11-12, pages 573–581, 2016. (Cited in pages 47 and 50.)

[Nastic 2016] Stefan Nastic, Hong-Linh Truong et Schahram Dustdar. *A Middleware Infrastructure for Utility-Based Provisioning of IoT Cloud Systems.* In 2016 IEEE/ACM Symp. Edge Comput., pages 28–40. IEEE, oct 2016. (Cited in pages 37, 38 and 40.)

[Natarajan 1995] Balas Kausik Natarajan. *Sparse approximate solutions to linear systems.* SIAM journal on computing, vol. 24, no. 2, pages 227–234, 1995. (Cited in page 93.)

[Newman 1998] Peter Newman, W Edwards, R Hinden, E Hoffman, F Ching Liaw, T Lyon et G Minshall. *Ipsilon's general switch management protocol specification version 2.0.* Rapport technique, Standards Track RFC 2297, Network Working Group, 1998. (Cited in page 19.)

[oneM2M 2016] oneM2M. *Technical SpecificationTS-0002-V2.7.1: Requirements.* oneM2M, vol. 1, pages 1–24, 2016. (Cited in pages 1, 2, 37, 44, 49, 56, 59, 79, 87, 107 and 114.)

[Ouedraogo 2018a] C. A. Ouedraogo, E. Bonfoh, S. Medjiah, C. Chassot et S. Yangui. *A Prototype for Dynamic Provisioning of QoS-oriented Virtualized Network Functions in the Internet of Things.* In 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), pages 323–325, June 2018. (Cited in page 72.)

[Ouedraogo 2018b] Clovis Anicet Ouedraogo, Samir Medjiah et Christophe Chassot. *A Modular Framework for Dynamic QoS Management at the Middleware Level of the IoT: Application to a oneM2M Compliant IoT Platform.* In IEEE Int. Conf. Commun., volume 2018-May, pages 1–7. IEEE, may 2018. (Cited in pages 72, 73 and 115.)

[Ouedraogo 2020] C. A. Ouedraogo, , Samir Medjiah, Christophe Chassot, Khalil Drira et José Aguilar. *A Cost-Effective Approach for End-to-End QoS Management in NFV-enabled IoT Platforms.* IEEE Internet of Things Journal, 2020. (Cited in pages 88 and 106.)

[Palkar 2015] S. Palkar, C. Lan, S. Han*et al.* *E2: a framework for NFV applications.* In Proceedings of the 25th Symposium on Operating Systems Principles, pages 121–136. ACM, 2015. (Cited in page 46.)

[Panda 2017] Aurojit Panda. *A New Approach to Network Function Virtualization.* PhD thesis, EECS Department, University of California, Berkeley, 2017. (Cited in page 50.)

[Park 2004] Kun I Park. Qos in packet networks, volume 779. Springer Science & Business Media, 2004. (Cited in pages xi and 31.)

[Petrov 2018] Vitaly Petrov, Maria A. Lema, Margarita Gapeyenko, Konstantinos Antonakoglou, Dmitri Moltchanov, Fragkiskos Sardis, Andrey Samuylov, Sergey Andreev,

Yevgeni Koucheryavy et Mischa Dohler. *Achieving End-to-End Reliability of Mission-Critical Traffic in Softwarized 5G Networks.* IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pages 485–501, mar 2018. (Cited in pages 37, 38, 39, 40 and 48.)

[Pfitscher 2019] Ricardo José Pfitscher, Arthur Selle Jacobs, Luciano Zembruzki, Ricardo Luis dos Santos, Eder John Scheid, Muriel Figueredo Franco, Alberto Schaeffer-Filho et Lisandro Zambenedetti Granville. *Guiltiness: A practical approach for quantifying virtual network functions performance.* Comput. Networks, vol. 161, pages 14–31, 2019. (Cited in page 86.)

[Pister 2009] Kris Pister, Pascal Thubert, Sicco Dwars et Tom Phinney. *Industrial routing requirements in low-power and lossy networks.* Citeseer, 2009. (Cited in page 36.)

