Universidad del Cauca

Adaptive Scaling in Network Functions Virtualization

Carlos Hernán Tobar Arteaga

Universidad del Cauca
Faculty of Electronics and Telecommunications Engineering
Line of Research in Advanced Services of Telecommunications
Popayán, Colombia
2021

Adaptive Scaling in Network Functions Virtualization

Carlos Hernán Tobar Arteaga

Thesis presented as a partial requirement to qualify for the title of:
Doctor in Telematics Engineering

Advisor: Ph.D. Oscar Mauricio Caicedo Rendón

Line of Research:
Advanced Services of Telecommunications

Universidad del Cauca
Faculty of Electronics and Telecommunications Engineering
Popayán, Colombia
2021

To

My parents and my sister.

Acknowledgments

A special thanks to the Ph.D. Oscar Mauricio Caicedo Rendón, professor of the Universidad del Cauca, for his valuable advises during my studies.

Abstract

Nowadays, the increase in network traffic, applications, and users imposes challenges in the scalability and adaptability of network functions and services. Traditional scaling solutions oversize statically the capacity of network appliances, designing them to support workload peaks. However, static scaling is inefficient at periods of low utilization. The Network Functions Virtualization (NFV) offers an alternative solution to static scaling, enabling dynamically to modify the capacity of network functions. NFV-based scaling has been proposed in several works. However, still there are open research questions on the NFV scaling decision-making process to be investigated, such as the precision on which, how, and when to scale network functions. This thesis hypothesizes that reinforcement learning can offer a valuable approach to develop scaling in NFV, and pursues the main objective of introducing scaling mechanisms for adaptive management of traffic and performance variations in NFV.

Resumen

En la actualidad, el aumento del tráfico, las aplicaciones y los usuarios imponen desafíos en la escalabilidad y adaptabilidad de las funciones y servicios de la red. Las soluciones de escalado tradicionales sobredimensionan estáticamente la capacidad de los dispositivos de red y los diseñan para soportar picos de carga de trabajo. Sin embargo, el escalado estático es ineficaz en períodos de baja utilización. La Virtualización de Funciones de Red (NFV) ofrece una solución alternativa al escalado estático, permitiendo modificar dinámicamente la capacidad de las funciones de red. El escalado basado en NFV se ha propuesto en varios trabajos. Sin embargo, todavía hay preguntas de investigación abiertas sobre el proceso de toma de decisiones de escalado de NFV que deben investigarse, como la precisión sobre cuáles, cómo y cuándo escalar las funciones de red. Esta tesis plantea la hipótesis de que el aprendizaje por refuerzo puede ofrecer un enfoque valioso para desarrollar el escalado en NFV, y persigue el objetivo principal de introducir mecanismos de escalado para la gestión adaptativa del tráfico y las variaciones de rendimiento en NFV.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Term**
$4G$	Fourth Generation
$5G$	Fifth Generation
$5GCN$	5G Core Network
$5GNSL$	5G Network Slice
AMF	Access and Mobility Management Function
$AUSF$	Authentication Server Function
CPU	Central Processing Unit
$eMBB$	enhanced Mobile BroadBand
EPC	Evolved Packet Core
IAT	Inter-Arrival Time
ILP	Integer Linear Programming
IMS	IP Multimedia Subsystem
IoT	Internet of Things
IP	Internet Protocol
LQN	Layered Queuing Network
LTE	Long Term Evolution
$MANO$	Management and Orchestration
$MARL$	Multi-Agent Reinforcement Learning
MDP	Markov Decision Processes
MEC	Multi-access Edge Computing
$mIoT$	massive Internet of Things
MME	Mobility Management Entity
$mMTC$	massive Machine Type Communications
NF	Network Function

Acronym	**Term**
NFV	Network Functions Virtualization
NRF	Network Repository Function
NS	Network Service
NSC	Network Service Chaining
NSP	Network Service Provider
PEPA	Performance Evaluation Process Algebra
PGW	Packet Data Network Gateway
QoS	Quality of Service
QPN	Queuing Petri Nets
RAM	Random Access Memory
RL	Reinforcement Learning
SDN	Software-Defined Networking
SGW	Serving Gateway
SMF	Session Management Function
UE	User Equipment
UPF	User Plane Function
URLLC	Ultra-Reliable Low-Latency Communication
VM	Virtual Machine
vMME	virtualized Mobility Management Entity
VNF	Virtualized Network Fuction
VUE	Vehicular User Equipment

1. Introduction

Nowadays, telecommunications network operators face different challenges related to the management of their networks and services. These challenges are motivated by the increase in network traffic, applications and users. According to Cisco projections (Cisco, 2020), by 2023 nearly two-thirds of the global population will have Internet access. The number of devices connected to Internet protocol (IP) networks will be more than three times the global population, and Internet of Things (IoT) connections will be half of the global connections. Over 70% of the global population will have mobile connectivity, being devices and connections of the fifth generation (5G) mobile networks over 10 percent of global mobile devices and connections. Modern applications will be in every business sector, experimenting an increased demand for new and enhanced ones that improves the customer experiences. IoT, artificial intelligence, machine learning, and business analytics are enabling smart applications to simplify customer transactions, help in health and medicine, assist daily task in the industry and automotive, and offer new entertainment contents.

Modern applications will have stringent requirements in terms of data transfer velocity, latency, capacity, and reliability. For instance, 5G networks envision to support different use cases, such as enhanced Mobile BroadBand (eMBB), massive IoT (mIoT), and Ultra-Reliable Low-Latency Communication (URLLC). eMBB addresses human-centric applications for access to multi-media content, services and data. mIoT is characterized by a very large number of connected devices typically transmitting a relative low volume of non-delay-sensitive data. URLLC includes, as examples, wireless control of industrial manufacturing or production process, remote medical surgery, and distribution automation in a smart grid. 5G networks must outperform previous generation systems. Performance requirements of 5G networks are: data rates per user of 0.1-1 $Gbps$, connection density of 1 million of devices per km^2, maximum end-to-end latency of 1 ms, traffic volume density of tens of $Gbps$ per km^2, mobility higher than 500 km per $hour$, and peak data rate of tens of $Gbps$ (Xiang et al., 2017).

Traditional development in networking involves that network appliances integrate network control functions with forwarding hardware vertically, with little or nothing programmability and proprietary interfaces. This traditional development paradigm have caused network operators to adopt error-prone manual configuration methods to provide and manage network resources, leading to more operational complexity and longer market time for new services. These problems are the major obstacle in scalability and adaptability of the network infrastructure with user and application needs in evolution (Chowdhury, 2021). Traditional scaling solutions oversize the capacity of network appliances designing them to support workload peaks. However, this static scaling is inefficient at periods of low utilization.

To deal with traditional development problems in networking, a new networking development paradigm, known as Network Softwarization, is being adopted. The primary technologies in Network Softwarization are Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) (Chowdhury, 2021). Particularly, NFV decouples Network Functions (NFs) from hardware middleboxes, and deploy the NFs as Virtualized Network Functions (VNFs) on commodity servers. Through the separation of NFs from proprietary hardware middleboxes, NFV promises to reduce capital investment by consolidating multiple NFs on the same commodity hardware, and reduce operational cost by leveraging advances in application orchestration for on-demand service provisioning. NFV offers an alternative solution to the static scaling, enabling dynamically modify the capacity of NFs. Scaling in NFV can be performed by increasing and reducing the NFs resources (*i.e.,* scaling up/down) or creating and removing their instances (*i.e.,* scaling out/in). It is noteworthy that scaling can be initiated by administrators, as an outcome of the network performance assessment, or by the network itself by using adaptive mechanisms (ETSI-GS-AFI, 2013).

In the NFV literature, several works have proposed scaling mechanisms by using different techniques. For instance, (Carella et al., 2016, Dutta et al., 2016) are based on threshold rules, a reactive technique, in which the scaling depends on the current traffic and performance levels. Performance variations, crossing predefined thresholds, can lead to violations of a performance target and transitory scaling oscillations. (Bilal et al., 2016a) uses time series forecasting that, based on historical data, enables to predict future resource usage; however, if there are changes in traffic patterns, an evolutionary strategy would be desirable, which would allow adapting the models to new traffic conditions. (Tang et al., 2015) is based on the Q-Learning method in which, as other methods in Reinforcement Learning (RL), an agent interacts with an environment and learns by trial and error; this method is adaptive but mistaken decisions may be taken until the agent learns an optimal scaling policy.

In addition to the drawbacks indicated, these works (Carella et al., 2016, Dutta et al., 2016, Bilal et al., 2016a, Tang et al., 2015) focus mainly on the scaling of individual NFs and, so, they do not consider the issues related to scale Network Services (NSs) formed by diverse NFs. In fact, in real network deployments, it is usual to have NSs composed by several NFs. For example, considering the control plane of the Evolved Packet Core (EPC) as an NS, three entities interact to perform signaling procedures which are the Mobility Management Entity (MME), the Serving Gateway (SGW) and the Packet Data Network Gateway (PGW). Related to composed NSs, (Wang et al., 2016) assumes all NFs of an NS must be scaled and introduces a factor that models the change in the traffic rate for each NF. (Zhang et al., 2016) focuses only on the flow migration and assumes all NFs must be scaled. In (Rankothge et al., 2017), only one NF is scaled out/in, which is chosen randomly. Works for compound NSs show that the scaling problem is not addressed enough yet. NFs can be dependent each other and the NS performance is not stationary, *i.e.* scaling one NF changes the state space, which leads to the whole performance cannot be maintained in a target range.

The above drawbacks reveal that the NFV scaling decision-making process needs precision on which NFs must be scaled, how to scale and when to trigger the scaling, allowing NSs to ensure an expected Quality of Service (QoS) and the efficient use of resources. Consider these issues. *i)* Which NFs. If a single NF is forming an NS (*e.g.,* a video server), there is an obvious answer to the question: which NFs should be scaled?. However, as in EPC of the previous example, the signaling NS has several NFs (*e.g.,* MME, SGW, and PGW), and therefore, the answer to the above question is not immediately. Choosing which NF depends on current workload, amount of allocated resources, and tasks complexity. Also, EPC NFs perform in synchronous mode; it means, NFs are interrelated, and so, their scaling. *ii)* How to scale refers to which scaling type is required; it means choosing between scaling out/in, scaling up/down, or both. *iii)* The third issue is regarding the instant in which scaling must start, taking into account that a predictability feature could be desirable. Enabling predictability would give to scaling performs more accurately. Thus, the research question raised in this thesis is: which, how, and when scale NFs in NFV?.

Considering the recent works on the use of machine learning in network management (Ayoubi et al., 2017, Mestres et al., 2017, Fadlullah et al., 2017, Wang et al., 2017, Moysen and Giupponi, 2018, Cui et al., 2018, Kim et al., 2019, Sultana et al., 2019, Sesto-Castilla et al., 2019, Liu et al., 2020, Shafik et al., 2020, Hussain et al., 2020, Fourati et al., 2021, Mollel et al., 2021), this thesis hypothesizes that RL and Multi-Agent Reinforcement Learning (MARL) can offer a valuable approach to develop scaling mechanisms in NFV. Comprehensive performance characterizations of NFV-based networks should also be performed by heuristics approaches or using performance models (*e.g.,* stochastic process algebra for performance evaluation) to realize practical and novel scaling mechanisms. Furthermore, scaling mechanisms should be accurate enough determining the appropriate method (out/in, up/down, or the both) and which NFs to scale, as well as the time for triggering the scaling.

The general objective of this thesis is to introduce scaling mechanisms for adaptive management of traffic and performance variations in NFV. The three specific objectives given below allow achieving the overall objective.

1. Characterize the performance of NSs (*e.g.,* services of control and data in the EPC) in NFV.

2. Model scaling mechanisms for adaptive management of traffic and performance varia-

tions in NFV by using the RL approach.

3. Evaluate the improvement in managing on traffic and performance variations when the scaling mechanisms are used.

1.1. Contributions and Scientific Production

The contributions of this thesis are:

- The performance characterization and the scalability analysis of an NFV-based EPC. It has been extensively evaluated the performance and scalability of an NFV-based EPC prototype. The evaluation results reveal the importance of using horizontal and vertical scaling to improve EPC performance to handle workload variations and save resources.
- An adaptive scaling mechanism for an NFV-based EPC aiming at manage performance variations. This mechanism is based on the RL approach and Gaussian processes (GPs) and allows handling the mean response time of EPC due to service request variations.
- An adaptive scaling mechanism for the NFV-based 5G Core Network (5GCN) control plane that follows a cooperative approach using MARL.
- The scalability and performance analysis of the NFV-based 5GCN control plane, which is modeled using the Performance Evaluation Process Algebra (PEPA). The scalability analysis considers concurrent users and the multiplicity of VNFs. For performance analysis, this thesis introduces new composite structures based on PEPA and intends to model and evaluate 5GCN procedures.

These contributions are available as three papers published and another one on review, which are listed below.

- **C. H. T. Arteaga**, F. Risso and O. M. C. Rendon, "**An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an NFV-based EPC**," 2017 13th **International Conference on Network and Service Management (CNSM)**, Tokyo, Japan, 2017, pp. 1-7, doi: 10.23919/CNSM.2017.8255982. Classification: **CORE B**.
- **C. H. T. Arteaga**, F. B. Anacona, K. T. T. Ortega and O. M. C. Rendon, "**A Scaling Mechanism for an Evolved Packet Core Based on Network Functions Virtualization**," in **IEEE Transactions on Network and Service Management**, vol. 17, no. 2, pp. 779-792, June 2020, doi: 10.1109/TNSM.2019.2961988. Classification: **JCR Q1**.

- **C. H. T. Arteaga**, A. Ordoñez and O. M. C. Rendon, "**Scalability and Performance Analysis in 5G Core Network Slicing**," in **IEEE Access**, vol. 8, pp. 142086-142100, August 2020, doi: 10.1109/ACCESS.2020.3013597. Classification: **JCR Q1**.
- **C. H. T. Arteaga**, O. M. C. Rendon, "**Cooperative Scaling for the 5G Core Network Control Plane**," To be submitted to Journal of Network and Computer Applications - Elsevier. Classification: **JCR Q1**.

1.2. Methodology and Organization

This thesis followed the scientific method, which is depicted in Fig. **1-1**. This method has five phases: problem definition, hypothesis construction, experimentation, conclusion, and publishing of findings. Problem definition identified the research question. Hypothesis construction formulated the hypothesis, also this phase involved the development of conceptual and technological proposals. Experimentation tested and analyzed the hypothesis and the results achieved. The conclusion allowed generating the conclusions and outlining the future works. Note that the conclusion phase allows to feedback the phase of hypothesis construction. In the last phase, publishing of findings, research papers were submitted to conferences and journals.

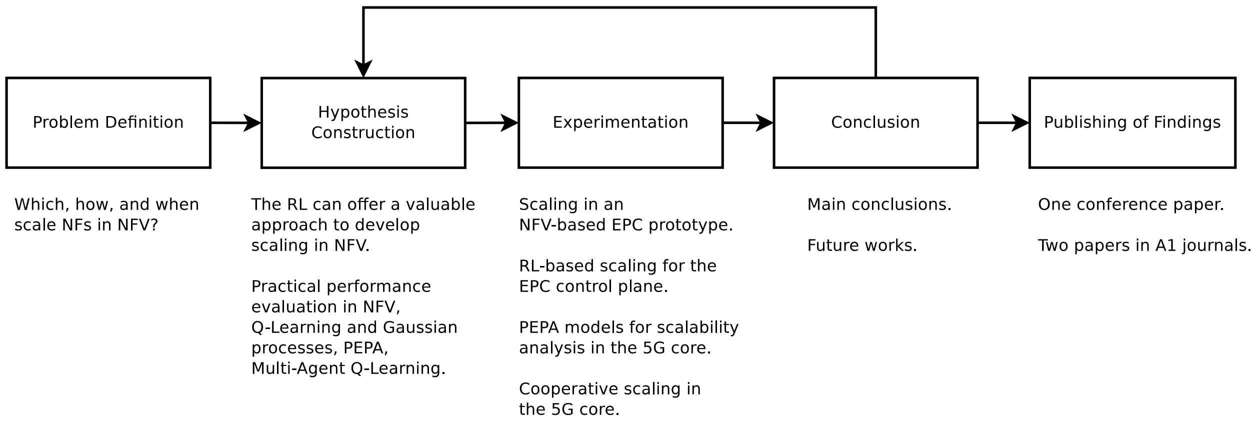

Problem Definition
Hypothesis Construction
Experimentation
Conclusion
Publishing of Findings
Which, how, and when scale NFs in NFV?
The RL can offer a valuable approach to develop scaling in NFV.
Practical performance evaluation in NFV, Q-Learning and Gaussian processes, PEPA, Multi-Agent Q-Learning.
Scaling in an NFV-based EPC prototype.
RL-based scaling for the EPC control plane.
PEPA models for scalability analysis in the 5G core.
Cooperative scaling in the 5G core.
Main conclusions.
Future works.
One conference paper.
Two papers in A1 journals.

Figure 1-1.: The scientific method followed in this thesis.

The organization of this document reflects the phases outlined above.

- This introductory chapter presents the problem definition, raises the hypothesis, defines the objectives of this thesis, summarizes the contributions and the scientific production, and outlines the methodology and the overall structure of this document.
- Chapter 2 develops the background and related work of this thesis.

- Chapter 3 presents a scaling mechanism for an NFV-based EPC.
- Chapter 4 presents an adaptive scaling mechanism for managing performance variations in NFV, as a case study in an NFV-based EPC.
- Chapter 5 presents a cooperative scaling mechanism for the 5GCN control plane.
- Chapter 6 presents a method for scalability and performance analysis in 5GCN slicing.
- Chapter 7 presents conclusions about the hypothesis and proposes some future works.
- Appendix A includes the scientific papers published and on review.

2. State-of-the-Art

This chapter presents, in its first part, the background of the concepts and technologies involved in this thesis. The second part of this chapter discusses the related work of this research.

2.1. Background

This background presents, first, concepts of network performance, scalability, and NFV. Second, Telco networks technologies: virtualized EPC (vEPC) and 5GCN. Third, concepts and models of RL for single-agent and multi-agents, GPs for system identification, and PEPA.

2.1.1. Network Performance and Scalability

Network performance refers to the behavior and QoS measures of a network as viewed by its users. Common performance metrics are throughput, latency, jitter, bit error rate, and packet loss (ETSI, 2014b). Network performance does not remain constant over time, and its variability depends on four factors (Callegati et al., 2014, Shea et al., 2014):

- The dynamic behavior of network resources (*e.g.*, a link between two nodes is down for a short while).
- The burstiness of network traffic (*e.g.*, a video streaming is started before it is playbacked).
- The incremental traffic during rush hours (*e.g.*, two busy hours are observed, around 12:00 pm and 6:00 pm).
- The sharing of resources of storage, processing and networking that may lead to performance fluctuations (*e.g.*, throughput instabilities may be present even at low load).

Performance variations impact applications negatively, particularly those that are data-intensive, and therefore performance management is a primary task (Chen and Zhang, 2014) that aims to provide functionalities to monitor and control the behaviour and effectiveness of networks (ITU, 2000). Performance management has three scopes: monitoring, characterization, and dynamic actuation.

- Monitoring is to collect data of measurable parameters (*e.g.,* throughput and delay) (ETSI, 2014b) that can be used to verify the physical and logical configuration of networks. Also, monitoring allows locating potential problems as early as possible (Li and Shen, 2011).

- Characterization allows assessing the performance of NSs to determine configurations that offer maximum performance (Cao et al., 2015), which is useful to plan and deploy new NSs (Moens and Turck, 2014).

- Dynamic actuation is to modify the configuration of NSs in response to workload and performance variations, which enables an adaptive behavior (Bilal et al., 2016b).

A traditional approach to address performance variations is to create NSs dimensioned for the highest workload, but this approach results in over costs (Heidari and Kanso, 2016, Gunasekaran et al., 2020, Zhang et al., 2021, Kumar et al., 2021). Therefore, novel mechanisms are needed for dynamic scaling (*i.e.,* scaling out/in and scaling up/down) of NSs, aiming to meet an expected performance, decrease costs of operation and administration as well as improve the use of computing, networking, and processing resources (Bilal et al., 2016b, Sharma et al., 2020, Luo and Wu, 2020, Singh et al., 2021). Scalability is the capacity of a network to adapt and respond to workload and performance variations (Becker et al., 2017, Henning and Hasselbring, 2021). There are two scaling methods (Coutinho et al., 2015, Becker et al., 2017, Balla et al., 2020): horizontal and vertical. Horizontal scaling (also called scaling out/in) is about creating or removing one or more instances of NFs (*e.g.,* more Virtual Machines (VMs) running an NF). Vertical scaling (or scaling up/down) is about increasing or reducing the capacity of the instances (*e.g.,* more memory, processing, and storage for a VM running an NF). Scaling out/up is useful when performance is degraded because the load of NSs is increased and scaling in/down is helpful when a determined level of performance may be supported with fewer resources or instances of NFs. Enabling horizontal and vertical scaling simultaneously allows full elasticity of NSs (Nguyen et al., 2020).

2.1.2. Network Functions Virtualization

Traditional networks are composed of diverse NFs deployed in specialized proprietary hardware, commonly called network appliances or middleboxes. This traditional development approach conveys several problems related to reduced flexibility, high operational and capital expenditure, and slow innovation for new NSs.

NFV is an emerging network solution that seeks to overcome the disadvantages of the traditional approach of development, deployment, and operation based on network appliances. NFV is a network architecture specified by ETSI (ETSI, 2013) that aims the deployment of NSs flexibly and dynamically (Chiosi et al., 2012, Han et al., 2015), enabling scalability and adaptability of the network infrastructure with user and application needs in evolution (Chowdhury, 2021). NFV has been conceived by Telco providers for enabling a major transformation of current and future telecommunication networks, such as EPC and 5G and 6G networks (Tola, 2021).