[Pizzolli 2016] Daniele Pizzolli, Giuseppe Cossu, Daniele Santoro, Luca Capra, Corentin Dupont, Dukas Charalampos, Francesco De Pellegrini, Fabio Antonelli et Silvio Cretti. *Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT.* In 2016 IEEE Int. Conf. Cloud Comput. Technol. Sci., pages 476–479. IEEE, dec 2016. (Cited in pages 37, 38 and 40.)

[Qazi 2014] Zafar Ayyub Qazi, Vyas Sekar et Samir Das. *A framework to quantify the benefits of network functions virtualization in cellular networks.* arXiv preprint arXiv:1406.5634, 2014. (Cited in page 20.)

[Qiu 2018] Juan Qiu, Qingfeng Du, Yu He, YiQun Lin, Jiaye Zhu et Kanglin Yin. *Performance anomaly detection models of virtual machines for network function virtualization infrastructure with machine learning.* In International Conference on Artificial Neural Networks, pages 479–488. Springer, 2018. (Cited in page 82.)

[Qu 2018] Chenhao Qu, Rodrigo N Calheiros et Rajkumar Buyya. *Auto-scaling web applications in clouds: A taxonomy and survey.* ACM Computing Surveys (CSUR), vol. 51, no. 4, page 73, 2018. (Cited in pages 44 and 47.)

[Quang 2018] Pham Tran Anh Quang, Kamal Deep Singh, Abbas Bradai et Abderrahim Benslimane. *QAAV: Quality of Service-Aware Adaptive Allocation of Virtual Network Functions in Wireless Network.* In IEEE International Conference on Communications, volume 2018-May, pages 1–6, 2018. (Cited in page 48.)

[Rahman 2018] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore et Biswanath Mukherjee. *Auto-Scaling VNFs Using Machine Learning to Improve QoS and Reduce Cost.* In IEEE International Conference on Communications, volume 2018-May, pages 1–6. IEEE, may 2018. (Cited in pages 47 and 48.)

[Rahman 2020] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore et Biswanath Mukherjee. *Auto-Scaling Network Service Chains using Machine Learning and Negotiation Game.* IEEE Transactions on Network and Service Management, 2020. (Cited in pages 47 and 48.)

[Rec 1994] ITUT Rec. *E. 800: Definitions of terms related to quality of service.* International Telecommunications Union, Geneva, 1994. (Cited in pages 1 and 33.)

[Reiss 2012] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz et Michael A Kozuch. *Heterogeneity and dynamicity of clouds at scale: Google trace analysis.* In Proceedings of the Third ACM Symposium on Cloud Computing, page 7. ACM, 2012. (Cited in page 44.)

[Ren 2017] Jianbao Ren, Yong Qi, Yuehua Dai, Yu Xuan et Yi Shi. *Nosv: A lightweight nested-virtualization VMM for hosting high performance computing on cloud.* Journal of Systems and Software, vol. 124, pages 137 – 152, 2017. (Cited in page 49.)

[Ren 2018] Yi Ren, Tuan Phung-Duc, Yi-Kuan Liu, Jyh-Cheng Chen et Yi-Hao Lin. *ASA: Adaptive VNF scaling algorithm for 5G mobile networks.* In 2018 IEEE 7th international conference on cloud networking (CloudNet), pages 1–4. IEEE, 2018. (Cited in pages 47 and 48.)

[Reunanen 2006] Juha Reunanen. Search strategies, pages 119–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. (Cited in page 93.)

[Riggio 2015] R. Riggio, M. K. Marina, J. Schulz-Zander*et al. Programming Abstractions for Software-Defined Wireless Networks.* IEEE Trans. Network and Service Management, vol. 12, no. 2, pages 146–162, 2015. (Cited in page 46.)

[Salimans 2017] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor et Ilya Sutskever. *Evolution strategies as a scalable alternative to reinforcement learning.* arXiv preprint arXiv:1703.03864, 2017. (Cited in page 30.)

[Santos 2017] José Santos, Tim Wauters, Bruno Volckaert et Filip De Turck. *Resource provisioning for IoT application services in smart cities.* 2017 13th Int. Conf. Netw. Serv. Manag. CNSM 2017, vol. 2018-January, pages 1–9, 2017. (Cited in pages 37, 38, 39 and 40.)

[Sauvanaud 2016] Carla Sauvanaud, Kahina Lazri, Mohamed Kaaniche et Karama Kanoun. *Anomaly Detection and Root Cause Localization in Virtual Network Functions.* Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE, pages 196–206, 2016. (Cited in page 85.)

[Sauvanaud 2018] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri et Guthemberg Da Silva Silvestre. *Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned.* J. Syst. Softw., vol. 139, pages 84–106, 2018. (Cited in page 85.)

[Schmidt 2018] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschlager, Alexander Acker et Odej Kao. *Unsupervised anomaly event detection for VNF service monitoring using multivariate online arima.* Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom, vol. 2018-December, pages 278–283, 2018. (Cited in page 82.)

[Shi 2020a] Yulong Shi, Jonathon Wong, Hans Arno Jacobsen, Yang Zhang et Junliang Chen. *Topic-Oriented Bucket-Based Fast Multicast Routing in SDN-Like Publish/Subscribe Middleware.* IEEE Access, vol. 8, pages 89741–89756, 2020. (Cited in pages 37, 38 and 40.)

[Shi 2020b] Yulong Shi, Yang Zhang et Junliang Chen. *Cross-layer QoS enabled SDN-like publish/subscribe communication infrastructure for IoT.* China Commun., vol. 17, no. 3, pages 149–167, 2020. (Cited in pages 37, 38 and 40.)

[Shih 2016] Ming-Wei Shih, Mohan Kumar, Taesoo Kim et Ada Gavrilovska. *S-nfv: Securing nfv states by using sgx.* In Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pages 45–48. ACM, 2016. (Cited in page 20.)

[Solé 2017] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana et Giovani Estrada. *Survey on models and techniques for root-cause analysis.* arXiv preprint arXiv:1701.08546, 2017. (Cited in page 85.)

[Stiliadis 1998] Dimitrios Stiliadis et Anujan Varma. *Latency-rate servers: a general model for analysis of traffic scheduling algorithms.* IEEE/ACM Transactions on networking, vol. 6, no. 5, pages 611–624, 1998. (Cited in page 64.)

[Szymański 2017] P. Szymański et T. Kajdanowicz. *A scikit-based Python environment for performing multi-label classification.* ArXiv e-prints, Février 2017. (Cited in page 100.)

[Tang 2015] Pengcheng Tang, Fei Li, Wei Zhou, Weihua Hu et Li Yang. *Efficient auto-scaling approach in the telco cloud using self-learning algorithm.* In 2015 IEEE Global Communications Conference (GLOBECOM), pages 1–6. IEEE, 2015. (Cited in pages 47 and 48.)

[Tariq 2014] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik et Kurt Rothermel. *PLEROMA: A SDN-based high performance publish/subscribe middleware.* Proc. 15th

Int. Middlew. Conf. Middlew. 2014, pages 217–228, 2014. (Cited in pages 37, 38, 39 and 40.)

[Tola 2019] Besmir Tola, Gianfranco Nencioni et Bjarne E. Helvik. *Network-Aware Availability Modeling of an End-to-End NFV-Enabled Service.* IEEE Trans. Netw. Serv. Manag., vol. 16, no. 4, pages 1389–1403, 2019. (Cited in page 86.)

[Toosi 2019] Adel Nadjaran Toosi, Jungmin Son, Qinghua Chi et Rajkumar Buyya. *ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds.* Journal of Systems and Software, 2019. (Cited in page 48.)

[Tootoonchian 2018] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy et Scott Shenker. *Resq: Enabling slos in network function virtualization.* In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18), pages 283–297, 2018. (Cited in page 45.)

[Tsoumakas 2009] Grigorios Tsoumakas, Ioannis Katakis et Ioannis Vlahavas. *Mining multi-label data.* In Data mining and knowledge discovery handbook, pages 667–685. Springer, 2009. (Cited in pages 91 and 102.)

[Tsoumakas 2010] Grigorios Tsoumakas, Ioannis Katakis et Ioannis Vlahavas. *Random k-labelsets for multilabel classification.* IEEE transactions on knowledge and data engineering, vol. 23, no. 7, pages 1079–1089, 2010. (Cited in page 106.)