NFV envisions the implementation of NFs as software running in virtualized environments, which is decoupled from the underlying hardware and can be instantiated in different locations without the need for installation of new vendor equipment. This implementation approach aims to transform how network operators architect, operate, and manage networks by leveraging server virtualization technology for consolidating network appliances onto standard high volume servers, switches, and storage equipment, which can be deployed in datacenters, network nodes, and end user promises. NFV brings benefits to network operators, such as lower capital expenditures, by eliminating the need to purchase costly specialized network appliances, reduced operating cost as through a centralization of the network management, and greater flexibility and scalability since it will require much less time and work to add new capabilities in the network (Yi et al., 2018).

NFV envisages the implementation of NFs as software-based entities that runs over a virtualized infrastructure constituted by compute, storage, and networking resources. VNFs interact each other to provide end-to-end NSs. NSs are complete end-to-end functionalities offered by network operators, which are delivered by composing NFs through a process called Network Service Chaining (NSC). An NS can be described by a Forwarding Graph of interconnected NFs and end points (John et al., 2013, Bhamare et al., 2016).

Fig. **2-1** represents the primary subsystems in NFV, which are VNFs, the NFV Infrastructure (NFVI), and the NFV Management and Orchestration (MANO) (ETSI, 2013). A VNF is the software implementation of an NF (*e.g.,* firewall and deep packet inspection), which can be deployed in virtual resources such as VMs or containers. A VNF can be decomposed into smaller functional modules for scalability, reusability, and faster response, or multiple VNFs can be composed together to reduce management and VNF traffic steering complexity (ETSI, 2013).

NFVI is the set of hardware and software resources that constitute the environment where VNFs are executed. The physical resources include high volume industry standard equipment providing computing, storage, and network hardware resources. Virtual resources are abstracted counterpart of computing, storage, and network resources. This abstraction is achieved using a virtualization layer, which decouples the virtual resources from the underlying physical resources. Typical virtualization technologies, where VNF can be executed, can be based on a hypervisor or containerized infrastructure (ETSI, 2013).

MANO framework is responsible for managing NFV including components such as NSs, NFs and NSC as well as physical and virtual infrastructures. The VNF Manager is in charge of the instantiation, scaling, termination and triggering of events during the lifecycle of a VNF, and supports automatic configuration (Cao et al., 2015, ETSI, 2014a). As is common in traditional networks, performance management is also critical in NFV-based networks (Mijumbi et al., 2016a).

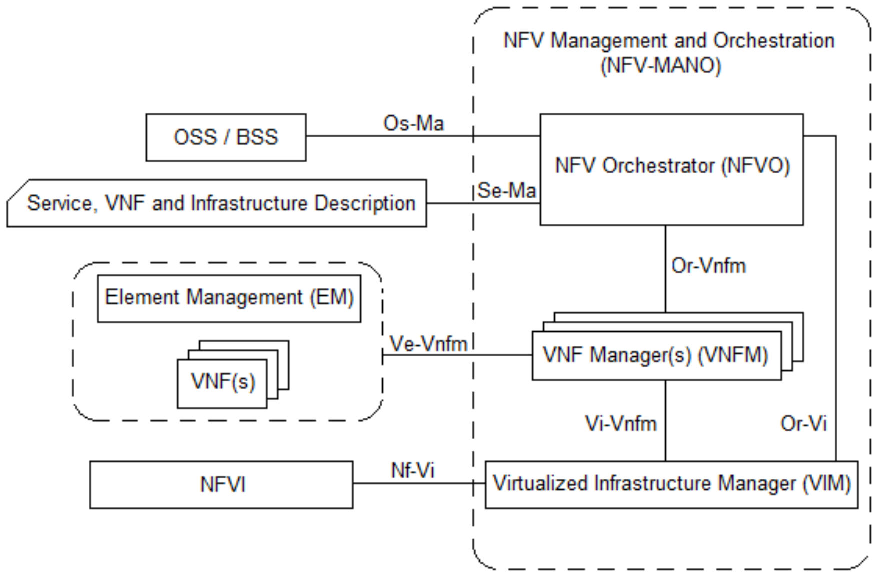


Figure 2-1.: High-level NFV architecture.

2.1.3. Virtualized Evolved Packet Core

The current standard of fourth generation (4G) mobile networks is the Long Term Evolution (LTE), and its core is the EPC (3GPP, 2018a). EPC is responsible for supporting traffic from multiple access networks by using four primary entities: MME, Home Subscriber Server (HSS), SGW, and PGW (Kempf et al., 2012). MME is the control entity in charge of managing the authentication and configuration of the User Equipment (UE) session, as well as handling mobility (*e.g.,* handover and paging). HSS is the repository that contains the information related to the end users (*e.g.,* authentication keys and UE capabilities). SGW and PGW compose the EPC data plane that is responsible for routing the packets. SGW supports IP data traffic between the Radio Access Network (RAN) and PGW. Furthermore, SGW configures the uplink and downlink tunnels for data transfer. PGW sends the EPC data traffic to external IP networks.

The vEPC concept refers to incorporate NFV into EPC. By vEPC, the mobile operators can dynamically address the rising number of LTE subscriptions, scale their networks to meet the traffic demands during peak hours, instantiate new services based on the network conditions in a near real-time way, and guarantee the availability of the NSs (Buyakar et al., 2017, Choi et al., 2018, Hawilo et al., 2014).

The control plane of an EPC or vEPC includes the (v)SGW, (v)PGW, and (v)MME, and operates as follows (3GPP, 2018a). When a UE requires to connect to an LTE network, it makes an attach request. A radio connection must first be established between the UE and an eNodeB in order to perform the UE attachment process. After the connection, the UE sends an attachment request to MME via the eNodeB. This request includes the International Mobile Subscriber Identity that identifies the UE. The attachment process involves several sub-processes, such as user authentication, security, and session setup. In brief, MME performs the UE authentication by using HSS. HSS updates the UE data and sends a response to MME. Then, the security setup includes encryption to ensure communication between the UE and MME. If the security setup is successful, a default "bearer" is created for the UE by the packet core. During the attachment process, an IP address is given to the UE by PGW, and Tunnel Endpoint Identifier values for the "bearer" are exchanged among the eNodeB, MME, SGW, and PGW. Also, a tunnel is created for the data traffic between the UE and PGW via SGW. Note that if the UE needs to disconnect, it makes a detachment request. When the UE sends this request, its entire state is cleared from all the EPC entities and MME sends the detachment response to the UE via the eNodeB.

2.1.4. 5G Core Network

5GCN is a service-based architecture (3GPP, 2018c, Campos et al., 2021), in which NFs cooperate to perform signaling procedures, such as connection, registration, mobility management, and session management (3GPP, 2018d). The connection management establishes and releases the signaling connection in the control plane. The registration management registers a user with the 5G network, and creates its user context, allowing the use of data services. The mobility management keeps track of the current location of UE, enabling the handover aware to radio conditions, load balancing, or QoS requirements. The session management controls the user plane functionality, such as packet routing and forwarding, packet inspection, and traffic steering. Session establishments enable that subscribers can use applications, such as browsing the web and advanced assisted driving.

Consider the session establishment procedure. Fig. **2-2** presents the service chain of this procedure, including NFs, service interfaces, and roles (*e.g.,* service consumer or service producer) taken by UE, the Access and Mobility Function (AMF), the Session Management Function (SMF), and the Network Repository Function (NRF). UE allows the user to connect to the 5G network and use its applications. AMF provides the communication service using the interface *Namf_Communication*. This service enables an NF to communicate with UE through Non-Access-Stratum (NAS) messages or the access network. *Namf_Communication* defines several operations, including *UEContextTransfer*, which offers the general registration procedure (3GPP, 2019). SMF supports, through its interface *Nsmf_PDUSession*, the establishment, modification, and release of data sessions between the User Plane Function (UPF) and the access network. Also, SMF configures traffic steering policies at UPF, allocates IP address to UE, and applies charging rules. NRF registers and discovers NFs. The discovery functionality maintains uniform resource locators (URLs), profiles, and services of NF instances available for composing service chains. Discovery and registration are offered by interfaces *Nnrf_NFDiscovery* and *Nnrf_NFManagement*, respectively (3GPP, 2018d).

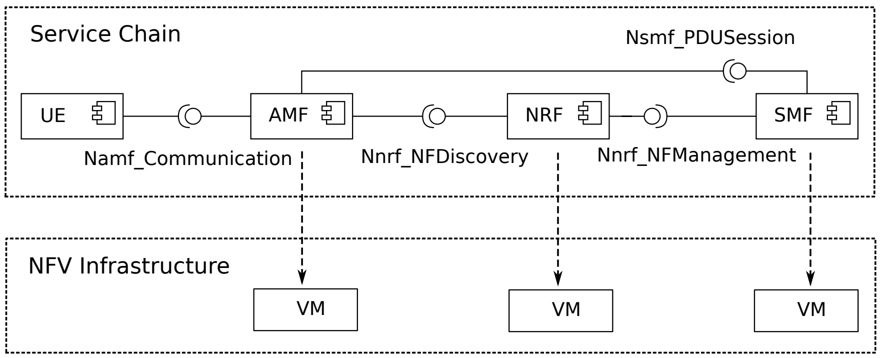


Figure 2-2.: Service chain and NFVI of the session establishment procedure.

Fig. **2-2** also shows NFVI of the session establishment procedure. In this example, NFVI indicates that each NF (AMF, SMF, and NRF) occupies a VM. However, other deployment alternatives are feasible; for instance, each VM can host several NF instances. The number of active NF instances can increase (scaling out) or decrease (scaling in) to face a growing or declining workload, respectively, to improve resource utilization. Moreover, the number of Central Processing Unit (CPU) cores can be dynamically allocated to the VMs to handle workload variations, leading to the scaling up (increasing CPU cores) or the scaling down (decreasing CPU cores). Scaling out/in (horizontal scaling) and scaling up/down (vertical scaling) allow adjusting the slice capacity to the workload dynamics.

Fig. **2-3** describes the behavior of the session establishment procedure by a sequence diagram. The session establishment includes an AMF, which serves as the single-entry point for a UE for all its communication. Once the user decides to use one application, for example, to browse the web, AMF needs to assign an SMF to manage the user session context. As NFs can be instantiated and deleted at any time, AMF needs to discover an available and suitable SMF via the NF Discovery operation performed between AMF and NRF. To accomplish successful discovery, SMF must register beforehand with NRF. It is noteworthy that the pair of messages (request-reply) in the communication between NFs indicates synchronous communication.

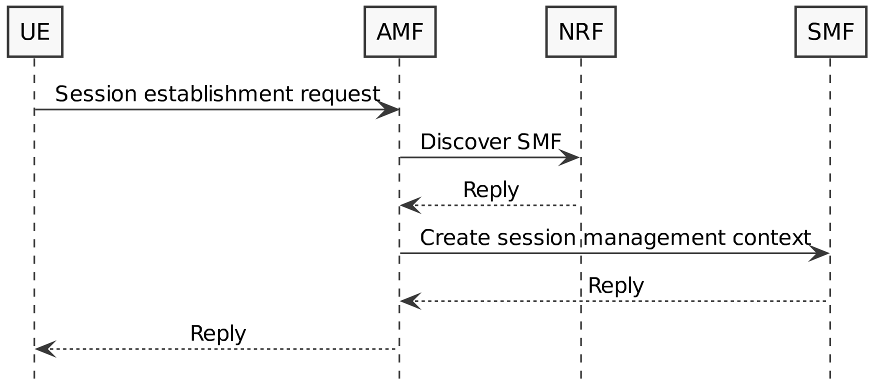

UE
AMF
NRF
SMF
Session establishment request
Discover SMF
Reply
Create session management context
Reply
Reply

Figure 2-3.: Session establishment procedure.

2.1.5. Reinforcement Learning

RL (Sutton and Barto, 2012) is a sub-field of machine learning, where an autonomous agent (*i.e.,* the learner and decision-maker) is able to learn by trial and error. The agent senses the state of its environment (*i.e.*, everything outside the agent and that interacts with it), it can take actions that affect the state depending on of goals relating to the state of the environment, and for each action taken, it receives a reward indicating how good was the action taken. Fig. **2-4** depicts the RL process.

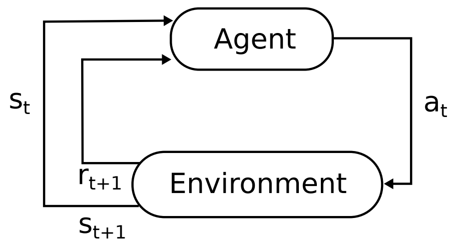


Figure 2-4.: Reinforcement learning process.

RL is formally defined by the Markov Decision Processes (MDP) framework (Mausam and Kolobov, 2012). An MDP has a state set S, an action set A, one-step state transition dynamics $\mathcal{P}^a_{ss'} = P(s_{t+1} = s'|s_t = s, a_t = a)$, which describes the probability of transitioning to state $s' \in S$ after taking $a \in A$ in state s, and the expected value of the next reward $\mathcal{R}^a_{ss'} = E(r_{t+1}|s_t = s, a_t = a, s_{t+1} = s')$ given the previously executed action and resulting state transition. Both state transitions and rewards can be stochastic. The learning goal in an MDP is to find a policy f that maps states to action selection probabilities, maximizing expected reward. When following a fixed policy f, the value (V) of a state s under that policy is defined as the total amount of reward R that the agent expects to accumulate when starting in state s and following f; thereafter:

$$V^f(s) = \mathbb{E}_f(R_t|s_t = s) = \mathbb{E}_f(\textstyle\sum_{k=0}^{\top} \gamma^k r_{t+k+1}|s_t = s)$$

The rewards are discounted by factor $\gamma \in [0,1)$ to ensure a bounded sum in infinite horizon MDPs. An important property in MDPs is the Markov property that establishes the future states only depend upon the present state. This property simplifies the analysis and implementation of RL systems.

Consider an example that illustrates the behavior of such an agent when it operates for routing packets. States can be defined by pairs formed by route and time for delivering a packet; and actions can refer to select a particular route to send packets. Also, let's suppose a routing policy that establishes that a packet must be sent by using the route with the minimum deliver time. In this example, the network is the environment and the agent interacts with it by sending packets and receiving as rewards the reciprocal of deliver times. The agent tries all available routes and receive a reward for each route. Through the experience, the agent learns which is the best route, for instance maximizing the expected value of the sum of rewards. An important feature of RL is that learning is continual and on-line, and so, the agent can adapt its policy to changing conditions of the network.

Q-Learning: Q-Learning is a method of RL (Watkins, 1989). On it, to each pair state-action is assigned an action value, which is the expected utility of taking an action $\mathbf{A}_t$ when the agent is in the state $\mathbf{S}_t$ and follows the optimal policy. A policy is a rule for selecting actions. The value function is represented by $Q(\mathbf{S}_t, \mathbf{A}_t)$; and Q, implicitly defines a current policy f, which is

$$f_t(\mathbf{S}_t) = a, \ \textit{such that} \ Q_t \ (\mathbf{S}_t, \mathbf{A}_t) = \underset{\mathbf{A}}{max} Q_t(\mathbf{S}_t, \mathbf{A}_t) \tag{2-1}$$

f_t and Q_t corresponds to the policy and values of Q at time t, respectively. That is, the current policy is always to choose actions with maximal estimated action value.

The agent through its experience adjusts the values of Q according to

$$Q_{t+1}(\mathbf{S}_t, \mathbf{A}_t) = (1-\alpha) Q_t(\mathbf{S}_t, \mathbf{A}_t) + \alpha(R_t + \gamma \underset{\mathbf{A}}{max} Q_t(\mathbf{S}_{t+1}, \mathbf{A}_{t+1})) \tag{2-2}$$

where R_t denotes the reward received at step t, α is a small positive number called learning factor, and $0 \leq \gamma \leq 1$ is a discount factor. If $\gamma = 0$, the agent tries only to maximize immediate rewards. But in general, $\gamma \neq 0$, the agent takes future rewards into account, and it takes better actions.

Q-Learning is popular because of its simplicity and has been used in different application domains, such as smart cities for intelligent traffic signal control (Kuang et al., 2021) and passengers assistance systems for crowded public transportation (Neelakantam et al., 2020). In e-Health, Q-Learning has been used for doctor's decision-making on the health of patients (Sreedhar and Swathi, 2021) and robotic dialogue strategies for people with dementia (Yuan et al., 2021). In 5G networks, Q-Learning has been used for radio resource adaptation of 5G base stations (Gowtam Peesapati et al., 2021) and QoS-aware load balancing in wireless networks (Rivera and Erol-Kantarci, 2021). Furthermore, Q-Learning can be used for scaling NFs as is proved in (Tang et al., 2015).

2.1.6. Multi-Agent Reinforcement Learning

A multi-agent system (Burguillo, 2018) is a group of autonomous, interacting entities sharing a common environment, which they perceive and upon which they act. The agents can be endowed with previous behaviors, but their most important property is they are enabled to learn new behaviors online, such that the performance of the agent and the whole multi-agent system gradually improves. This is usually useful in an environment that changes over the time. Precisely, NFV has this feature of dynamism. For instance, a composed NS can be modeled as a multi-agent system, which can run an adaptive task to manage traffic changes.

The generalization of MDP to the multi-agent case is the stochastic game. In a stochastic game, each agent has its own set of actions, *i.e.,* for n agents the joint-action space is $A = A^1 \times A^2 \times ... \times A^n$. The state transition and reward functions now depend on the joint action of all agents:

$$\mathcal{R} : S \times A^1 \times ... \times A^n \times S \rightarrow \mathbb{R}^n$$
$$\mathcal{P} : S \times A^1 \times ... \times A^n \times S \rightarrow [0, 1]$$

The rewards can be the same or not for all agents. A special case of stochastic games is the stateless setting described by normal-form games. Normal-form games are one-shot interactions, where all agents simultaneously select an action and receive a reward based on their joint action, after which the game ends. There is no state transition function, and the reward function can be represented by an n-dimensional payoff matrix, for n agents. A policy of an agent is simply a probability distribution over its actions.

Learning in a multi-agent setting is inherently more complex than in the single-agent case (Busoniu et al., 2008), as agents interact both with the environment and potentially with each other. Learning is simultaneous, meaning that changes in the policy of one agent may affect the rewards and hence the optimal policy of others. Moreover, agents may have conflicting interests, yet cooperation with competitors may yield short or long term benefits. This makes it difficult to judge the learning process, since myopic maximization of individual rewards might not lead to the best overall solution. The fact that the reward function depends on the actions of other agents leads to an important characteristic of MARL: the environment is non-stationary and as a result each agent is essentially pursuing a moving target. Moreover, the fact that multiple agents influence the environment means that, from the perspective of the individual agents, the Markov property no longer holds. All these challenges must be addressed in the NFV scaling context.

MARL has been used in different application domains, such as robotics for redundant robot control (Perrusquía et al., 2020) and the dynamic ocean monitoring by a swarm of buoys (Kouzehgar et al., 2020). In medicine, MARL has been used for iteratively-refined interactive 3D medical image segmentation (Liao et al., 2020) and personalized hypertension risk prediction (Abrar et al., 2021). In networking, MARL has been used for task offloading and resource allocation in cybertwin based networks (Hou et al., 2021) and resource allocation for delay-sensitive vehicle-to-multi-edges (V2Es) communications in vehicular networks (Wu et al., 2021).

2.1.7. Gaussian Processes for System Identification

A GP is a collection of random variables which have a joint multivariate Gaussian distribution (Rasmussen, 2004). A GP is a generalization of the Gaussian distribution but over functions and it is fully described by its mean and variance. For an input $\mathbf{x}$ and output y, with a relationship $y = f(\mathbf{x})$, $y^1, ..., y^n \sim \mathcal{N}(0, \Sigma)$, where $\Sigma_{pq} = Cov(y_p, y_q) = C(\mathbf{x}_p, \mathbf{x}_q)$ gives the covariance between output points corresponding to input points $\mathbf{x}_p$ and $\mathbf{x}_q$. The mean $\mu(\mathbf{x})$ usually is assumed to be zero, and a common choice for covariance function is

$$C(\mathbf{x}_p, \mathbf{x}_q) = v \exp\left[-\frac{1}{2}\sum_{d=1}^{D} w_d (x_p^d - x_q^d)^2\right] \tag{2-3}$$

where D is the input dimension and $\Theta = [w_1, ..., w_D, v]^T$ are the hyperparameters, they define the probability distribution of the functions and do not parameterize the function itself.

The system identification is done giving a set of N D-dimensional input vectors $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]$ and a vector of output data $\mathbf{y} = [y^1, ..., y^N]^T$, and tunning the hiperparameters of the covariance function, that is done by maximizing the log-likelihood of the hiperparameters. Based on the data $(\mathbf{X}, \mathbf{y})$, and given a new input vector $\mathbf{x}^*$, the GP find the predictive distribution of the corresponding output y^*.

GPs have been used in different application domains, such as robotics for survivable robotic control (Raza et al., 2021) and soft robots with shape memory actuators (Sabelhaus and Majidi, 2021). In medicine, GPs have been used for outlier detection in image segmentation (Popescu et al., 2021) and qualification of drug dosing regimens in pediatrics (Siivola et al., 2021). In telecommunication systems, GPs have been used for efficient orchestration of virtualization resource in radio access network (Zou et al., 2021) and optimizing the design of a multiband microstrip antenna for mobile terminals (Zhang et al., 2020).

2.1.8. Performance Evaluation Process Algebra

PEPA is a language for the analysis of concurrent systems (Hillston, 1994), such as the control plane of 5GCN. PEPA offers formality, abstraction, and compositionality. Formality gives a precise meaning to all terms in the language. Abstraction allows building up complex models from components but disregarding their internal behavior; in this thesis, NFs are modeled as PEPA components. Compositionality allows modeling the interaction between components by cooperation processes. A PEPA model is a composition of entities or components intended to perform actions sequentially. Actions can involve one (independent) or several components (composite) (Tribastone, 2013). PEPA defines the following operators.

Prefix $(\alpha, r).P$ is the basic form to construct the behavior of components, such as AMF and SMF. $(\alpha, r).P$ carries out activity (α, r) that has a type α and a duration exponentially distributed with mean $1/r$ time units. $(\alpha, r).P$ subsequently behaves as P. A prefix allows capturing the behavior that involves precedence between two distinct activities. For example, SMF creates a session context to manage the use of applications by subscribers. Subsequently, SMF sends back a reply. The complete model of this sequential component is $(createSession, r_{cs}).Reply_{smf}$.

Choice $P + Q$ represents a system which may behave either as P or Q. Consider 5G multimedia services that support emergency sessions. To provide such services, AMF may contain information of an SMF configured statically, being unnecessary to request the discovery operation to NRF. In this way, AMF can be modeled as $(call_{nrf}, p \cdot r_{call}).Disc + (call_{smf}, (1 - p) \cdot r_{call}).SC$. The choice operator enables actions $call_{nrf}$ and $call_{smf}$, which are executed with probabilities p and $(1 - p)$, respectively. Once UE sends a session creation request, AMF may perform the discovery of an available SMF by using NRF or the session creation directly using an SMF for an emergency.

Constant $A \overset{def}{=} P$ models the cyclic behavior of an NF. Once an NF completes the operations requested by a client, the NF returns to the initial state to serve a new client. For example, the behavior of the processing of the VM hosting SMF (SMFP) can be modeled by using a constant. SMFP gets access to the processor by the action get_{smfp}, performs the action *createSession*, and returns to the initial state. As these actions are sequential and cyclic, SMFP can be modeled by $SMFP \overset{def}{=} (get_{smfp}, r_p).(createSession, r_{cs}).SMFP$.

Cooperation $P \underset{L}{\bowtie} Q$ models the interactions between components by the synchronization of P and Q over the action types in the set L. All the other actions are performed independently. Consider that AMF cooperates with SMF for creating a user session context. This cooperation can be modeled by $(cs, r_{cs1}).(update, r_u).AMF \underset{\{cs\}}{\bowtie} (cs, r_{cs2}).(rep, r_r).SMF$. In this cooperation, the two components perform the shared action cs (create session), subsequently behaving as $(update, r_u).AMF \underset{\{cs\}}{\bowtie} (rep, r_r).SMF$. Then, actions *update* (update state) and *rep* (generate reply) are carried out independently. The cooperation operator defines the overall system model and, so, it is also known as the system equation.

The cooperation operator is fundamental to analyze scalability, since it allows to specify a pool of threads working in parallel in a processor, as well as multiple NF instances for each NF type. In this thesis, the number of threads per processor and the number of NF instances are noted as N_t and N_{nf}, respectively. For scalability analysis, consider P and Q as the components that model two multi-thread, multi-instance NFs: NF_1 and NF_2, respectively. The cooperation between P and Q is modeled as $P[N_{nf_1} \cdot N_{nfp} \cdot N_t] \underset{L}{\bowtie} Q[N_{nf_2} \cdot N_{nfp} \cdot N_t]$. The product $N_{nf} \cdot N_{nfp} \cdot N_t$ is the total number of processors allocated to an NF instance. On the other hand, the product $N_{nf} \cdot N_{nfp}$ is the total number of processors allocated to an NF. Consider the outlined example again, the cooperation between components AMF and SMF may be defined as $AMF[N_{amf} \cdot N_{nfp} \cdot N_t] \underset{L}{\bowtie} SMF[N_{smf} \cdot N_{nfp} \cdot N_t]$, where N_{amf} and N_{smf} are the number of instances of AMF and SMF, respectively; N_{nfp} is the number of processor allocated to each NF instance of AMF and SMF, and N_t is the number of threads that each processor can handle.

PEPA has been used in different domains to model, evaluate, and verify the performance of systems, such as intelligent transportation for automatic generation of test cases for autonomous vehicle (Chen, 2020) and a decentralized trust management system (Chen et al., 2020). In cloud computing, PEPA has been used for modeling hybrid cloud-fog systems (Chen et al., 2021) and predictive runtime modelling of kubernetes microservices (Burroughs, 2021).

2.2. Related Work

This section presents the related work to this thesis, which was developed applying a systematic review approach (Kitchenham, 2004). The research included four stages: *i)* define a review protocol; *ii)* identification of inclusion and exclusion criteria; *iii)* search relevant publications; and *iv)* data extraction and synthesis. The research questions were:

- What studies have been reported regarding network performance assessment in NFV?
- What approaches have been reported regarding scaling mechanisms in NFV?
- What scaling solutions are regarding NFs in isolation?
- What scaling solutions are regarding composed NSs?

2.2.1. Telco Cloud Scaling

In the NFV literature, some works have investigated issues related to performance assessment. For instance, the work (Cao et al., 2015) demonstrated the importance of network performance characterization in NFV and they proposed a general framework for NSC characterization. The work (Rajan et al., 2015) uncovered performance bottlenecks in a software-based LTE core network architecture. The work (Prados-Garzon et al., 2017b) proposed a theoretical framework to evaluate the performance of an LTE virtualized MME (vMME) hosted in a data center. The aforementioned works focus on the characterization of network performance but they do not propose any automatic solution for managing its changes.

Other works have proposed scaling mechanisms for performing scaling by using techniques such as threshold rules, time series, and the Q-Learning method. The work (Carella et al., 2016) used the technique of static threshold-based rules. Thresholds separate three performance regions: poor, good and oversized. If performance is in the good region no action is taken; if performance crosses from good to poor, scaling up/out starts. On the other hand, if performance crosses from good to oversized scaling down/in starts. In particular, the work (Carella et al., 2016) proposes a scaling procedure starts when a performance measurement, such as the requests numbers and CPU idle time, crosses predefined levels. These thresholds indicate that either a service quality is no longer acceptable or the capacity can be reduced without affecting the service quality. In threshold-based scaling, performance could exceed a target level at times that thresholds are crossed, which may incur in violations of expected quality. Also, if performance crosses consecutively thresholds, transitory scaling oscillations may happen. These transitory changes occur because scaling modifies the performance levels. Oscillations can lead to a significant problem because each scaling decision triggers procedures for provisioning or releasing resources.

To overcome the possible problems of oscillations in static threshold-based approaches, the work (Liu et al., 2018) proposed a reactive auto-scaling mechanism that aims at building dynamic thresholds. This mechanism is based on fuzzy logic that allows adjusting thresholds based on dynamic workloads and cluster size for a web application. Performance evaluation was conducted with real-life Wikipedia traces in the Amazon Web Services cloud platform. Experimental results demonstrated that the fuzzy auto-scaler reduces cloud resources usage (*i.e.,* number of VMs) and minimizes Service Level Agreement (SLA) violations in terms of keeping the average response time under 200ms. As mentioned by the authors, the creation of an optimal auto-scaler is far from trivial and other techniques to determine the best candidate VMs for the scaling-in need to be explored. Scaling-in is riskier than scaling-out, as over-provisioning only costs money, but under-provisioning means losing customers.

The work (Bilal et al., 2016a) proposed two scaling mechanisms that use time series models for predicting CPU usage. The first one predicts the day-ahead usage, and so, schedules required resources, which is useful for known cases, such as new year holidays or live events. The second mechanism operates online and is used to predict the usage during short ranges of time when unexpected events occur. The drawback of this work is that it does not include an evolutionary strategy that would allow adapting the models to changes in traffic patterns.

The work (Tang et al., 2015) presented an algorithm based on the Q-Learning method that defines states, scaling actions and rewards. States refer to workloads of resources and services, such as usage of CPU and number of bearers requested by users; and rewards weigh performance and resource use. The drawback of this work is that the proposed algorithm performs exploration steps on the network directly, so a non-optimal scaling decision can be made until the algorithm has converged.

The previous scaling solutions may present inaccuracies in the decision-making, particularly in dynamic environments, which is a significant problem because each scaling decision triggers procedures for provisioning or releasing resources. Reactive solutions that use static threshold-based rules, such as (Carella et al., 2016), can lead to alternating changes between states because the scaling decision is taken considering only the current values of workload or performance, which can cross several times the thresholds. In reactive solutions that use dynamic thresholds, such as (Liu et al., 2018), determining the best candidate VMs for the scaling-in may be riskier than scaling-out because possible under-provisioning would mean losing customers.

In solutions using predictive models, such as (Bilal et al., 2016a), an evolutionary strategy would be desirable, which would allow adapting the models to changes in traffic patterns. Solutions based on RL, such as (Tang et al., 2015), learn by interacting with the environment, and they are adaptive. However, up to complete the learning, mistake decisions can be taken. In addition to the above limitations, the previously cited works focus mainly on the scaling of individual NFs, but they do not consider the issues related to compound services that are associated with how to scale different NF types in an NS as a whole.

Related to compound NSs, the work (Wang et al., 2016) presents online algorithms to determine the optimal numbers of NF instances and their optimal placement on servers. The optimization objective is to minimize the operational cost, attributed to the power consumption of the hosting server, and the deployment cost, related to the process of transferring a VM image containing the NF, booting it and attaching it to devices on the server. The drawback of this work is that it assumes all NFs in an NS must be scaled. However, a deeper performance characterization should be conducted to analyze the scalability.

The work (Zhang et al., 2016) focus on the migration of flows, which is the process that follows the decision-making; particularly for stateful NFs. In this work, a heuristic algorithm selects a subset of existing flows to migrate them to the new instances. Numerical experiments and validation results revealed that the proposed algorithm could significantly decrease the number of packets affected by buffering and minimize the latency introduced by flow migration. The drawback of this work is that it assumes all NFs must be scaled without their scalability analysis.

The work (Rankothge et al., 2017) presents a scaling algorithm based on genetic programming. A comparison with Integer Linear Programming (ILP), traditionally used to optimize the allocation of virtual machines, revealed that this genetic-based scaling spends only a few milliseconds for competitive solutions while ILP solution takes hours. The drawback of this work is that only one NF is chosen for scaling randomly, missing the performance analysis of each NF.

The works for compound NSs show that the scaling problem is not addressed enough yet. (Wang et al., 2016) assumes all NFs of an NS must be scaled and introduces a factor that models the change in the traffic rate for each NF. (Zhang et al., 2016) focuses only on the flow migration and assumes all NFs must be scaled; and in (Rankothge et al., 2017), one NF is chosen randomly, which is increased/decreased in one instance. However, the NFs can be dependent each other and the performance of the NS is not stationary, *i.e.* scaling one NF changes the state space, which leads to the whole performance cannot be maintained in a target range.

Table **2-1** summarizes the most relevant related work to Telco cloud scaling regarding performance assessment, support for the two scaling methods (*e.g.,* scaling out/in and up/down), and the technique used for implementing the scaling mechanism. From this summary, it can be noted that only one work considers the two issues, composed services, and support for the two scaling methods. However, this work has the restriction described above, in which one NF is chosen randomly, which is increased/decreased in one instance. Scaling compound services requires considering multiple configurations, which allows selecting the most appropriate and thus achieve greater elasticity. Chapter 5 proposes a mechanism that considers the joint scalability of NFs in an NS allowing to choose the best configuration according to variations in the number of users.

2.2.2. Evolved Packet Core Scaling

In the NFV research, several works have proposed the use of vertical and horizontal scaling methods in EPC. Most of these works have used horizontal scaling since it provides high availability and performance because of the distribution of the workload in multiple instances of the EPC entities. In turn, vertical scaling represents an excellent cost-benefit ratio for

Table 2-1.: Related work to Telco cloud scaling.

Work	**Performance Assessment**	**Scaling Method**		**Used Technique**
		Horizontal	**Vertical**	
Cao et al. (2015)	✓			
Rajan et al. (2015)	✓			
Prados-Garzon et al. (2017b)	✓			
Carella et al. (2016)		✓	✓	Threshold rules
Liu et al. (2018)		✓		Fuzzy logic
Bilal et al. (2016a)		✓		Time series
Tang et al. (2015)		✓		Q-Learning
Wang et al. (2016)		✓		Optimization
Zhang et al. (2016)		✓		Heuristics
Rankothge et al. (2017)		✓		Genetic programming

mobile operators since the increase of resources per EPC entity facilitates the network management. However, scaling based on the two methods (horizontal and vertical) has not been deeply analyzed in a vEPC.

Some investigations have proposed an MME architecture to support horizontal scaling based on three components: a frontend, a set of workers, and a database. The front-end is a proxy that provides interfaces to other EPC entities. One or more workers (*i.e.,* instances of MME) process the control traffic in a stateless way. The database stores each worker state. Thus, the front-end and the database are transparent to other entities (*e.g.,* SGW and PGW), while the workers scale horizontally to face the workload variations (Prados-Garzon et al., 2017a, Premsankar et al., 2015).

By using a similar workers-based architecture, the EPC entities (MME, SGW, and PGW) can be deployed as clusters of replicas that share the incoming workload. Each EPC entity is composed of replicas (*i.e.,* workers), a load balancer (*i.e.,* front-end) that distributes the incoming workload among the replicas by using a round-robin policy, and a shared database that stores the replicas state with several synchronization options (*i.e.,* no sync, session sync, and always sync). The corresponding performance evaluation reveals that the sync options outperform the no sync option in relation to latency (approx. 71%) and throughput (approx. 75%) (Satapathy et al., 2017).

SCALE is a framework for performing horizontal scaling of MME by re-designing the MME functionality into two parts: a load balancer and an MME processing cluster. Furthermore, the authors use consistent hashing on the access patterns of the available devices registered in MME to reduce memory usage intelligently. Thus, the reuse of sessions created by SCALE allows allocating fewer resources to process more link creation requests for data and control traffic (Banerjee et al., 2015).

Cloud Native Solution for MME (CNS-MME) is a proposal that uses a microservices architecture for scaling MME horizontally. This architecture deploys the CNS-MME as a virtualized cluster of microservices. In this cluster, a load balancer separates the control processes (*i.e.,* attach and detach) and delivers them separately to groups of VNFs intended for each process. The other entities (*i.e.,* HSS, SGW, and PGW) are also deployed as VNFs by using containers technology. CNS-MME is highly available and supports automatic scaling of the microservice required for load balancing. The authors determine that CNS-MME outperforms (approx. 7%) an MME based on a monolithic architecture. They also conclude that CNS-MME reduces the consumption of processing resources (approx. 26%) (Amogh et al., 2018).

The work (Ren et al., 2016) proposed an algorithm for scaling vEPC horizontally in order to address the trade-off between the performance and the operational costs. In particular, they use analytic and simulation models to study the response time and the operational costs of an NFV-based EPC. In the models, the legacy network equipment is considered as a reserved block of servers, and the instances of VNFs are powered on and off according to the number of tasks requests at a particular time.

The work (Taleb et al., 2015) elaborated architectural models and discussed the feasibility of offering EPC as a service in a cloud infrastructure by using 1:1 and 1:N mapping. In the first mapping option, each EPC entity is deployed as an NF running in a VM that enables vertical scaling. In the second option, each EPC entity is decomposed into multiple elements: a front-end, a set of workers, and a database. This decomposition is utilized to perform horizontal scaling. It is important to highlight that these architectural models are merely conceptual; they were neither implemented nor tested by their authors. Chapter 3 leverages some of these models and goes beyond by proposing a threshold-based and straightforward algorithm that turns them operable.

Table **2-2** summarizes the most relevant related work to EPC scaling. This table reveals several facts. First, the solutions (Premsankar et al., 2015, Prados-Garzon et al., 2017a, Banerjee et al., 2015, Amogh et al., 2018) focus on scaling MME only. Second, all works apply horizontal scaling rather than vertical scaling. Third, horizontal and vertical scaling is not provided for EPC in the related works. An EPC is making up of interrelated entities, and its scaling must be analyzed in detail. None of the research works have analyzed and incorporated scaling in vEPC by moving between the horizontal or vertical scaling. Chapter 3 proposes a scaling mechanism that depending on workload variations goes from horizontal scaling to vertical one or vice-versa. Also, Chapter 3 carries out the performance analysis of individual vEPC entities to determine which one requires to be scaled (finer scalability) for improving the vEPC performance as a whole.

Table 2-2.: Related work to EPC scaling.

Work	Involved Functions	Scaling Method	
		Horizontal	Vertical
Premsankar et al. (2015)	MME	✓	
Prados-Garzon et al. (2017a)	MME	✓	
Banerjee et al. (2015)	MME	✓	
Amogh et al. (2018)	MME	✓	
Hahn and Gajic (2015)	SGW, PGW	✓	
Satapathy et al. (2017)	MME, SGW, PGW	✓	
Ren et al. (2016)	MME, SGW, PGW	✓	
Prados-Garzon et al. (2018)	MME, SGW, PGW	✓	
Taleb et al. (2015)	MME, SGW, PGW	✓	✓

2.2.3. 5G Core Network Scaling

In NFV literature, some works propose scaling mechanisms to manage traffic variations in 5GCN. The work (Alawe et al., 2018b) compared two machine learning techniques for scaling the AMF of 5G horizontally; AMF has similar behavior to MME. In particular, a Recurrent Neural Network (Zaremba et al., 2014) and a Deep Neural Network (Canziani et al., 2016) are utilized to anticipate the scaling process of AMF by forecasting the arrival of session requests in 5G. Simulation results show that the forecast-based scalability outperforms the threshold-based solutions concerning the adaptability to traffic changes. In contrast to this work, which scales AMF only. This thesis considers the joint scaling by the cooperation of other NFs, such as SMF and NRF.

The work (Dutta et al., 2016) proposed a mechanism based on threshold values to scale a cloud-based 5G mobile system elastically. In this mechanism, a decision-making module defines when to trigger the vertical/horizontal scaling as a function of the Mean Opinion Score (Streijl et al., 2016) and the usage of RAM and CPU. As output, this mechanism indicates the resources to be allocated per instance and prevents the disruption of the NSs. This work discusses the applicability of the proposed approach within the NFV MANO framework for a cloud-based 5G core, but details of NFs in the 5G core, such as AMF, SMF, and NRF, are not involved. In contrast, this thesis analyzes the scalability of the 5G core, considering the characteristics of those NFs.

The work (Trivisonno et al., 2018) proposed an approach that aims at serving mIoT (Atzori et al., 2010) connections efficiently. This approach allows modeling and evaluating an end-to-end IoT 5G Network slice (5GNSL) for the device registration and core network bearer setup, which requires to connect diverse NFs, such as AMF, SMF, and the Authentication Server Function (AUSF). The authors by simulations measured the control plane signaling and the user plane traffic generated by devices accessing randomly, but they did not analyze the scalability of the IoT 5GNSL modeled. In contrast to this work, this thesis analyses the scalability of two NSs in the 5G core: session establishment and user registration in a Vehicle-to-Everything (V2X) 5GNSL.

The work (Campolo et al., 2018) presented a network slicing architecture to support V2X applications (Chen et al., 2017). This architecture introduces a service layer in which 5GCN comprises several NFs, such as AMF, AUSF, and Unified Data Modeling (UDM). The core network includes multiple AMF instances to manage the high signaling load generated by devices-mobility and avoid the increase in latency during slice registration procedures. The authors evaluated the user plane of a V2X 5GNSL by using the Mininet emulator (Fontes et al., 2015) in which they measured latency, throughput, and packet drops. In contrast to this slicing architecture, this thesis does not focus on the scaling and performance evaluation of the user plane, but in the control plane. In particular, this thesis considers metrics such as throughput, processor utilization, and average response time in the registration procedure.

The work (Rotter and Van Do, 2021) proposed a queuing model for a threshold-based algorithm that controls the number of 5GCN UPF instances in response to traffic load changes, saving resource consumption, and keeping high utilization of requested resources. Performance measures calculated by the model are blocking probability of sessions, the average number of UPF instances, the average number of busy UPF instances, and utilization. 5G base stations and UPF perform tasks of the 5G data plane, providing necessary procedures to convey data flows between end-devices and data networks. In contrast to this work, this thesis models the 5GCN control plane using PEPA, focusing on signaling tasks, such as user registration and session establishment.

The work (Harutyunyan et al., 2021) studied the joint user association, NSC placement, and VNF scaling in 5GCN and Multi-access Edge Computing (MEC). The study classifies NFs into stateful, control plane, and user plane NFs. Also, it uses ILP to analyze the trade-offs between vertical, horizontal, and hybrid scaling. Analysis results demonstrated that vertical scaling is more efficient in utilizing VNF resources. Horizontal scaling provides high availability but inefficiency because of under-utilization. In turn, hybrid scaling exhibits a better compromise between high availability and resource utilization. In contrast to this work, this thesis models the 5GCN control plane as a concurrent system using PEPA and analyzes its scalability.

The work (Qu et al., 2021) studied a dynamic VNF scaling problem in 5GNSLs to guarantee a required delay in presence of real-world time-varying traffic with non-stationary characteristics. A 5GNSL is represented as an NSC, where a source node generates service requests that traverse through several VNFs in sequence towards a destination node. Traffic arrivals are modeled as non-stationary traffic time series with different timescales, which are used to learn how the traffic changes and predict the resource demand. At traffic inflection points, decisions of VNF scaling and migration are made using a Q-Learning algorithm. In contrast to this work, this thesis models the 5GCN control plane as a concurrent system using PEPA and analyzes its scalability.

Table **2-3** summarizes the most relevant related work to 5GCN scaling. This table reveals several facts. First, works (Alawe et al., 2018b, Dutta et al., 2016) conducted scalability analysis for the 5GCN control plane, (Rotter and Van Do, 2021) proposed a scaling mechanism for the 5GCN user plane, and (Harutyunyan et al., 2021) analyzed the optimal scaling in 5GCN and MEC. However, they do not focus on 5G network slicing. Second, works (Trivisonno et al., 2018, Campolo et al., 2018) focused their studies on 5GNSLs (*e.g.,* IoT and V2X 5GNSL), but they have not conducted scalability analysis. Furthermore, the work (Campolo et al., 2018) does not evaluate the 5GCN control plane. Third, the work (Qu et al., 2021) studied the scalability in the 5GNSL user plane. In contrast to the related work given in Table **2-3**, Chapter 6 carries out a detailed scalability analysis in 5G network slicing for the control plane using the PEPA formalism. Also, Chapter 5 proposes a cooperative scaling mechanism for the 5GCN control plane based on MARL.

Table 2-3.: Related work to 5GCN.