[Van der Merwe 1998] Jacobus E Van der Merwe, Sean Rooney, L Leslie et Simon Crosby. *The tempest-a practical framework for network programmability.* IEEE network, vol. 12, no. 3, pages 20–28, 1998. (Cited in page 19.)

[Virtualisation 2012] Network Functions Virtualisation. *An introduction, benefits, enablers, challenges & call for action.* In White Paper, SDN and OpenFlow World Congress, 2012. (Cited in page 20.)

[Waller 2014] Jan Waller. *Performance Benchmarking of Application Monitoring Frameworks.* Softwaretechnik-trends, vol. 35, 2014. (Cited in page 82.)

[Wang 2018] Tao Wang, Jiwei Xu, Wenbo Zhang, Zeyu Gu et Hua Zhong. *Self-adaptive cloud monitoring with online anomaly detection.* Future Generation Computer Systems, vol. 80, pages 89 – 101, 2018. (Cited in page 82.)

[Weng 2018] Jianping Weng, Jessie Hui Wang, Jiahai Yang et Yang Yang. *Root Cause Analysis of Anomalies of Multitier Services in Public Clouds.* IEEE/ACM Trans. Netw., vol. 26, no. 4, pages 1646–1659, 2018. (Cited in page 85.)

[White 2017] Gary White, Vivek Nallur et Siobhán Clarke. *Quality of service approaches in IoT: A systematic mapping.* J. Syst. Softw., vol. 132, pages 186–203, 2017. (Cited in page 81.)

[Yager 1994] Ronald R Yager et Loftia A Zadeh. *Fuzzy Sets, Neural Networks and Soft Computing*, 1994. (Cited in page 26.)

[Yan 2012] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei et Jennifer 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, pages 1734–1747, 2012. (Cited in page 85.)

[Yang 2004] Lily Yang, Ram Dantu, Terry Anderson et Ram Gopal. *Forwarding and control element separation (ForCES) framework*, 2004. (Cited in page 19.)

[Yasukata 2017] K. Yasukata, F. Huici, V. Maffione*et al. HyperNF: building a high performance, high utilization and fair NFV platform.* In Proceedings of the 2017 Symposium on Cloud Computing, pages 157–169. ACM, 2017. (Cited in page 46.)

[Yu 2017] Bangchao Yu, Wei Zheng, Xiangming Wen, Zhaoming Lu, Luhan Wang et Lu Ma. *Dynamic resource orchestration of service function chaining in network function virtualizations.* In International Conference on 5G for Future Wireless Networks, pages 132–145. Springer, 2017. (Cited in page 48.)

[Yu 2019] Guang Yu, Zhiping Cai, Siqi Wang, Haiwen Chen, Fang Liu et Anfeng Liu. *Unsupervised Online Anomaly Detection with Parameter Adaptation for KPI Abrupt Changes.* IEEE Trans. Netw. Serv. Manag., vol. PP, no. c, page 1, 2019. (Cited in page 82.)

[Zadeh 1993] Lotfi A Zadeh. *Fuzzy logic, neural networks and soft computing.* In Safety evaluation based on identification approaches related to time-variant and nonlinear structures, pages 320–321. Springer, 1993. (Cited in page 26.)

[Zhang 2013] Min-Ling Zhang et Zhi-Hua Zhou. *A review on multi-label learning algorithms.* IEEE transactions on knowledge and data engineering, vol. 26, no. 8, pages 1819–1837, 2013. (Cited in page 100.)

[Zhang 2018] Lei Zhang, Xiaorong Zhu, Su Zhao et Ding Xu. *A novel virtual network fault diagnosis method based on long short-term memory neural networks.* IEEE Veh. Technol. Conf., vol. 2017-Septe, pages 1–5, 2018. (Cited in page 86.)

[Zhou 2018] Honggang Zhou, Yunchun Li, Hailong Yang, Jie Jia et Wei Li. *BigRoots: An Effective Approach for Root-Cause Analysis of Stragglers in Big Data System.* IEEE Access, vol. 6, pages 41966–41977, 2018. (Cited in page 85.)

[Zitzler 2001] Eckart Zitzler, Marco Laumanns et Lothar Thiele. *SPEA2: Improving the strength Pareto evolutionary algorithm.* TIK-report, vol. 103, 2001. (Cited in page 68.)