Work	5G network slicing	Scalability Analysis	Functional plane
Alawe et al. (2018b)	-	Multiplicity of AMF	Control plane
Dutta et al. (2016)	-	General NFs in 5GCN	Control plane
Trivisonno et al. (2018)	IoT 5GNSL	-	Control plane
Campolo et al. (2018)	V2X 5GNSL	-	User plane
Rotter and Van Do (2021)	-	UPF	User plane
Harutyunyan et al. (2021)	-	General NFs in 5GCN and MEC	User and control planes
Qu et al. (2021)	General 5GNSLs	General NFs in 5GCN	User plane

2.2.4. Modeling Formalisms for Scalability and Performance Analysis

In NFV literature, some works analyze performance and scalability using mathematical formalisms. The work (Schneider et al., 2019) introduced an approach that uses Queuing Petri Nets (QPNs) for specifying and analyzing virtualized NSs. In particular, the authors analyze the end-to-end delay, throughput, and queue lengths of a video streaming service. NSs following the proposed specification technique can be interpreted as scalable templates, where the behavior of each NF instance is formally specified using QPNs. The authors state that scaling-out new instances, their approach can specify how traffic is balanced between multiple instances of the same NF or whether incoming traffic from multiple instances is synchronized. However, this approach does not show a scalability analysis of NSs.

The work (Prados-Garzon et al., 2017b) proposed a solution that uses Queuing Networks to modeling and evaluating the performance of an LTE vMME hosted in a data center. In particular, this solution allows computing the number of vMME processing instances to provide a target system delay given the number of users in the system. Although the proposed solution allows evaluating the scalability of vMME, the current 5GCN architecture defines other NFs, such as AUSF and NRF, which should be involved in the analysis (3GPP, 2018c). Moreover, this solution omits essential performance indices, such as throughput and processor utilization.

The work (Di Mauro et al., 2021) used Queuing Networks for assessing a softwarized IP Multimedia Subsystem (IMS), evaluating its performance and availability. For performance analysis, an optimization problem of resource allocation is formalized by modeling each IMS NF as an M/G/c system. The availability assessment is based on Stochastic Reward Nets (Ciardo et al., 1993), allowing to characterize the behavior of IMS with events of failure and repair. Availability models are used to derive a set of optimal configurations satisfying a given availability requirement while minimizing deployment costs. Although this assessment is for IMS, its methodology can be adapted to other networks. This thesis uses Layered Queuing Networks (LQN), a Queuing Network model, for comparing the accuracy of PEPA-based 5GNSL models.

PEPA has been used to model and evaluate distributed systems. The work (Almutairi and Thomas, 2020) modeled a web-based sales system in the presence of denial of service attacks. Evaluation results regarding throughput and population level show how the attacks negatively impact the orders of customers. The work (Sanders et al., 2020) proposed the Imperial PEPA Compiler, an alternative to the PEPA Eclipse plugin (Tribastone et al., 2009), for robustness analysis of resource allocation. The proposed tool overcomes some limitations, such as the size and complexity of models imposed by the plugin. The work (Hillston et al., 2011) introduced an approach that uses continuous PEPA models to represent large systems with multiple replications in components, such as clients, servers, and devices. From this approach, Chapter 6 leverages replication to represent NF instances and processors that host them, the continuous approximation for efficient analysis of large systems, and the modeling patterns for processor computation and synchronous communication.

The work Tribastone (2010) presented an approach that maps an LQN model to PEPA; LQN is a traditional model for describing (software and hardware) systems with layers and resource contention. This approach uses PEPA to model the semantics of layered multi-class servers, resource contention, the multiplicity of threads, and processors. The authors validated the mapping approach through simulations regarding the accuracy in the translation of LQN to PEPA. The validation allowed concluding that PEPA-based methods are useful in the LQN context to build up efficient models with large component replications. From this approach, Chapter 6 adopts two PEPA structures, namely fork/join synchronization, and the accuracy-based validation.

Table **2-4** summarizes the most relevant related work to modeling formalisms for scalability analysis, revealing several facts. First, 5G Network Slicing lacks a method to analyze the scalability of slices in the core network regarding different configurations of NFs. Second, in 5GCN, scalability and performance assessment must use modeling formalisms to achieve efficient analysis and compositionality. In this sense, PEPA allows stochastic simulation and approximation techniques to analyze models efficiently with a large number of instances, threads, and processors related to NFs. Also, PEPA provides the composition principle that facilitates the modeling of systems with many orchestration alternatives as the 5GCN offers. Third, a scalability analysis needs to consider the multiplicity, threads, and processors of NF instances. Chapter 6 introduces a method based on PEPA for modeling, evaluating, and analyzing the scalability and performance of 5GNSLs in the core network systematically. That chapter illustrates how to use the proposed method by two case studies: the session establishment 5GNSL and the user registration in the Vehicle-to-Everything 5GNSL.

Table 2-4.: Related work to modeling formalisms.

Work	Modeling Formalism	Application Scenario	Scalability Analysis
Schneider et al. (2019)	Queuing Petri Nets	Video streaming	-
Prados-Garzon et al. (2017b)	Queuing Networks	LTE EPC	Multiplicity of vMME
Di Mauro et al. (2021)	Queuing Networks	5G IMS	Multiplicity of IMS NFs
Hillston et al. (2011)	PEPA	Web application	Multiplicity of threads
Almutairi and Thomas (2020)	PEPA	Web-based sales system	-
(Sanders et al., 2020)	PEPA	Distributed computing systems	-
Tribastone (2010)	PEPA	Distributed application	-

2.3. Final Remarks

This chapter presented, initially, the main concepts addressed in this thesis, such as network performance and scalability, NFV, EPC, 5GCN, RL, MARL, GPs, and PEPA. The second part of this chapter offered the main related work, which has been classified into Telco cloud scaling, EPC scaling, 5GCN scaling, and modeling formalisms for scalability and performance analysis.

Unlike the related work presented in this chapter, first, this thesis proposes a scaling mechanism for an NFV-based EPC that incorporates horizontal and vertical scaling (see Chapter 3), which is implemented in a data center. Second, this thesis argues that scaling decision-making should be adaptive and highly accurate to avoid performance violations, transitory oscillations, and wrong decisions. Chapter 4 addresses this problem by combining the Q-Learning method with GPs-based system models. Using the system models, an RL agent can iteratively improve its scaling policy, and therefore, take more accurate scaling decisions. Third, Chapter 5 proposes a cooperative scaling for the 5GCN control plane based in MARL, which overcomes the scaling of independent agents. Fourth, this thesis argues that in 5GCN, scalability and performance assessment must use modeling formalisms to achieve efficient analysis and compositionality. From this approach, Chapter 6 adopts PEPA structures to analyze the performance and scalability in 5G network slicing.

3. A Scaling Mechanism for an Evolved Packet Core based on NFV

Nowadays, mobile operators face challenges like the growing number of users and the increasing demand for current and new applications (Soliman and Song, 2017). In this sense, it is noteworthy that, first, according to Cisco projections, by 2021, there will be around 5,500 millions of mobile phones in the world and mobile traffic will increase eleven times between 2016 and 2021 (Cisco, 2017). Second, the applications that will generate such traffic will be diverse (*e.g.,* high-definition video, eMBB, and mIoT) and, further, they will have different requirements of QoS (Bilal et al., 2016a).

To deal with the challenges aforementioned, telco operators need to evolve their network infrastructure (access and core) seeking to increase capacity, support different types of traffic, ensure different levels of QoS, and accelerate Time-To-Market (Soliman and Song, 2017). The access and core of 4G refer to the Evolved Packet System (EPS). In particular, the core corresponds to the EPC that is responsible for handling the control and data traffic of mobile networks. EPC (3GPP, 2018a) contains different entities such as MME, HSS, SGW, and PGW. The traditional strategy to scale EPC has been replacing these entities with ones with higher capacity, which leads to their oversizing. Since this strategy increases both the Capital Expenditures (CAPEX) and the Operational Expenditures (OPEX), the operators are always looking for technological alternatives to scale EPC by using more flexible and efficient architectures and algorithms (Sanjana, 2016).

An alternative to evolve EPC is NFV. In NFV, NFs are implemented in software and then deployed as virtualized instances in hardware commodity (ETSI, 2014a, Hawilo et al., 2014). A fundamental benefit of NFV is to provide scalability, that is, the ability of a network to be either expanded or contracted without requiring significant architectural changes (Becker et al., 2017). Incorporating the NFV-based scaling to EPC can improve its performance regarding, for instance, registrations per second and latency (Han et al., 2015, Sadiku and Musa, 2013).

In the literature, research works have incorporated horizontal (*i.e.,* add or remove instances) or vertical scaling (*i.e.,* increase or decrease the resources assigned to instances) in the NFs of EPC to improve its performance. In particular, some works have scaled horizontally only the MME (Premsankar et al., 2015, Prados-Garzon et al., 2017a, Banerjee et al., 2015, Amogh et al., 2018). The work (Hahn and Gajic, 2015) scaled both the SGW and PGW horizontally. The work (Satapathy et al., 2017) scaled each EPC entity (*i.e.,* MME, SGW, and PGW) individual and horizontally. The work (Ren et al., 2016) proposed an algorithm for scaling EPC horizontally but did not take in account the joint operation of MME, SGW, and PGW. The work (Taleb et al., 2015) elaborated architectural models and discussed the feasibility of offering scalability in EPC. The work (Alawe et al., 2018b) scaled horizontally the 5G AMF that behaves as the MME of EPC. The work (Dutta et al., 2016) proposed a scheme to scale horizontally and vertically cloud resources of 5G applications but not EPC. The aforecited works present the following gaps. First, most of them focus on scaling EPC horizontally. Second, they do not propose a scaling mechanism for EPC that exploits vertical and horizontal scaling to cope with workload variations. Third, they do not analyze the performance of individual EPC entities deeply to determine which entity must be scaled to improve the EPC performance.

To fill the gaps above mentioned, this chapter proposes a scaling mechanism that exploits horizontal and vertical scaling for handling workload variations and improving performance in EPC. This mechanism includes three modules (data collection, scaling decision, and scaling execution) and an algorithm that defines its operation. This algorithm is threshold-based, straightforward, and implementable in real scenarios. Also, a prototype of the proposed mechanism is developed and deployed in a real public cloud. In such deployment, an evaluation is conducted, regarding registrations per second, latency, CPU, and RAM, and considering a varying workload. The scalability evaluation results reveal that with the mechanism, the registrations per second increase and the latency decreases regarding an EPC without scaling.

The key contributions presented in this chapter are:

- A scaling mechanism characterized by being threshold-based, straightforward, and implementable in real LTE EPC scenarios. This mechanism moves between horizontal and vertical (or vice-versa) scaling to handle workload variations (related to registrations per second mainly) and improve EPC performance.

- A scaling mechanism prototype deployed and tested in the public cloud offered by Amazon Web Services (AWS). This prototype, its detailed explanation, and the corresponding instructions to its deploying and testing are available in Maca et al. (2019).

- An extensive scalability evaluation. The evaluation results reveal that the mechanism increases the registrations per second about 308% and decreases the corresponding latency approximately 70% regarding an EPC without scaling. The mechanism achieves these results by keeping the CPU usage lower than 90% and the used capacity of registrations per second between 65% and 90%, which corroborates the importance of used both horizontal and vertical scaling to improve EPC performance, handle workload variations, and save resources.

The rest of this chapter is organized as follows. Section 3.1 introduces the scaling mechanism. Section 3.2 presents the methodology and the results of the performance evaluation. Section 3.3 highlights conclusions.

3.1. Scaling Mechanism

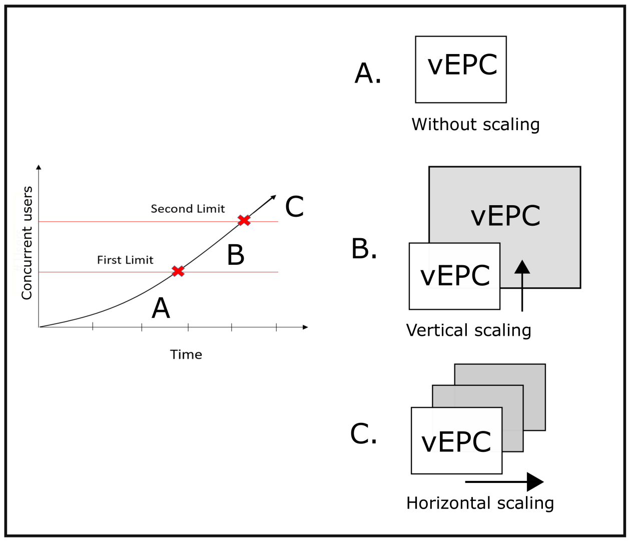
This section, first, exposes a motivating scenario. Second, this section presents the scaling mechanism formed by three modules (*i.e.,* Data Collection, Scaling Decision, and Scaling Execution) and an algorithm that defines its operation.

3.1.1. Motivating Scenario

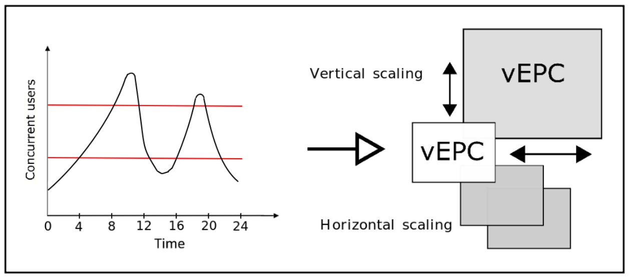
Before introducing the scaling mechanism, consider a vEPC scenario. Fig. **3-1** presents a vEPC with specific resources in memory, number of cores, and storage for managing the control traffic. When the number of concurrent users of the vEPC increases, the set of static resources allocated to the vEPC cannot support the workload. As a consequence, the QoS provided to end-users degrades. When the number of concurrent users decreases, the vEPC is over-provisioned and, thus, it wastes resources.

Fig. **3-1**a illustrates a scenario in which the number of concurrent users of vEPC increases and, therefore, the amount of control traffic also does. To face the increasing traffic, network operators can perform different actions. First, using vEPC without scaling when the number of concurrent users is below the first limit (point A). Second, applying vertical scaling to vEPC when the number of concurrent users exceeds the first limit but it is below the second one (point B). Third, incorporating horizontal scaling to vEPC when the number of concurrent users passes the second limit (point C).

Fig. **3-1**b presents a case where the workload of vEPC varies during the day. To face these variations, it is necessary to scale vEPC, taking advantage of both vertical and horizontal scaling. This paper argues that moving between vertical and horizontal scaling is necessary to support workload variations in vEPC and avoid the wasting of resources. For instance, when the initial deployment of vEPC reaches a predefined threshold due to a traffic peak, it is possible to take advantage of the simplicity of the vertical scaling by increasing the resources allocated to MME, SGW, and/or PGW. If the vertical scaling is not enough, it is possible to leverage the horizontal scaling by creating new instances of MME, SGW, and PGW. When the traffic peak disappears, it is possible to remove the unnecessary instances and decrease the resources allocated to MME, SGW, and PGW.



(a) Combining vertical and horizontal scaling.



(b) Scaling in a daily workload.

Figure 3-1.: Motivating scenario.

3.1.2. General Operation

Fig. **3-2** presents how the mechanism operates in a high-abstraction level. Initially, it is necessary to define performance regions (*i.e.,* regions I, II, and III are delimited by points A, B, and C in Fig. **3-1**a) of vEPC by evaluating its behavior with static resources and with variations in both number of concurrent users (*i.e.,* C_{users1} and C_{users2}) and number of registrations per second (*i.e.,* Th_1 and Th_2). In this approach, the Network Administrator participates only during the estimation of regions. After the regions are defined, the mechanism operates as follows.

If the vEPC performance is in the region I, the mechanism does not perform any action because vEPC with static resources allocation supports the workload. If the vEPC performance passes from Region I to Region II, the vertical scaling is activated and, thus, the resources per VNF are increased to face the workload. If the performance passes from Region II to Region III, the horizontal scaling starts and, so, more VNFs are created to support the

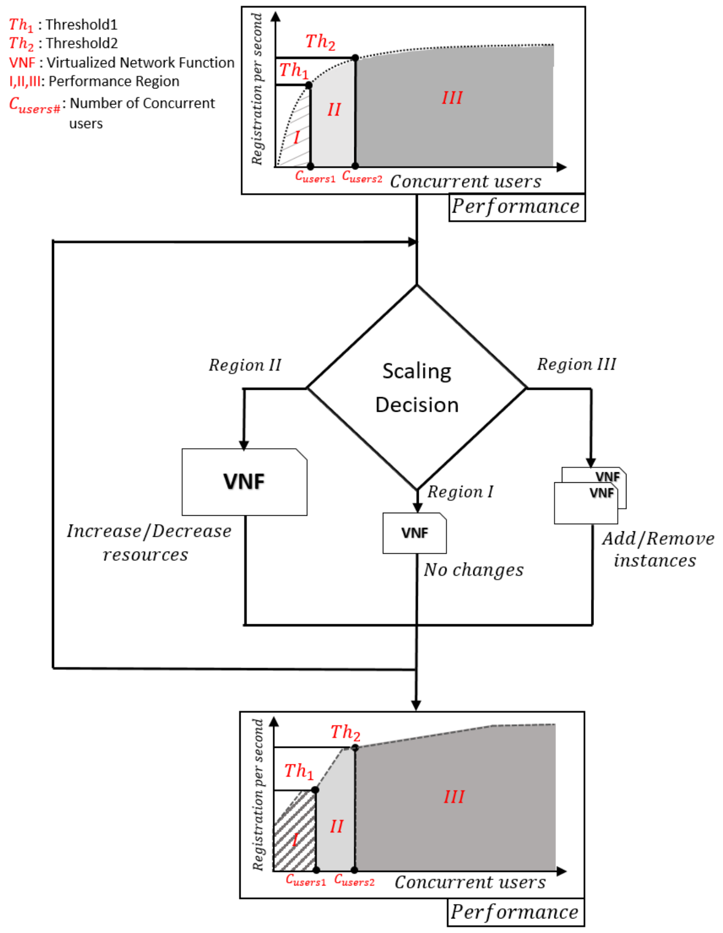


Figure 3-2.: Scaling mechanism.

workload. If the performance passes from Region III to Region II, one or more VNFs are removed to handle the workload. If the vEPC performance passes from Region II to Region I, VNF resources are released to support the workload.

3.1.3. Algorithm

Algorithm 1 details the process to perform scaling in vEPC by moving between horizontal and vertical (or vice-versa) scaling depending on workload variations. The algorithm has as input data the number of concurrent users (n), vEPC performance metrics (*i.e.*, number of registrations per second, latency, and RAM and CPU usage), vEPC performance thresholds (*i.e.*, Th_1, Th_2, C_{users1}, and C_{users2}), and the configuration (*i.e.*, horizontal -instances- or vertical -resources-) of vEPC entities. The algorithm results are the vEPC performance and the new configuration for achieving it.

The proposed algorithm operates as follows. If the number of registrations per second is lower than Th_1 and the number of concurrent users is lower than C_{users1}, the vEPC performance is in Region I. Thus, the current vEPC configuration does not need changes since it can handle the workload. Therefore, it is not necessary to apply neither vertical nor horizontal scaling.

If the number of registrations per second is between Th_1 and Th_2, and the number of concurrent users is between C_{users1} and C_{users2}, the vEPC performance is in region II. Therefore, it is necessary to apply vertical scaling by adding resources (storage, processing, and/or memory) to VNFs. When the workload varies and the performance returns to Region I, it is necessary to apply vertical scaling by reducing resources to avoid their wasting.

If the number of registrations per second is higher than Th_2 and the number of concurrent users is greater than C_{users2}, the vEPC performance is in region III. Therefore, it is necessary to apply horizontal scaling by adding the number of instances of MME, SGW, and PGW. When the workload varies and the performance returns to Region II, it is necessary to apply horizontal scaling by diminishing the number of instances of MME, SGW, and PGW.

```
**Data:** Number of concurrent users ($n$), performance metrics, performance
      thresholds, a vEPC scaling configuration
**Result:** Performance metrics and a new vEPC scaling configuration
**for** *each* $t$ **do**
    **if** *number of registrations per second* $< Th_1$ **and** $n < C_{users1}$ **then**
        Region I;
        Activate vEPC configuration with static resources allocation → MME(),
          SGW(), PGW();
    **else if** $Th_1 \leq$ *number of registrations per second* $< Th_2$ **and** $C_{users1} \leq n <$
      $C_{users2}$ **then**
        Region II;
        Activate vEPC configuration with vertical scaling → MME(# resources),
          SGW(# resources), PGW(# resources);
    **else if** *number of registrations per second* $\geq Th_2$ **and** $n \geq C_{users2}$ **then**
        Region III;
        Activate vEPC configuration with horizontal scaling → MME(# instances),
          SGW(# instances), PGW(# instances);
**end**
```

Algorithm 1: Scaling in vEPC.

3.1.4. Mechanism Modules

Fig. **3-3** illustrates the modules of the mechanism: Data Collection, Scaling Decision, and Scaling Execution. The Data Collection is in charge of monitoring and collecting data from the vEPC. This module takes the vEPC performance measurements (*i.e.,* registrations per second, latency, CPU, and RAM) when the number of concurrent users varies. Once the Data Collection gathers the performance data, it stores them and plots the performance metrics versus the number of concurrent users to establish the performance behavior of the vEPC. By using such behavior information, the network administrator can determine when the vEPC is becoming saturated and define the thresholds values to obtain/maintain a vEPC performance target. Finally, it is noteworthy that this module provides to the Scaling Decision the vEPC performance information needed to make decisions.

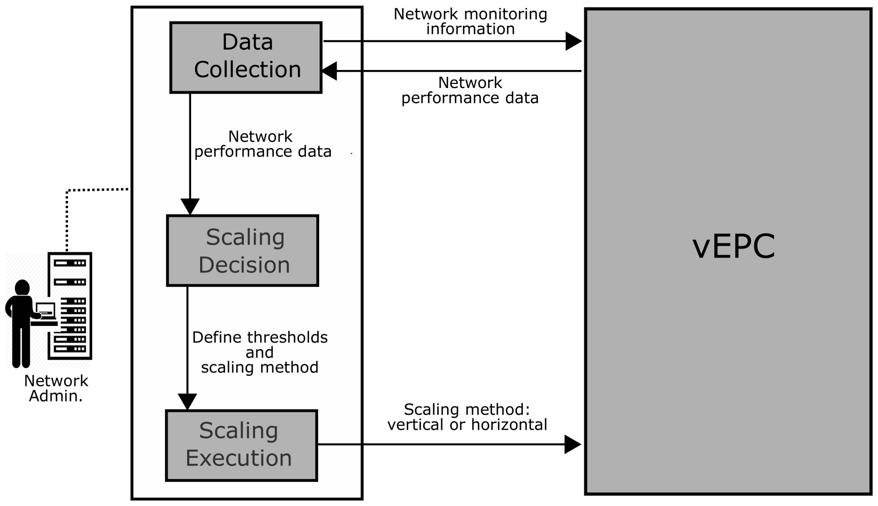


Figure 3-3.: Modules of the scaling mechanism.

The Scaling Decision uses Algorithm 1 to make decisions about what scaling method and how many resources must be used to reach the target performance defined by the network administrator. This module informs such decisions to the Scaling Execution that is in charge of applying them. To sum up, the vertical scaling increases or decreases the resources allocated to vEPC entities (MME, SGW, and PGW). The horizontal scaling adds or removes the instances of these vEPC entities. It is important to highlight that if the Scaling Decision module determines that there is no need for scaling, the Scaling Execution does not perform any action, and vEPC without scaling supports the workload.

3.2. Evaluation and Analysis

This section presents the evaluation results of the mechanism. First, the section exposes the test environment. Second, the section presents the evaluation and performance analysis of the vEPC.

3.2.1. Test Environment

To evaluate the mechanism, the open source vEPC from the Indian Institute of Technology Bombay (IIT Bombay) (Vutukuru et al., 2016) is deployed in AWS. The IIT Bombay vEPC (from now on called B-vEPC) emulates the behavior of a typical EPC based on NFV that handles control and data traffic. B-vEPC has two versions. Evaluation uses version 1.0 to define the vEPC baseline (evaluation without scaling) and to analyze vertical scaling (see Fig. **3-4**). In turn, version 2.0 is used to analyze the behavior of vEPC with horizontal scaling (see Fig. **3-5**).

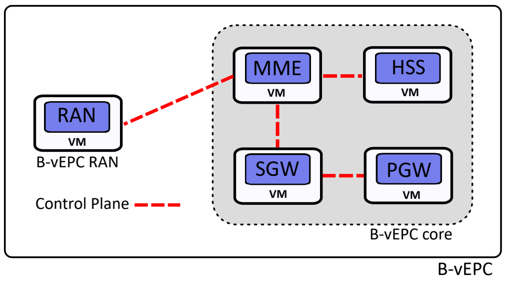


Figure 3-4.: B-vEPC for the baseline and vertical scaling.

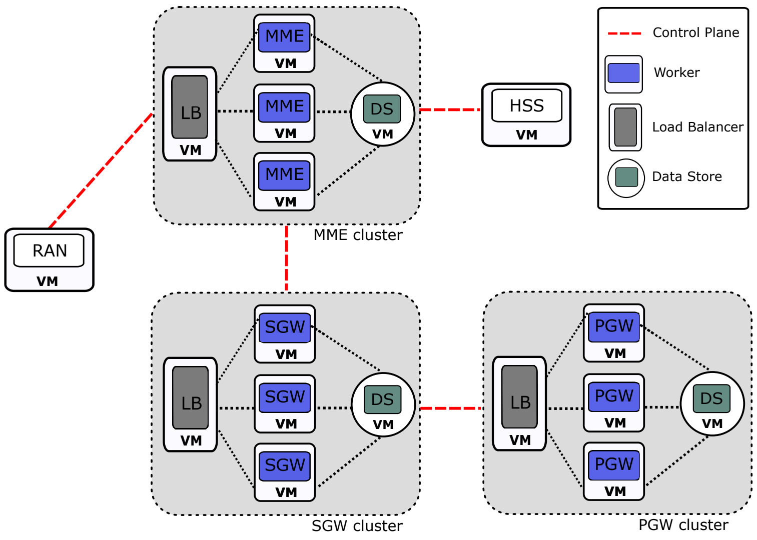


Figure 3-5.: B-vEPC for horizontal scaling.

About the B-vEPC emulator (see Fig. **3-4**), it is necessary to mention that RAN is a module that combines UE and eNodeB functionalities and generates control traffic to the core entities (MME, SGW, PGW, and HSS). RAN does not implement the radio processes that take place between UE and eNodeB; this module only focuses on the control and data traffic that B-vEPC handles. Furthermore, RAN generates multiple threads related to attach and detach processes and also handles communication with MME. B-vEPC responds to UE attach and detach processes from controlling traffic. Attach is the process to connect UE (represented by RAN) to B-vEPC and includes the authentication, security and session setup. Detach is the process to disconnect UE from B-vEPC.

Since the focus is on assessing the scaling of B-vEPC when it handles control traffic, the performance evaluation is regarding registrations per second, latency, and usage of CPU and RAM. The registrations per second refer to the number of processes of attaching and detaching completed by B-vEPC (Baldo et al., 2011). Latency is the time that a UE takes to perform the processes mentioned above (Nikaein and Krea, 2011). Evaluation considers that the CPU and RAM usage are at the saturation level when they reach values higher than 90% in any vEPC entity (Felter et al., 2015).

To carry out the performance evaluation, the number of concurrent users of B-vEPC is varied. This number was 10, 25, 50, 100, and 200. It is relevant to mention that B-vEPC supports a maximum of 200 concurrent users. This number of concurrent users is not a limiting factor for the evaluation because, for example, B-vEPC can generate 16,440 registrations in 120 seconds, it means 137 registrations per second. All evaluation cases took the average values for 30 measurements with a 95% confidence level and performed all tests in a 120 seconds interval.

3.2.2. Implementation and Deployment

The prototype of the mechanism was implemented by using Python 3.6. In particular, the Data Collection module was developed with the library Requests 1.1.0 (Chandra and Varanasi, 2015) that is responsible for providing to the Scaling Decision the current vEPC status (*i.e.,* number of concurrent users, number of registrations per second, and CPU usage). The Scaling Decision module was implemented by using Flask 1.0.2 (Grinberg, 2014) that is in charge of receiving the vEPC status from Data Collection and comparing it with predefined thresholds. The Scaling Execution module was developed with the AWS-CLI 1.16 API that allows modifying instances to provide both vertical and horizontal scaling.

The mechanism and the B-vEPC were deployed in AWS. For the vertical scaling, the Scaling Decision module used an Identity and Access Management (IAM) role that is useful to allocate and de-allocate resources securely to any AWS entity (*e.g.,* a VM instance). For the horizontal scaling, three different Virtual Private Clouds (VPCs) were used to isolate the clusters of MME, PGW, and SGW. The access to these clusters was enabled through the public IP addresses of Load Balancers. Detailed explanation of the prototype and the corresponding instructions to its deploying and testing are in (Maca et al., 2019).

3.2.3. Performance with Static Resources Allocation - Baseline

Table **3-1** presents the allocated resources for the baseline evaluation (see Fig. **3-4**). Note that each entity is a VNF running on a VM that communicates by the client/server paradigm with the next entity responsible for processing the requests and sending the corresponding responses of the B-vEPC control plane.

Table 3-1.: Resources allocated to the baseline.

Entity	**Resources**		
	RAM[1]	**CPU**[2]	**Storage**[1]
RAN	4	4	10
MME	1	1	10
HSS	2	1	10
SGW	1	1	10
PGW	1	1	10

[1] Gigabytes
[2] Cores

Fig. **3-6** depicts the baseline evaluation results of B-vEPC regarding registrations per second. Stress tests use 200 concurrent users, generating 137 registrations per second during 120 seconds, achieving a total of 16,440 registrations. The evaluation results reveal that the slope of the number of registrations per second is positive and steeper up to 50 concurrent users. From this point, when the number of concurrent users increases, the slope of number of registrations per second decreases profoundly and after 100 is near zero. Thus, to keep the registrations per second, it is necessary to apply a scaling method.
Fig. **3-7** presents the baseline evaluation results of B-vEPC regarding latency. These results reveal that latency is lower than 100 ms up to 50 concurrent users; however, after this point, the latency increases up to 230 ms. These results agree with the obtained values of registrations per second. Thus, after 50 concurrent users, the time to perform the attach and detach processes increases considerably, which may avoid that B-vEPC completes these processes successfully.
Fig. **3-8** depicts the B-vEPC baseline evaluation results regarding CPU usage. These results reveal that the use of CPU in MME and SGW is near to 90% for 50 concurrent users and near to 96% for 200 concurrent users. This CPU behavior indicates that MME and SGW are at (or almost) saturation levels, explaining the fall of the slope of registrations per second and the increasing of delay (see Fig. **3-6** and Fig. **3-7**). It is necessary to mention that the use of CPU in PGW behaves differently from MME and SGW, increasing only up to

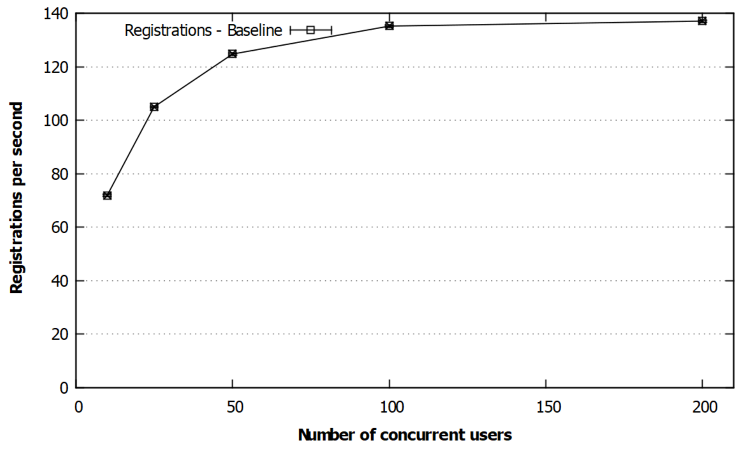


Figure 3-6.: Registrations per second in the baseline evaluation.

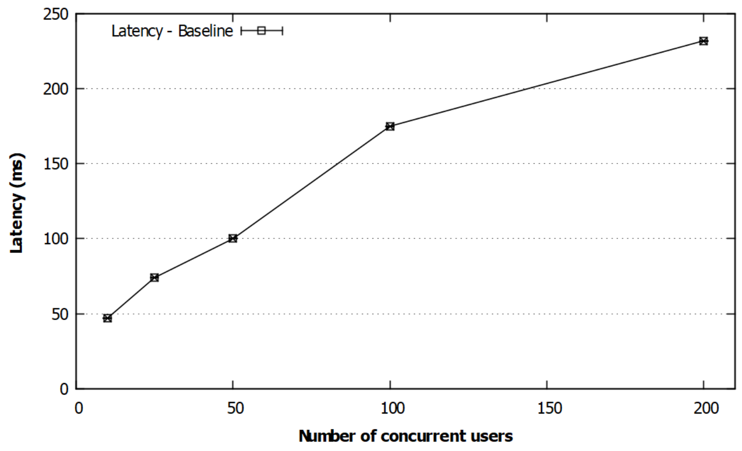


Figure 3-7.: Latency in the baseline evaluation.

59%. This behavior indicates that PGW is less relevant than MME and SGW for the control processes in B-VEPC.

Fig. **3-9** illustrates the B-vEPC baseline evaluation results regarding RAM usage. These results reveal that the maximum use of RAM is 620 MB. This behavior indicates that RAM does not saturate for any workload variation and, therefore, RAM presents a minimal influence on the performance behavior of B-vEPC.

To sum up, the baseline evaluation of B-vEPC reveals that MME and SGW play an essential role in control plane processes. In particular, bottlenecks are in MME and SGW that limit the B-vEPC performance. Furthermore, RAM has a minimal influence on such performance.

3.2.4. Performance with Vertical Scaling

This performance evaluation varied the processing capacity of the B-vEPC entities involved in the control plane processes. In particular, tests increased the number of processing cores

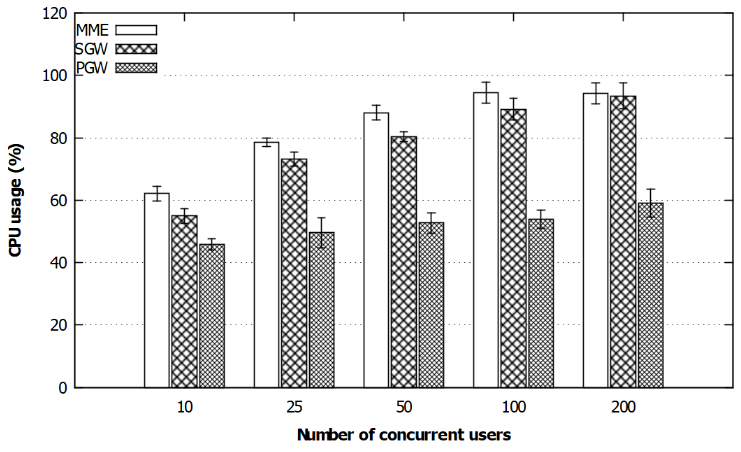


Figure 3-8.: CPU usage in the baseline evaluation.

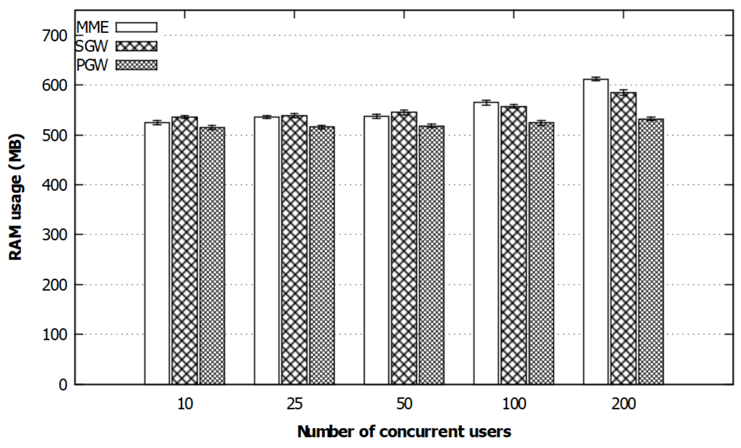


Figure 3-9.: RAM usage in the baseline evaluation.

per entity of B-vEPC to analyze its performance behavior regarding registrations per second, latency, and CPU usage. Note that tests did not analyze RAM since the previous baseline analysis revealed its negligible incidence in the B-vEPC performance.

Vertical scaling evaluation used between one and three cores per entity (MME, SGW, and PGW), giving a total of 27 alternatives. All possible configurations were analyzed; however, for the sake of brevity, the next paragraphs only discuss the most important results. These results were chosen by taking into account that, for instance, increasing the number of PGW cores did not improve the performance results; it was due to, in the control processes, PGW did not consume much processing and, so, increasing cores just led to their wasting. Table **3-2** presents the most relevant configurations for each B-vEPC entity. Vx denotes Configuration number 1, 2, 3 or 4 for vertical scaling.

Fig. **3-10** illustrates the evaluation results of the B-vEPC performance for vertical scaling

Table 3-2.: B-vEPC configurations for vertical scaling.

Entity	V1		V2		V3		V4	
	RAM[1]	CPU[2]	RAM	CPU	RAM	CPU	RAM	CPU
MME	1	2	1	1	1	2	1	3
SGW	1	1	1	2	1	2	1	3
PGW	1	1	1	1	1	1	1	1

[1] Gigabytes
[2] Cores

regarding registrations per second. Stress tests used 200 concurrent users, generating 137 registrations per second during 120 seconds, achieving a total of 16440 registrations. These results reveal, first, scaling both MME and SGW is better (approx. 63%) than scaling just MME or only SGW. Second, with two CPU cores in MME and SGW (Config V3), registrations per second are higher (approx. 80%) than the baseline. Third, with three CPU cores in MME and SGW (Config V4), the registrations per second are higher (approx. 89%) than the baseline. Fourth, increasing more than three CPU cores in MME and SGW does not lead to a significant increase in the performance regarding registrations per second because the resources are over-provisioned.

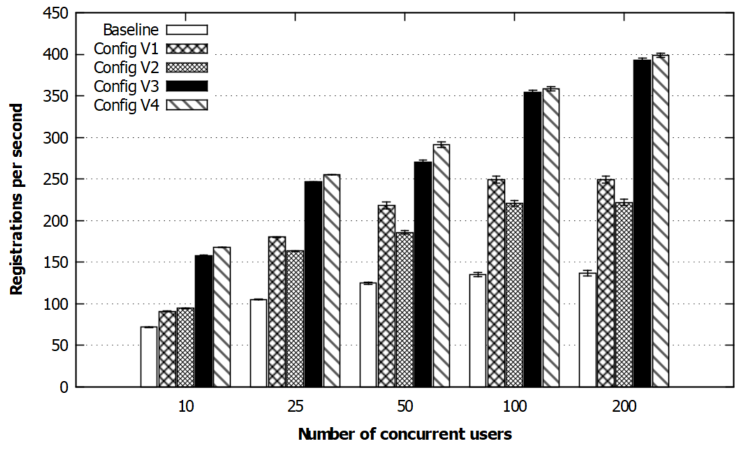


Figure 3-10.: Registrations per second with vertical scaling.

Fig. **3-11** depicts the evaluation results of the B-vEPC performance for vertical scaling regarding latency. These results reveal that when scaling both MME and SGW, the latency is lower (approx. 54%) than when scaling just MME or only SGW, and lower (approx. 70%) than the baseline. Furthermore, with three CPU cores in MME and SGW (Config V4), the latency is lower (approx. 5%) than with two CPU cores in MME and SGW (Config V3).
Fig. **3-12** presents the B-vEPC performance evaluation results regarding the use of CPU in MME. These results reveal that when the number of concurrent users is equal to or higher than 50, the MME from the baseline saturates quickly reaching levels higher than 90%. For

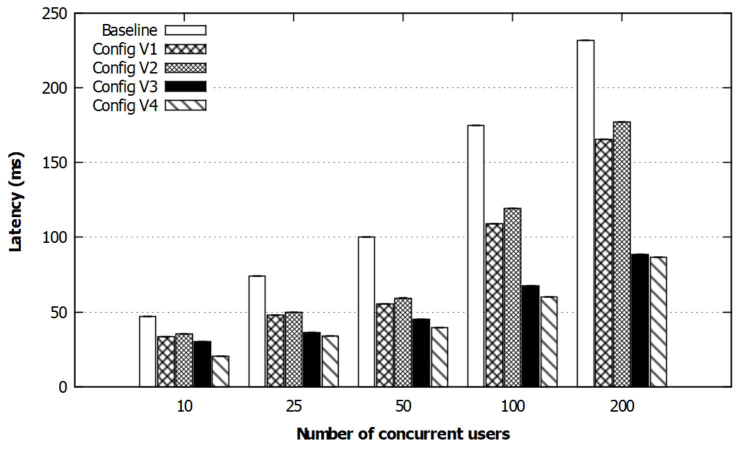


Figure 3-11.: Latency with vertical scaling.

instance, for 200 concurrent users, the CPU usage is almost 97%. This saturation indicates that MME is the first entity that needs to scale vertically to handle control processes. When MME scaled vertically (Config V1), its maximum use of CPU is 65% for 200 concurrent users. Also, SGW scaled vertically only (Config V2) to test its relevance in the control plane processes. As expected, the evaluation results reveal that scaling only SGW leads to an increase in the use of CPU in MME because SGW generates an increasing workload to MME. When MME and SGW vertically scale simultaneous and uniformly, the use of CPU in MME never reached the CPU saturation level. Here, uniform vertical scaling means that SGW and MME used the same resources.

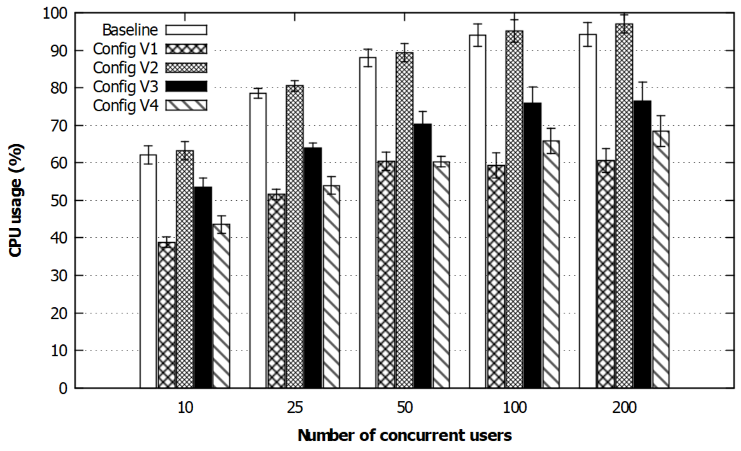


Figure 3-12.: CPU usage in MME with vertical scaling.

Fig. **3-13** illustrates the B-vEPC performance evaluation results regarding the use of CPU in SGW. These results reveal that when the number of concurrent users is equal to or higher than 50, the SGW from the baseline saturates quickly reaching levels higher than 90%. For instance, for 200 concurrent users, the CPU usage is almost 97%. It is important to note that scaling only MME vertically (Config V1) leads to saturating the CPU of SGW because MME sends too many control requests to SGW. In this sense, when MME and SGW scale vertical and uniformly, the use of CPU in MME and SGW does not exceed the CPU saturation level. These results corroborate the relevance of MME and SGW in control processes and, thus, the importance of scaling them simultaneous and uniformly.

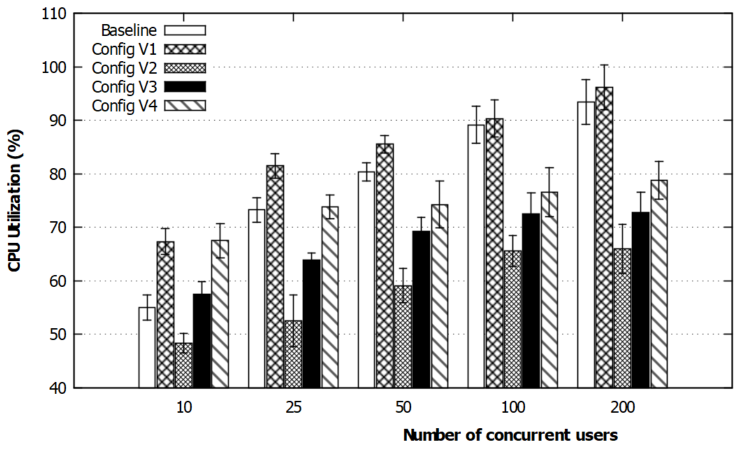


Figure 3-13.: CPU usage in SGW with vertical scaling.

Fig. **3-14** depicts the B-vEPC performance evaluation results regarding the use of CPU in PGW. These results reveal that although PGW handles the control requests received from SGW, its use of CPU never reaches the CPU saturation level for any of the vertical scaling configurations. These results indicate that PGW does not require vertical scaling for handling the workload of control processes.

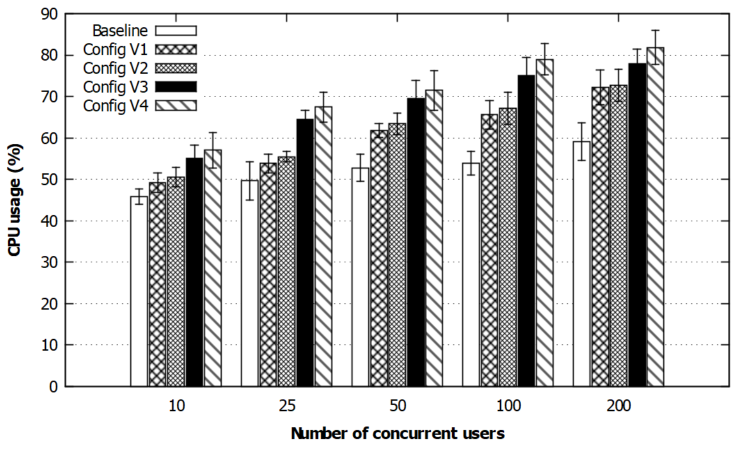


Figure 3-14.: CPU usage in PGW with vertical scaling.

To sum up, the B-vEPC performance evaluation for vertical scaling highlights the following facts. First, the number of processing cores is the most relevant resource to handle the control processes. Second, PGW does not require vertical scaling for 200 concurrent or less because it does not exceed the CPU saturation level in any configuration. Third, B-vEPC reaches its best performance when MME and SGW scale simultaneously and uniformly. Fourth, scaling vertically with three CPU cores in MME and SGW (Config V4) does not represent a significant improvement in performance regarding registrations per second and latency in comparison with scaling with two CPU cores in MME and SGW (Config V3). The above results figure out that it is necessary to use another scaling method to improve the performance of B-vEPC.

3.2.5. Performance with Horizontal Scaling

Fig. **3-5** presents the B-vEPC used to perform horizontal scaling. This B-vEPC consists of a set of clusters for MME, SGW, and PGW. Each cluster contains a load balancer, workers, and a data store. The load balancer uses the round-robin algorithm Farooq et al. (2017) to distribute the control traffic among the workers. The workers are exact replicas of the B-vEPC entities used in the baseline evaluation; these workers connect to the load balancer and the data store. The data store is in charge of storing the state of workers, and its primary function is to guarantee fault tolerance of the entity. For instance, if a worker fails, the data store assigns the workload to another worker that is in operation.

The clusters of MME, SGW, and PGW operate with stateless workers that save, delete, update, and read signaling information in a shared data store. This sharing of information enables any stateless worker in the cluster to serve any UE. For example, once a UE_x performs an attach process served by an $\text{MME}worker_x$, another $\text{MME}worker_y$, can perform the corresponding detach by retrieving from the data store the information of UE_x.

Horizontal scaling evaluation deployed between one and three workers per cluster, giving a total of 27 alternatives. All alternatives were evaluated and analyzed; however, for the sake of brevity, the next paragraphs discuss only the most relevant results. These results were selected by taking into account that, for instance, increasing the number of PGW workers did not improve the performance results; it is because, in the control processes, PGW did not consume much processing and, so, increasing workers just led to their wasting. Table **3-3** presents the most significant configurations carried out per cluster. Hx denotes Configuration number 1, 2, 3, 4, 5 or 6 for horizontal scaling. Table **3-4** presents the resources assigned to B-vEPC for horizontal scaling (Satapathy et al., 2017).
Fig. **3-15** illustrates the evaluation results of the B-vEPC performance for horizontal scaling regarding registrations per second. Stress tests used 200 concurrent users, generating 137 registrations per second during 120 seconds, achieving a total of 16440 registrations. These results reveal:

- With two workers in the MME cluster (Config H1), registrations per second are better

Table 3-3.: B-vEPC configurations for horizontal scaling.

Cluster	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**
	Workers	**Workers**	**Workers**	**Workers**	**Workers**	**Workers**
MME	2	2	3	3	3	3
SGW	1	2	1	2	3	3
PGW	1	1	1	1	1	2

Table 3-4.: Resources allocated for horizontal scaling.

Entity	**Resources**		
	RAM [1]	**CPU** [2]	**Storage** [1]
RAN	4	4	10
MME	1	1	10
HSS	2	1	10
SGW	1	1	10
PGW	1	1	10
LOAD BALANCER	2	1	10
DATA STORE	2	2	15

[1] Gigabytes
[2] Cores

(approx. 16%) than the baseline.

- With three workers in MME and one worker in SGW clusters (Config H3), the registrations per second are higher (approx. 36%) than baseline.

- If the number of workers is increased in equal proportion in all clusters, the number of registrations per second increases. This increase is about 186% with two workers in MME and SGW clusters (Config H2) and near to 308% with three workers in MME and SGW (Config H5).

- Two workers in the PGW cluster (Config H6) do not generate a significant improvement in performance regarding registrations per second.

- Increasing only the number of workers in the MME cluster becomes a bottleneck between MME and SGW because SGW cannot respond to the requests from MME.

- B-vEPC behaves better in performance regarding registrations per second when the number of workers in MME and SGW is the same.

Fig. **3-16** depicts the evaluation results of the B-vEPC performance for horizontal scaling regarding latency. Although the number of registrations per second increases because the workers distribute the workload to respond to more requests, B-vEPC with horizontal scaling adds nodes that, in turn, increase latency. Thus, scaling needs a trade-off between the

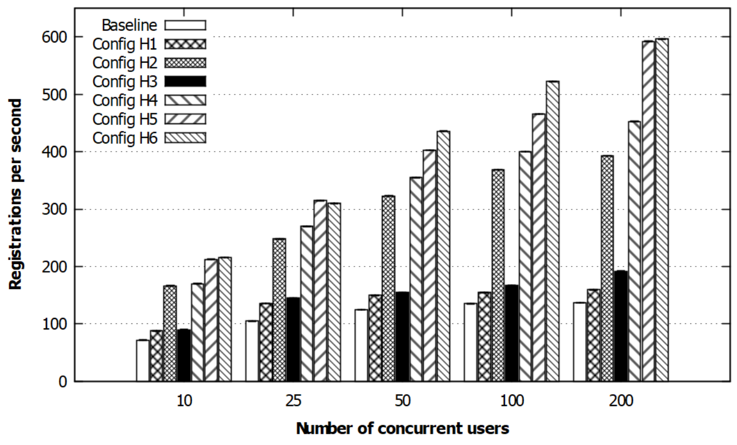


Figure 3-15.: Registrations per second with horizontal scaling.

number of workers, the registrations per second, and the latency.

To sum up, the B-vEPC performance evaluation for horizontal scaling provides the following results. First, MME and SGW are the clusters that most affect the performance. Second, PGW does not require horizontal scaling for 200 concurrent users or less because increasing the number of workers in PGW does not provide a significant improvement in performance. Third, B-vEPC presents its best performance when the workers of MME and SGW are uniform. Fourth, the increasing of workers in the MME and SGW must provide a trade-off between latency and registrations per second.

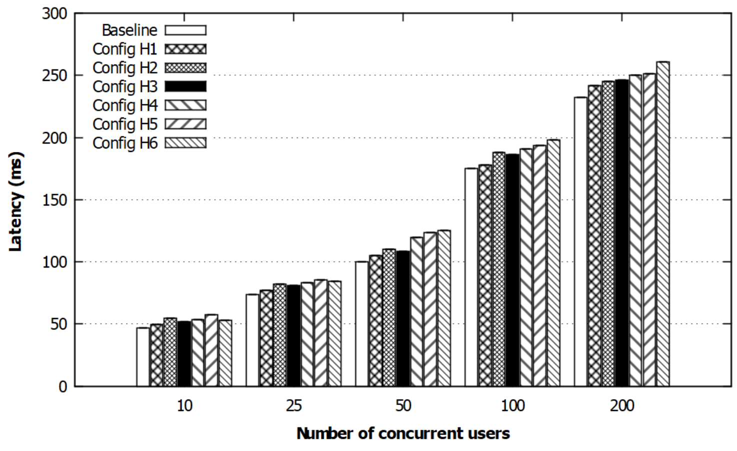


Figure 3-16.: Latency with horizontal scaling.

3.2.6. Thresholds Definition

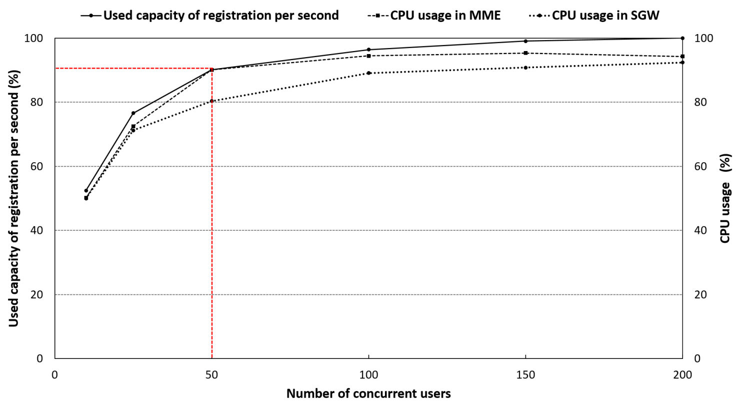
The mechanism operates by using three regions that determine the changing from one scaling method to another one. To define these regions (I, II, and III), tests evaluate the configurations V3 (see Table III) and H5 (see Table IV) considering mainly two metrics: the use of CPU and the capacity of registrations per second (*i.e.,* the relation between the maximum and the current number of concurrent supported users). These configurations were selected because, according to the previous evaluations (Subsections 3.2.3, 3.2.4, and 3.2.5), they offer a good trade-off between the consumption of resources and the performance metrics. Regarding the first metric, evaluation considers that if the CPU usage of any EPC entity is higher than 90% (Felter et al., 2015), B-vEPC is in saturation level (see Fig. **3-17**a and **3-17**c) and, thus, B-vEPC needs scaling. About the second metric, evaluation considers that if it is higher than 90%, B-vEPC is in saturation level (see Fig. **3-17**a, **3-17**b, and **3-17**c) and, so, B-vEPC needs scaling.

Fig. **3-17**a illustrates the used capacity of registrations per second for B-vEPC without scaling (baseline) and the corresponding CPU usage in MME and SGW. Note that for 50 concurrent users, both the used capacity of registrations per second and the CPU usage in MME reach 90%. This fact frames Region I. Fig. **3-17**b depicts the used capacity of registrations per second for B-vEPC with vertical scaling (Config V3) and the corresponding CPU usage in MME and SGW. Note that for 100 concurrent users, the used capacity of registrations per second reaches 90%. This fact frames Region II since if only one metric is in saturation level, it is needed to scale. Fig. **3-17**c illustrates the used capacity of registrations per second for B-vEPC with horizontal scaling (Config H5) and the corresponding CPU usage in MME and SGW. For 175 concurrent users, both the used capacity of registrations per second and the CPU usage in MME reach 90%. This fact frames Region III.

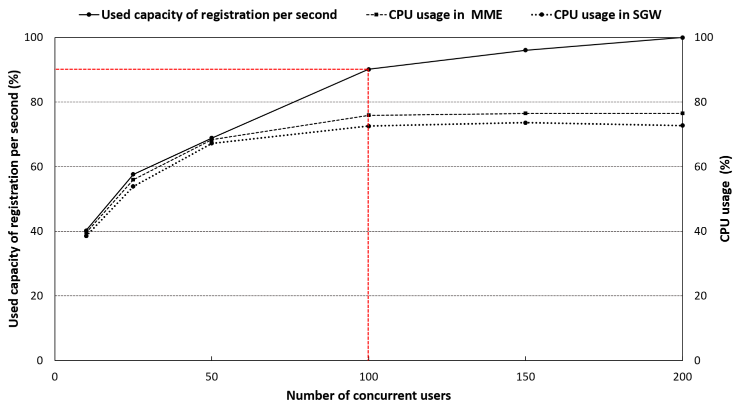
In summary, Region I is framed up to 50 concurrent users. Region II is from 51 to 100 concurrent users. Region III is from 101 to 175 concurrent users. Recall that Fig. **3-2** depicts these regions.

3.2.7. Performance with both Vertical and Horizontal Scaling

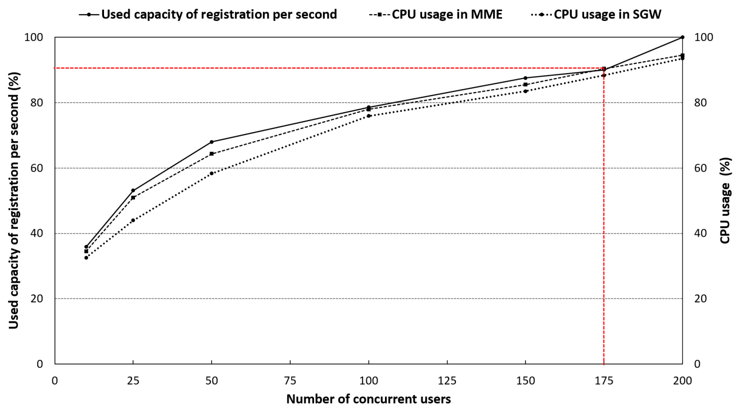
To evaluate the scaling mechanism, tests vary the number of concurrent users from 40 to 70, 70 to 20, 20 to 60, 60 to 150, 150 to 175, and 175 to 100, and measure the used capacity of registrations per second as well as the CPU usage (in MME and SGW). It is noteworthy that in the evaluations, the execution period of Algorithm 1 was setting up at 10 minutes. In this period, Algorithm 1 compares the measured average throughput with the previously defined thresholds. The rate of change in scaling configurations depends on the rate of variation of the number of users.



(a) B-vEPC performance baseline.



(b) B-vEPC performance with vertical scaling.



(c) B-vEPC performance with horizontal scaling.

Figure 3-17.: Evaluation of B-vEPC configurations.

Fig. **3-18** presents the evaluation results, revealing:

- As expected that B-vEPC with baseline configuration can handle 40 concurrent users because these amount of users is in Region I.

- At the next point, B-vEPC with vertical scaling handles the 70 concurrent users (Region II). When the number of concurrent users falls to 20, B-vEPC without scaling supports the workload again.

- B-vEPC with vertical scaling is again necessary for 60 concurrent users.

- B-vEPC with horizontal scaling is necessary for handling 150 concurrent users.

- In the last two points, from 175 to 100 concurrent users, the mechanism passes from horizontal scaling to vertical scaling.

- Finally, it is important to highlight two facts about the mechanism. First, it always uses less than 90% of the CPU available in the EPC entities. Second, it keeps the used capacity of registrations per second between 65% and 90%. These facts indicate that the mechanism handles workload variations and saves resources.

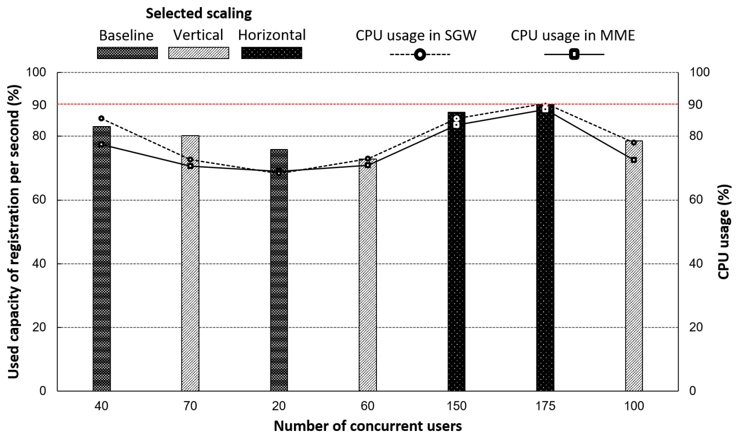


Figure 3-18.: Scaling mechanism evaluation.

Considering the above results, the proposed mechanism has the following features:

- It is automatic and reactive.

- It uses thresholds to determine when moving from a scaling method to another one.

- The Network Administrator participates actively in the setting up of thresholds aforementioned. The definition of thresholds without human intervention (autonomically) is out of the scope of the scaling mechanism described in this chapter.

3.2.8. Initial Cost Analysis

This section presents an initial analysis of the proposed mechanism regarding OPEX. vEPC has an OPEX related to the power consumption of servers hosting the VMs used to deploy the vEPC VNFs, namely MME, SGW, PGW, and HSS. Considering this relationship and according to (Bouras et al., 2016), the OPEX of vEPC is defined as follows.

$$OPEX_{vEPC} = n_{servers} \cdot P_{per\_server} \cdot C_{kWh} \tag{3-1}$$

Where $n_{servers}$ is the total number of servers used in the vEPC deployment, P_{per_server} is the power consumption of each server, and C_{kWh} is the per hour cost of a kilowatt. Evaluations (see Section 3.2) used one sever to the baseline, one server to B-vEPC with vertical scaling and two servers to B-vEPC with horizontal scaling.

Consider, first, the vEPC has been deployed by using HP Blade Servers. Second, the average power per Blade at 90% of the processing load is 280 Watts (Beckett and Bradfield, 2011). Third, in 2019, the cost of a kilowatt per hour in Popayán (Colombia) is 0.17 USD. Using these values in Equation 3-1, the costs per hour of the baseline, B-vEPC with vertical scaling, and B-vEPC with horizontal scaling are 0.048 USD, 0.048 USD, and 0.10 USD, respectively.

In the evaluation scenario presented in Fig. **3-18**, the scaling mechanism uses horizontal scaling twice and five times vertical scaling. If this scaling pattern repeats daily, the monthly cost of the proposed mechanism is 44.54 USD. In turn, the monthly cost of B-vEPC with statically allocated resources is 69.29 USD. Therefore, in the scenario above mentioned, the mechanism saves approximately 35.7%. This brief analysis highlights the advantages of using the proposed mechanism to save costs related to energy consumption.

3.2.9. Qualitative Analysis

Individual or combined scalability. The works (Premsankar et al., 2015, Prados-Garzon et al., 2017a, Banerjee et al., 2015, Amogh et al., 2018) reduce the problem of scaling EPC by scaling MME individually. Meanwhile, the works (Satapathy et al., 2017, Ren et al., 2016), and the work presented in this chapter worry about scaling the three major entities of EPC. Scaling just MME requires the oversizing of SGW and PGW to avoid bottlenecks that can affect the performance of EPC as a whole. Recall that oversizing leads to waste resources outside peak demand hours. Furthermore, according to evaluation results, oversizing PGW is unnecessary for supporting traffic related to signaling processes.

Evaluation method. Performance evaluations can be carried out by using theoretical and simulation models, data traces, testbeds, and real environments. The theoretical and simulation models provide flexibility to work with a large number of EU and workers of EPC entities.

However, such models consider certain assumptions that abstract the main characteristics of behavior and performance of EPC. Data traces offer greater flexibility for data analysis but, for scalability analysis, they require the collection of metrics for all possible EPC scaling configurations, which may be unfeasible in production EPC environments. The use of testbeds and real environments allows analyzing the behavior and performance of each EPC entity more realistically than in simulation and data traces. Nonetheless, testbeds and real environments usually are resources-constrained. The works (Prados-Garzon et al., 2017a, Ren et al., 2016) use simulations to perform their LTE EPC scaling evaluations. the work (Alawe et al., 2018b) uses several data traces to evaluate their scaling approach for 5GCN. The works (Banerjee et al., 2015, Amogh et al., 2018, Satapathy et al., 2017) evaluate their scaling solutions by emulations on testbeds. It is noteworthy that unlike the works above mentioned, the solution proposed in this chapter has been deployed and tested in a real public cloud by considering a workload of up to 200 concurrent users and three instances per EPC entity.

Scaling metrics. Performance metrics allow making decisions about when scaling EPC entities. The work (Prados-Garzon et al., 2017a) uses the mean system delay metric. The works (Banerjee et al., 2015, Amogh et al., 2018) consider the CPU utilization metric. The work (Alawe et al., 2018b) uses the control traffic requests metric. The scaling mechanism uses thresholds based on the metrics: CPU utilization and used capacity of registrations per second. Once an EPC entity is near to outcome a threshold, the solution triggers another scaling mechanism to avoid EPC goes to saturation zones. Keeping the EPC entities in the non-saturation zone implies that latency has low values and the attending of registration requests.

3.3. Final Remarks

This chapter presented a scalability and performance characterization of an NVF-based EPC followed by a scaling mechanism that moves from vertical to horizontal scaling (and vice-versa) to adapt EPC to workload variations and avoid resource wasting.

In detail, the scaling mechanism provides the capability to adapt to variations in the number of concurrent users for handling control traffic. The mechanism determines if an initial static configuration of EPC can handle a particular workload or if it is necessary to increase resources or generate replicas of EPC entities to support such a workload. The evaluation confirmed that MME and SGW are the most critical EPC control entities in both vertical and horizontal scaling; they must be scaled simultaneously using the same amount of resources. When EPC scales vertically, the latency is 70% lower than the baseline, and the registrations per second are almost 180% higher than the baseline. When EPC scales horizontally, the registrations per second are 308% higher than the baseline, and the latency increases; thus, it is necessary to consider a trade-off between these metrics. Also, this chapter presented an initial costs analysis highlighting that the proposed mechanism is cost-effective regarding energy consumption.

4. An Adaptive Scaling Mechanism for Managing Performance Variations in Network Functions Virtualization

NFV enables to dynamically modify the capacity of NSs to face changes such as the number of users and performance variations (ETSI, 2013, Han et al., 2015). NSs are end-to-end functionalities created by composing NFs, virtualized or not (Bhamare et al., 2016). The network performance allows assessing the quality of NSs; it is quantified by measurable parameters (*e.g.,* throughput and delay) (ITU, 2008), and its variation is associated with changes of underlying resources and the usage patterns of services and applications (Shea et al., 2014, Callegati et al., 2014). For instance, in the EPC, a control entity sends and receives signaling messages for a proper operation (3GPP, 2018a). The performance of such an entity can be measured in terms of the Mean Response Time (MRT) that can vary depending on the number of requests to establish sessions, update users location and perform handover (Rajan et al., 2015).

The traditional solution to address performance variations is to oversize the capacity of NFs (Heidari and Kanso, 2016), which means that these are usually designed to support workload peaks. However, oversizing is inefficient at time slots of low utilization. NFV offers an alternative solution, the dynamic scaling of NFs that allows managing performance variations and improving the efficiency of using resources (Mijumbi et al., 2016b). Scaling (ETSI, 2014a), the process of modifying the capacity of NFs, can be performed by increasing and reducing their resources (*i.e.,* scaling up/down) or creating and removing their instances (*i.e.,* scaling out/in). It is noteworthy that scaling can be initiated by administrators, as an outcome of the network performance assessment, or by the network itself by using adaptive mechanisms (ETSI-GS-AFI, 2013).

In the NFV literature, several works have proposed scaling mechanisms by using different techniques. For instance, the solutions (Carella et al., 2016, Dutta et al., 2016) are based on threshold rules, a reactive technique, in which the scaling depends on the current traffic or performance, whose variations can lead to violations of a performance target and transitory scaling oscillations. This problem may be present also in solutions based on optimization, such as (Wang et al., 2016), which takes scaling decisions based only on instant traffic and it does not consider a prediction horizon to evaluate them. The solution (Bilal et al., 2016a) uses time series forecasting that, based on historical data, enables to predict future resource usage; however, if there are changes in traffic patterns, an evolutionary strategy would be desirable, which would allow adapting the models to new conditions. The work (Tang et al., 2015) is based on the Q-Learning method in which, as other methods in RL, an agent interacts with an environment and learns by trial and error; this approach is adaptive but mistaken decisions may be taken until the agent learns an optimal scaling policy.

Considering the above limitations, this chapter argues that the scaling in NFV should be adaptive and highly accurate to avoid violations of expected levels of QoS and transitory scaling oscillations. So, this chapter considers that: (*i*) the use of RL for scaling is a good option since learning evolves whereas agents interact with their environment; and (*ii*) the scaling policy of Q-Learning can be iteratively improved before taking a final decision, and therefore the scaling could be more accurate.

The key contributions presented in this chapter are:

- An adaptive scaling mechanism based on Q-Learning and GPs, which are utilized by an agent to carry out an improvement strategy of a scaling policy, and therefore, to make better decisions for handling performance variations.
- An evaluation of the mechanism for managing variations of MRT in an NFV-based EPC, corroborating it is more accurate than approaches based on static threshold rules and Q-Learning without the policy improvement strategy.

The rest of this chapter is organized as follows. Section 4.1 introduces the proposed mechanism in an NFV-based EPC. Section 4.2 evaluates and analyzes the behavior of the mechanism. And Section 4.3 provides some conclusions.

4.1. Scaling Mechanism

4.1.1. A Motivating Scenario: NFV-based EPC

The current standard of 4G mobile networks is LTE, and its core is EPC (3GPP, 2018a). The main entities of EPC are MME, HSS, SGW, and PGW. MME is the control entity responsible for signaling, mobility management of users, bearer management and QoS provisioning. HSS stores the administrative and user information utilized by MME. SGW and PGW compose the EPC data plane, which is in charge of routing and forwarding packets. For the proper EPC operation, SGW and PGW also interact with MME.

From NSC (John et al., 2013) point of view, EPC is composed of two NSs: an NS for signaling (*i.e.*, control plane) and other NS for packet forwarding (*i.e.*, data plane). Fig. **4-1** depicts these NSs by a Service Graph (SG) that follows the description proposed by (Garay et al., 2016). The nodes: eNB and PDN are Service Access Points (SAPs) of SG. In particular, eNB is part of the access network and represents the base stations from where the mobile users are connected. In turn, PDN is any external data network like the Internet. Note that continuous arrows are service links that represent logical connectivity for the data plane, and dotted arrows are service links for the control plane. Each link is labeled with the protocol used. An SG is a directed graph depicting only one direction of the flow. However, a reverse chain is assumed since the communication is bidirectional.

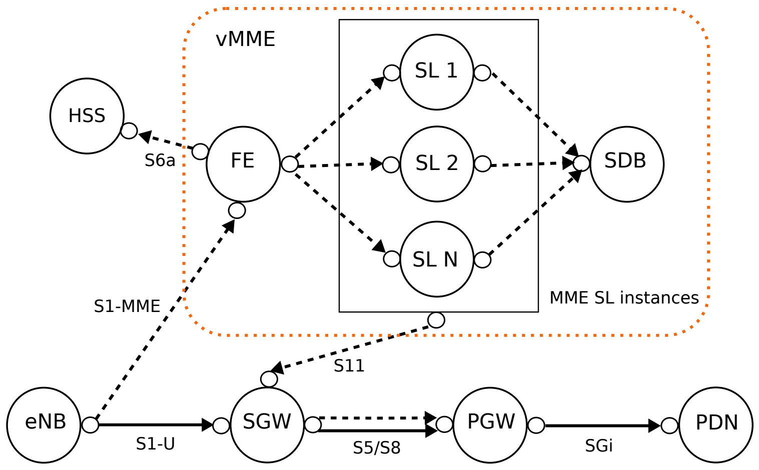


Figure 4-1.: Service graph of an NFV-based EPC.

In Fig. **4-1**, service graph considers a vMME (Prados-Garzon et al., 2017b) that can scale by increasing or decreasing the number of instances of its service logic, which enables to scale the control plane of EPC. For example, EPC may have more demand at evening than early morning depending on the number of service requests from users and, so, it is needed scaling vMME adaptively to improve the use of resources in the control plane. The vMME is formed by three components: Front-End (FE), MME Service Logic (SL) and State DataBase (SDB). FE acts as the communication interface with other entities of the network and balances the load among several MME SL instances that are in charge of processing control messages. SDB stores the user session state, hence enabling stateless SL.

4.1.2. Adaptive Scaling Mechanism - Overview

The proposed mechanism aims at maintaining the MRT of the EPC control plane less than a particular threshold (*e.g.,* 1 *ms* (Prados-Garzon et al., 2017b)). To satisfy such aim, the proposed mechanism can make scaling-out/in of vMME. In this sense, the infrastructure can instantiate up to K instances (*e.g.,* $K = 4$) of vMME, and measure the offered workload and MRT.

The adaptive scaling mechanism is based on RL (Sutton and Barto, 2012), a sub-field of machine learning, where an agent learns a decision-making process by interacting with an environment. From Markov Decision Processes (Mausam and Kolobov, 2012), the agent and environment interact at discrete time steps. At each time step t, the agent receives some representation of the state of the environment, $S_t \in S$, where S is the set of possible states. Based on S_t, the agent selects an action, $A_t \in A$, where A is the set of available actions in the state S_t. One step later, the agent receives a numerical reward, $R_{t+1} \in \mathbb{R}$, and finds itself a new state, S_{t+1}.

Fig. **4-2** depicts the RL process instantiated to the scaling of the EPC control plane, particularly the vMME; the environment is the NS of signaling in EPC, the states are pairs formed by the current number of instances and performance, and actions refer to the number of available instances. The steps, labeled as *1)* observe the state, *2)* take an action, and *3)* receive a reward, refer to a direct RL process called Policy Evaluation Process, which means that the agent interacts directly with the environment.

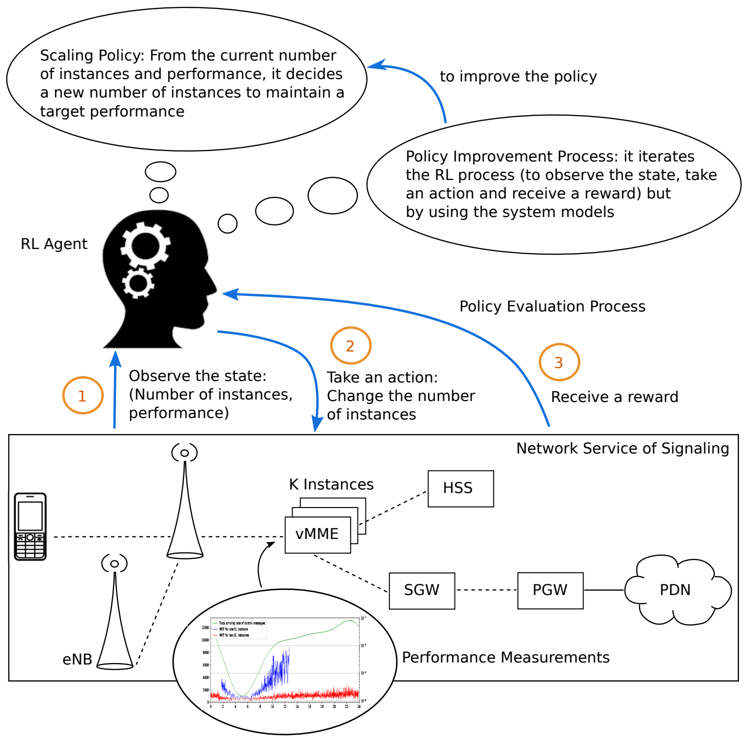

Scaling Policy: From the current number of instances and performance, it decides a new number of instances to maintain a target performance
to improve the policy
Policy Improvement Process: it iterates the RL process (to observe the state, take an action and receive a reward) but by using the system models
RL Agent
Policy Evaluation Process
1
Observe the state: (Number of instances, performance)
2
Take an action: Change the number of instances
3
Receive a reward
Network Service of Signaling
K Instances
vMME
HSS
SGW
PGW
PDN
eNB
Performance Measurements

Figure 4-2.: Overview of the adaptive scaling mechanism.

Reward calculations consider the performance target (*e.g.,* MRT less than 1 ms) and the utilization factor $\rho = \lambda/(k \cdot \mu) < 1$ (Zukerman, 2016) of vMME, where λ is the workload, k is the number of instances and μ is the service rate of an instance. ρ must be less than 1 to guarantee stability and allows defining expected ranges of utilization to be supported by vMME, which are $\frac{k-1}{k} \leq \rho < 1$. Given these ranges, the reward function is

$$R(k, \lambda) = \begin{cases} +1, & t_r < 1\ ms\ \wedge\ \frac{k-1}{k} \leq \rho < 1 \\ -1, & in\ other\ case \end{cases} \tag{4-1}$$

The scaling mechanism uses Q-Learning (Watkins, 1989) as a method to perform RL. In this method, to each pair state-action is assigned an action value, which is the expected utility of carrying out an action A_t when the agent is in the state S_t and follows the most optimal policy. A policy is a rule for selecting actions. The value function is represented by $Q(S_t, A_t)$; and Q, implicitly defines the current policy f:

$$f_t(S_t) = a,\ such\ that\ Q_t\ (S_t, A_t) = \underset{A}{max} Q_t(S_t, A_t) \tag{4-2}$$

f_t and Q_t correspond to the policy and values of Q at time t, respectively. That is, the current policy consists of choosing the action with maximal estimated value. The agent, through its experience, adjusts the values of Q according to:

$$Q_{t+1}(S_t, A_t) \leftarrow (1 - \alpha)Q_t(S_t, A_t) + \alpha(R_{t+1} + \gamma \underset{A}{max} Q_t(S_{t+1}, A)) \tag{4-3}$$

where R_{t+1} denotes the reward received at step $t + 1$, α is the *learning factor* (a small positive number) that allows the agent to retain what has been learned, and $0 \leq \gamma \leq 1$ is the discount factor that determines the importance of future rewards.

For learning, the agent needs to explore the environment and carry out the trial and error process; but, wrong actions lead to unnecessary procedures of provisioning and releasing of resources that must be avoided. It is here, where the system models are used. In this sense, for instance, in the EPC context, signaling workload is considered as the input and MRT as the output of a dynamic system that is modeled by a regression function h. By observing these metrics, h is trained for predicting values of workload and use such predictions to run hypothetical iterations of the RL process, which allows getting the optimal policy before applying it to a particular NS. This is the Policy Improvement Process (see Fig. **4-2**).

In summary, the agent runs two processes: the policy improvement and the policy evaluation; the first one allows the agent to foresee the results of its action, and the second one is the current execution of the policy. If there is no variation in the conditions of the NS (*e.g.,* number the users), which is reflected in the corresponding MRT, the agent probably just needs one iteration (to observe, take action and receive a reward) for improving its policy.

But, if variations happen, the agent will need more iterations in the policy improvement to adapt to new conditions.

4.1.3. System Modeling

An NS is considered as a dynamic system. For this system, a function h is estimated given data $\mathcal{D}$: $\mathbf{x}_i \in \mathbb{R}^D$ (input) and $y_i = h(\mathbf{x}_i) + \varepsilon_i \in \mathbb{R}$ (output); the term $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is independent Gaussian measurement noise, which considers variations of y_i in relation to the values of h. The estimation of h refers to a regression problem that can be approached by parametric and non-parametric models. Parametric models impose a fixed structure on h which limits its representational power. Non-parametric models allow determining the shape of the underlying function h from the data and assumptions about its smoothness. Note that the term non-parametric does not imply models without parameters, but that the number of parameters is flexible and grows with the sample size.

Modeling uses regression based on GPs (Deisenroth, 2010) that combines non-parametric models with Bayesian modeling and inference. A GP is fully specified by a mean function $m_h(\cdot)$ that describes how the average function is expected to look, and a covariance function which is also called a kernel.

$$\begin{aligned} k_h(\mathbf{x}, \mathbf{x}') &= \mathbb{E}_h[(h(\mathbf{x}) - m_h(\mathbf{x}))(h(\mathbf{x}') - m_h(\mathbf{x}'))] \\ &= cov_h[h(\mathbf{x}), h(\mathbf{x}')] \end{aligned} \tag{4-4}$$

This function (kernel) specifies the covariance between any two function values. Here, $\mathbb{E}_h$ is the expected value with respect to the function h.

Modeling considers a Radial Basis Function (RBF) kernel (Álvarez et al., 2012) and a mean function $m_h = 0$. The RBF kernel, also known as the squared exponential kernel, has the form

$$k_h(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}, \mathbf{x}'\|^2}{2\ell^2}} \tag{4-5}$$

where $||\cdot||$ represents the Euclidean norm and ℓ is the characteristic length-scale, which is a hyper-parameter that describes how smooth the function h is; a small length-scale value means that values of h can change quickly, while a large value characterizes h that changes slowly.

Considering h as a random function, Bayesian inference allows inferring a posterior distribution $p(h|\mathcal{D})$ over h from the GP prior $p(h)$, the data $\mathcal{D}$ and assumptions on the smoothness of h. The posterior is used to predict $h(\mathbf{x}_*)$ values at arbitrary inputs $\mathbf{x}_* \in \mathbb{R}^D$. Briefly, Bayesian inference has three steps: (i) a prior on the unknown quantity has to be specified (in this case, h), (ii) data are observed; and (iii) a posterior distribution over h is computed that refines the prior by incorporating evidence from the observations.

For instance, consider data $\mathcal{D}$ of workload and MRT, which are represented in vectors $\mathbf{x}, y \in \mathbb{R}$, respectively:

$\mathbf{x} = [1000, 1200, 4600, 6400, 8200, 10000]^T$,
$y = [0.0001, 0.0002, 0.00015, 0.0002, 0.00035, 0.002]^T$;

$\mathbf{x}$ is measured in service requests per seconds and y in seconds. Fig. **4-3** (left) plots samples from the GP prior; the prior uncertainty about h is constant (gray area) because there no observations. After having observed six function values (the data $\mathcal{D}$) represented by small circles in Fig. **4-3** (right), samples from GP posterior depict that the uncertainty varies and depends on the location of the training inputs. This example has chosen a length-scale ℓ of 3000, which allows to have the smoothness of h in Fig. **4-3**.

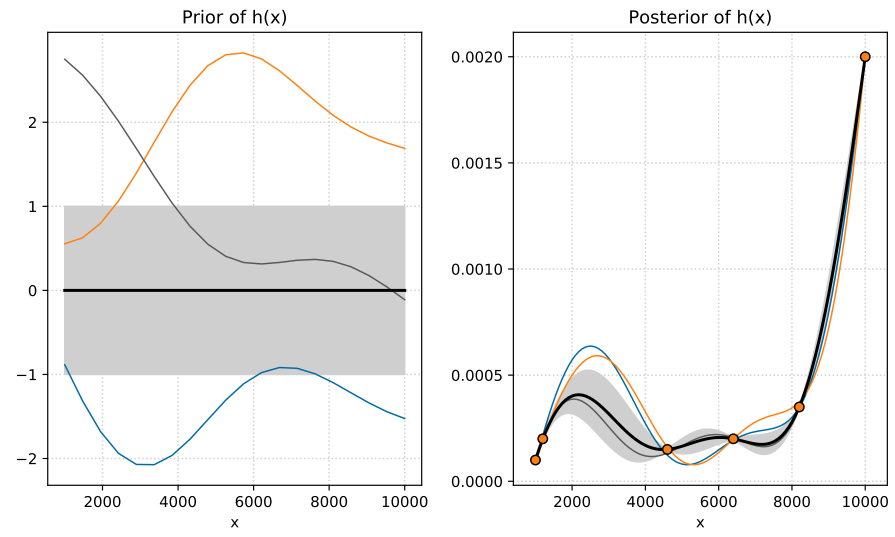


Figure 4-3.: Prior and posterior of $h(x)$.

4.1.4. Scaling Processes

By using pseudo-codes, this subsection details the processes of policy improvement and policy evaluation.

```
Data: GP-based system models, Q-Learning parameters (e.g., $\gamma$ and $\alpha$), a reward
      function (e.g., see Equation 4-1) and a learning threshold (e.g., $\varepsilon_l$)
Result: An improved value function (i.e., Q) which is used by Algorithm 3
for each $t$ do
    Initialize a variable $error$ to a value greater than $\varepsilon_l$; this variable allows
      finishing this iterative process;
    while $error > \varepsilon_l$ do
        Store $Q$ before the RL process in $previousQ$;
        Get the scaling action $a$ using Equation (4-2);
        Estimate $S_{t+1}$ using GPs-based system models;
        Calculate the reward using Equation (4-1);
        Update $Q$ using Equation (5-8);
        Replace $S_t$ with $S_{t+1}$;
        Store $Q$ after the RL process in $finalQ$;
        Calculate the $error$ using the Mean Squared Error (Equation 4-6);
    end
end
```

Algorithm 2: Policy improvement.

The Mean Squared Error (MSE) mentioned in the previous algorithm is given by

$$MSE = \frac{1}{N}\sum_{i=1}^{N} e_i^2 \tag{4-6}$$

where $e_i = previousQ - finalQ$ is the error between the value function before and after an iteration, and N is the number of elements of Q.

4.2. Evaluation and Analysis

By simulations, this chapter evaluates the GPs-based system modeling and the behavior of the proposed adaptive scaling mechanism. Also, this chapter compares it with other ones: *(i)* based on static threshold rules, and *(ii)* based on Q-Learning without system models for improving the policy.

4.2.1. System Modeling

Evaluation generates a synthetic workload and gather data of MRT from a simulated vMME. The synthetic workload is generated using the expressions given in (Wang et al., 2015), which

```
Data: An improved $Q$ as result of Algorithm 2, Q-Learning parameters (e.g., $\gamma$
      and $\alpha$) and a reward function (e.g., see Equation 4-1)
Result: A scaling-out/in action transferred to vMME
Since the agent receives its reward after that it takes action; this algorithm runs in
 two steps;
for each $t$ do
    Measure and store the current state in $S_t$;
    Get the scaling action $a$ using Equation (4-2);
    Modify the number of instances of vMME according to the action $a$;
end
for each $t+1$ do
    Measure and store the current state in $S_{t+1}$;
    Calculated the reward using Equation (4-1);
    Update $Q$ using equation (5-8);
end
```

Algorithm 3: Policy evaluation.

considers that traffic in a mobile network exhibits a spatial-temporal pattern (Xu et al., 2017). To measure MRT, evaluation simulates the vMME of Fig. **4-1** as a queuing model. Simulations followed a discrete event process implemented in Python (Team-SimPy). As service rates, evaluation uses the defined in (Prados-Garzon et al., 2017b): 120.000 packets per second for FE, 10.167 control procedures per second for MME SL and 100.000 transactions per second for SDB. Also, evaluation considers a vMME composed by up to four SL instances; hence, four GPs-based models are needed for building the regression models of the vMME, one for each scaling configuration. Evaluation uses scikit-learn (Pedregosa et al., 2011) for creating GPs, tuning their hyper-parameters and performing predictions.

Fig. **4-4** plots the signaling workload and some samples of MRT gathered from the simulated vMME (one and four instances for clarity), which are label as measured MRT. Also, Fig. **4-4** plots MRT estimated from the models. Note that there is a good accuracy between measured and estimated MRT, which is confirmed by quantifying the MSE of the predictions (Equation 4-7): $8.6 \cdot 10^{-7}$ for one SL instance and $5.0 \cdot 10^{-7}$ for four SL instances.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} e_i^2 \tag{4-7}$$

where $e_i = measured\ MRT - estimated\ MRT$ is the prediction error, and N is the number of samples.

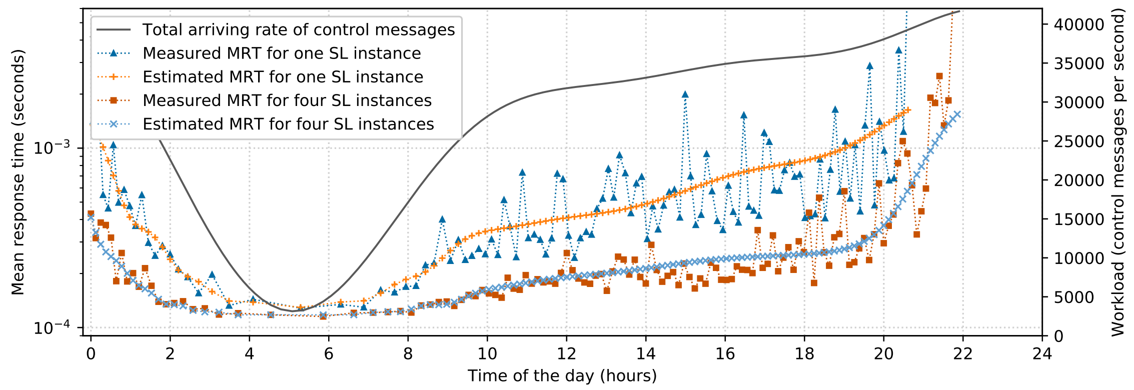


Figure 4-4.: Data samples of workload, and measured and estimated mean response time.

4.2.2. Adaptive Scaling Behavior

The operation of the scaling mechanism was simulated over an overall time of 24 hours, using as signaling workload the total arrival rate of control messages given by Fig. **4-4**. Also, the vMME of Fig. **4-1** was simulated as a queuing model, and the system models based on GPs, built previously, were used. The simulation followed a discrete-events process in which by each time step the Policy Improvement (Algorithm 2) and Policy Evaluation (Algorithm 3) were executed. The time step was 10 minutes, which allows achieving a right level of granularity.

In the simulations, γ is 0.8 because its value close to 1 allows both the agent to consider future rewards and the expected reward can converge. In turn, α is 0.1 because this small value enable the agent to retain the learning. A learning threshold $\varepsilon_l = 10^{-3}$ is a small enough difference between Q before and after an iteration of the policy improvement algorithm, which allows the loop to terminate.

Fig. **4-5** presents the simulation results by plotting, in different scales, MRT and the number of SL instances vs time. These results reveal that MRT is smaller than its maximum allowed value (1 ms) all time, which corroborates the expected accuracy of the mechanism to determine the correct number of instances at the right time. The changes of the number of SL instances are performed at 0:40 AM (from four to three instances), 1:50 AM (from three to two instances), 3:10 AM (from two to one instance), 7:00 AM (from one to two instances), 9:00 AM (from two to three instances) and 6:07 PM (from three to four instances).

Consider some internal details. Fig. **4-6** illustrates the operation of the scaling mechanism by plotting the number of iterations per execution of the Policy Improvement (Algorithm 1). It can be noted that at the beginning, the policy improvement algorithm performs 160 iterations, which enables to the agent getting the initial policy. At times of change (0:40 AM, 1:50 AM, and so on), about 200 iterations are needed to improve policy because it

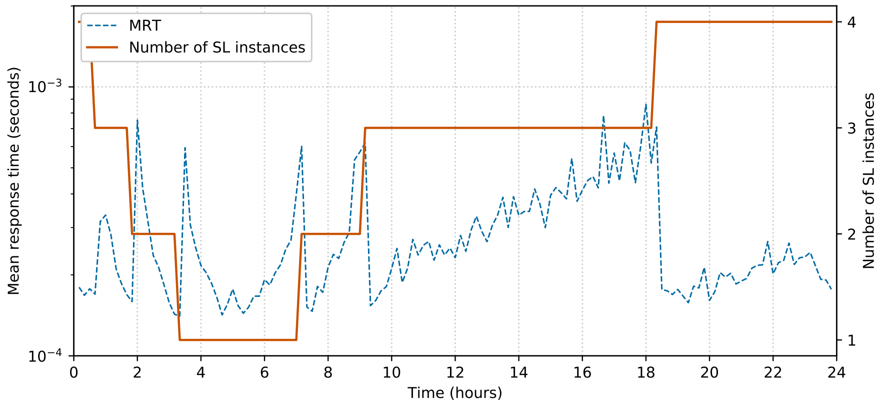


Figure 4-5.: Adaptive scaling mechanism based on Q-Learning and system models.

is the border between two states, but in the other times only one iteration is required. In summary, the proposed mechanism adapts to changes in the environment and learns by using the GPs-based models. This strategy allows applying scaling actions to vMME only when the agent reaches an improved policy.

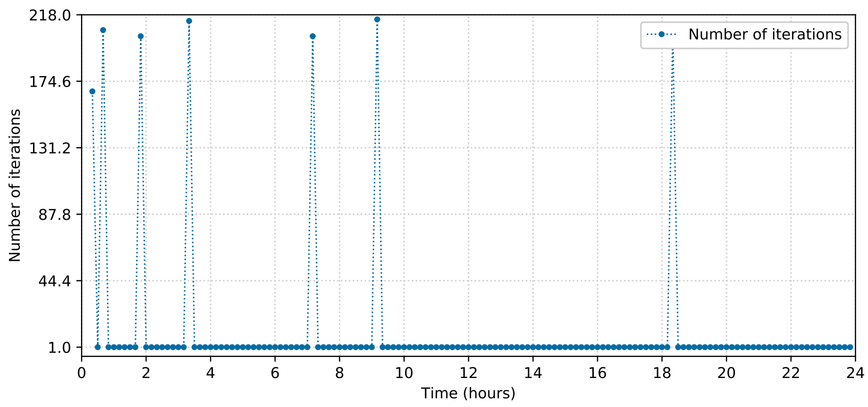


Figure 4-6.: Number of iterations in the policy improvement.

4.2.3. Comparison

Fig. **4-7** presents the behavior of a mechanism that uses static threshold rules. For the shake of comparison, it is used as rules those defined in the reward function (Equation 4-1). Note that this approach is accurate in the time to scale because of its reactive characteristic, however, when variations cross the thresholds several times, transitory oscillations happen, such as the occurred close to 2:00 AM. Also, non-performance target compliances happen when a threshold is crossed. In short, the proposed mechanism is better than mechanisms based on static threshold rules because it avoids transitory oscillations and violations of the performance target.

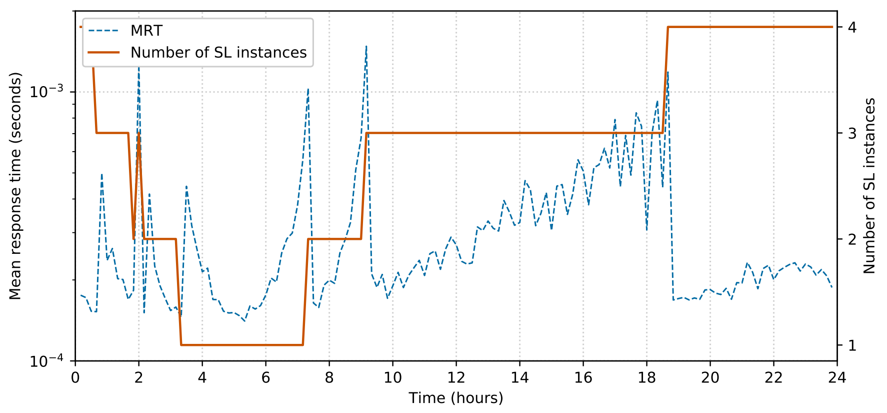


Figure 4-7.: Mechanism based on static threshold rules.

Fig. **4-8** depicts the behavior of a mechanism that uses Q-Learning without system models for policy improvement. It is maintained the same conditions that in the proposed mechanism, this means, the parameters $\gamma = 0.8$ and $\alpha = 0.1$. Note that a decision to scale, from two to one SL instance, is taken around 4:00 AM, fifty minutes after the proposed mechanism; this delay is caused by the small α value, which retains the previous learning of Q-Learning. A similar situation is observed between 7:00 AM and 9:00 AM. These delays cause that MRT exceeds its maximum value (1 ms). Other point to highlight is the transitory scaling oscillations that the agent makes when the vMME needs to be scaled, such as at 3:00 AM and 4:00 AM. These oscillations are because the agent tries wrong actions until it can achieve a good policy. In brief, the proposed scaling mechanism is better than the based on Q-Learning without models for policy improvement. It is corroborated by the time when the scaling happens, the correct number of instances selected and the performance target compliance.

To sum up, simulations confirm the expected accuracy of the scaling mechanism. At each time step, the Q-Learning agent uses the system models for improving its scaling policy, and next, taking the best action based on that improved policy.

4.3. Final Remarks

This chapter presented an adaptive mechanism that learns a scaling policy for managing network performance variations aiming at being accurate in the time for scaling and the correct number of instances to increase or decrease. The mechanism combines Q-Learning with GPs-based system models that allow it to adapt to dynamic environments and improve

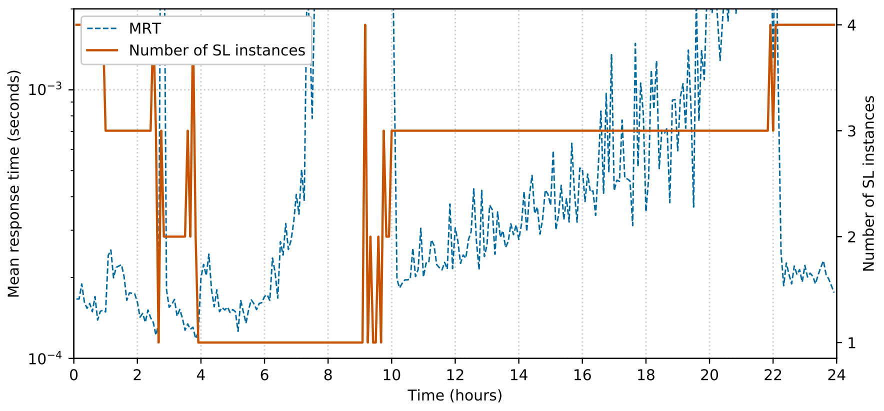


Figure 4-8.: Mechanism based on direct Q-Learning.

its scaling policy before taking any action. Also, this chapter corroborated by simulations that the mechanism is more accurate than mechanisms based on static threshold rules, which are widely used, and Q-Learning without system models for improving its policy.

5. Cooperative Scaling for the 5G Core Network Control Plane

The 5GCN is based on NFV that aims the deployment of NSs flexibly and dynamically (Chiosi et al., 2012, Han et al., 2015), enabling scalability and adaptability of the 5G network infrastructure according to user and application needs (Chowdhury, 2021). The 5GCN control plane follows a service-based architecture (3GPP, 2018c), in which NFs, such as AMF, SMF, and NRF interact to perform signaling procedures (*e.g.,* registration of UE and handover) (3GPP, 2018d). Scaling the control plane is an essential feature in 5GCN since it enables NSs to dynamically adapt to workload variations raised by the increasing number of users and demand of applications. Moreover, NFV-based scaling brings cost reduction compared to traditional solutions of scaling, which statically oversize the capacity of NFs for the highest predictable workload peak (Adamuz-Hinojosa et al., 2018). Due to NSs in 5GCN control plane are end-to-end functionalities formed by composing NFs (Bhamare et al., 2016), scaling NSs implies to scale its constitutive NFs, and can mainly be done by increasing/decreasing the number of instances (horizontal scaling) or the number of resources allocated to them (vertical scaling) (ETSI, 2014a, Balla et al., 2020).

Solutions for scaling the 5GCN control plane, such as (Alawe et al., 2018a,b), use the independent scaling approach, assuming that NFs (*e.g.,* the AMF) operate in isolation, which is unaware of the interdependent nature of NFs that compose NSs. The same assumption of NFs operating in isolation underlies solutions in the Telco cloud, such as (Tang et al., 2015, Carella et al., 2016, Dutta et al., 2016). A joint scaling of the control plane in the 4G core is addressed in (Prados-Garzon et al., 2018) by using an open network of G/G/m queues. However, this solution may not be suitable for 5GCN control plane due to its particular modeling for the 4G core network. The work (Harutyunyan et al., 2021) used ILP to analyze the trade-offs between vertical, horizontal, and hybrid scaling in 5GCN and MEC. This work classifies NFs into stateful, control plane, and user plane NFs, and does not model 5GCN control plane NFs as interacting entities.

Considering the above drawbacks, it is remarked that scaling of the control plane in 5GCN should be done from a cooperative approach. NFs of 5GCN control plane perform in synchronous mode, meaning that they are interdependent, and so, scaling one NF leads to change performance conditions of the other ones. In addition, scaling an NF in isolation assumes that the other NFs have been statically over-sized, which does that the traditional static dimensioning problem remains. Furthermore, independent scaling takes into account only a partial view of the behavior of NSs and may result in that quality requirements cannot be maintained most of the time.

This chapter proposes Coop-Scaling, a novel scaling mechanism for the control plane of 5GCN in a cooperative way. Coop-Scaling uses MARL to carry out cooperative scaling that allows learning a scaling policy maintaining the average response time below a target. Coop-Scaling is formed by software modules that enable agents to learn over an environment (*e.g.,* NSs of the 5GCN control plane) by using the RL approach. By simulations, Coop-Scaling is evaluated regarding successful episodes, average response time, and capacity utilization. The evaluation results corroborate that Coop-Scaling outperforms scaling based on a single-agent or independent agents. To sum up, the contributions of this chapter are:

- A cooperative scaling mechanism (called Coop-Scaling) based on MARL for the control plane of 5GCN.
- A prototype of Coop-Scaling.
- An evaluation of Coop-Scaling, corroborating that cooperative scaling outperforms scaling based on single-agent or independent agents regarding the average response time.

The rest of the chapter is structured as follows: Section 5.1 specifies Coop-Scaling describing its modules. Section 5.2 gives the simulation process followed in the evaluation and compares the performance of Coop-Scaling and non-cooperative scaling approaches. Section 5.3 gives some conclusions.

5.1. Cooperative Scaling based on Multi-Agent Reinforcement Learning

This section introduces Coop-Scaling presenting, first, an overview, and second, its architecture.

5.1.1. Overview

Coop-Scaling aims at managing workload variations in 5GCN by selecting the most appropriate scaling configuration of its NFs. To achieve this goal, Coop-Scaling carries out three tasks involving a making-decision process. Tasks of Coop-Scaling are iterative and form a scaling control loop, which is depicted in Fig. **5-1**, involving states and transitions between them. First, Coop-Scaling collects data over traffic and performance metrics, such as the number of users and latency. Second, Coop-Scaling estimates scaling actions that maintain workload in a desirable range. Third, Coop-Scaling applies the scaling actions to 5GCN updating its states. The control loop of Fig. **5-1** will be implemented by scalability multi-agents that learn from experience using the Q-Learning method.

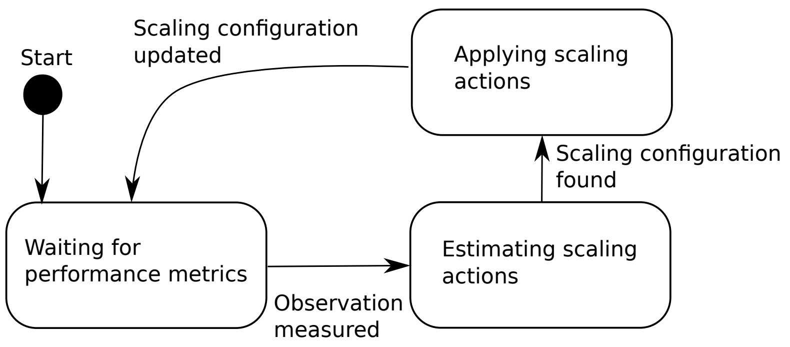

Start
Scaling configuration updated
Applying scaling actions
Scaling configuration found
Waiting for performance metrics
Estimating scaling actions
Observation measured

Figure 5-1.: Control loop for scalability.

5.1.2. Coop-Scaling Architecture

Fig. **5-2** depicts the architecture of Coop-Scaling that operates over the 5GCN control plane. In this plane, NFs such as AMF, SMF, and NRF can scale horizontally to support varying signaling traffic due to the increase in the number of users. Coop-Scaling manages such as traffic variations as part of a cognitive management plane, implementing the control loop of Fig. **5-1**. The following paragraphs describe the 5GCN control plane and the subsystems of the cognitive management.

5GCN Control Plane. This plane represents complete NSs that implement signaling procedures in 5GCN. The overall capacity of this plane depends on the current number of users demanding communication services and the capacity of NFs. Hence, scaling NFs can modify the 5GCN control plane elastically to support varying traffic conditions. The entities of the 5GCN control plane are presented below.

Users and their User Equipment. 5G mobile networks expect to provide communication services to a huge number of users (human and machine type). 5G usage scenarios (ITU, 2015) are a broad classification of application types, such as eMBB, massive Machine Type Communications (mMTC), and URLLC, which require the control plane of 5GCN to support signaling traffic.

Human users use their UE to request communication services and use applications through the evolved Node B (not shown in Fig. **5-2**), which is also valid for non-human applications such as IoT and Vehicular Communications. Therefore, for the control plane of 5GCN, users and signaling traffic generated by UE represent the workload.

Scaled Network Service. The control plane of 5GCN supports signaling procedures served by NSs (3GPP, 2018d). These procedures are, for example, registration, session establishment, and mobility management. Registration management allows registering or deregistering a UE (User) with 5GCN and establishing its user context. Mobility management allows keeping track of the current location of a UE. A UE needs to register with the network to get authorized to receive services, to enable mobility tracking and reachability. In this Chapter, the NS for session establishment is used to illustrate the operation of Coop-Scaling. Each NS is implemented by composing NFs, which scale to handle variations in the number of UE that require services to the 5G network.

In addition to NFs, the 5GCN control plane needs distributed monitoring agents, which forward scaling states, workload data, and performance metrics to Coop-Scaling. Monitoring agents can be deployed into NF managers in an NFV-based 5GCN control plane.

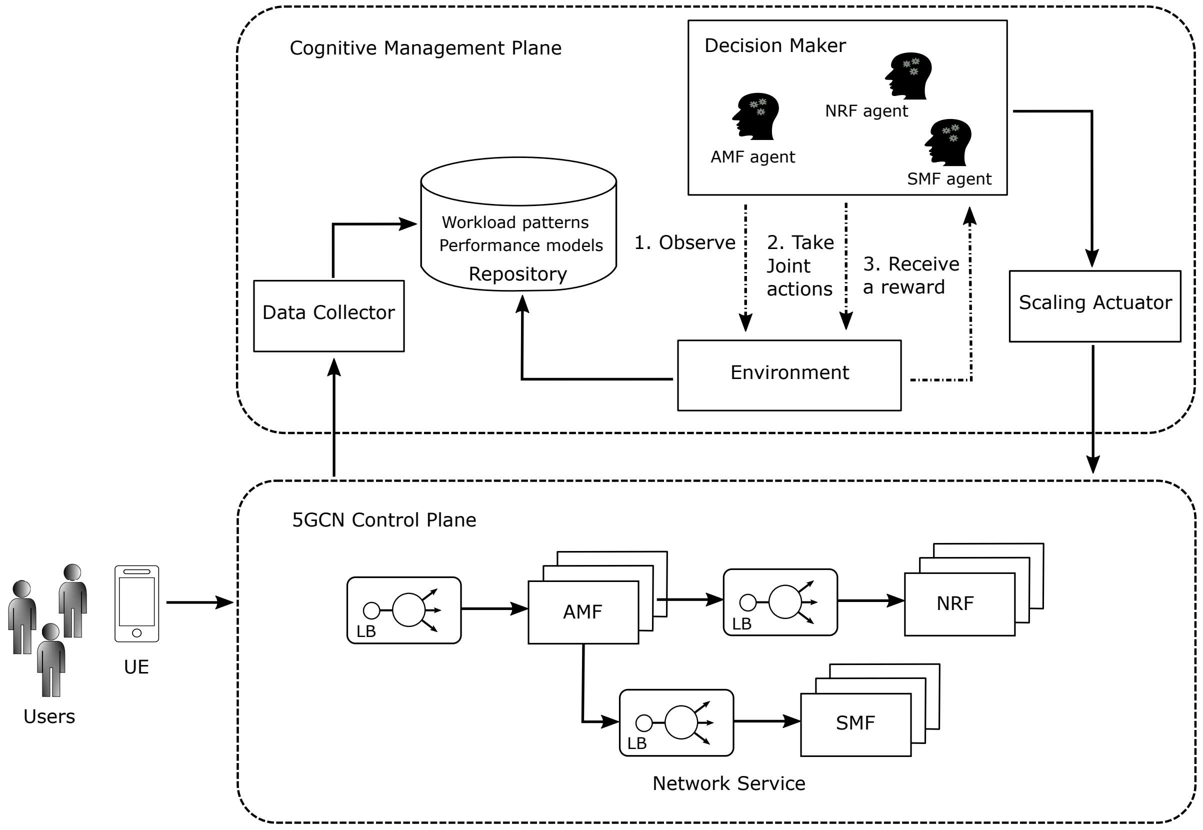


Figure 5-2.: Coop-Scaling architecture.

Cognitive Management Plane. This plane implements the control loop for scalability given in Fig. **5-1**, in which management agents cooperate to make scaling decisions. The following paragraphs describe the modules that compose the cognitive management plane.

Data Collector. This module collects workload and performance metrics and stores them into a repository to create workload patterns and performance models. Workload patterns and performance models define the environment that agents interact with. Performance metrics are average response time and capacity utilization, which characterize the behavior of the mentioned NS and that usually vary depending on the workload. The average response time is the expected time that the AMF takes to respond requests of session establishment by UE. While the capacity utilization indicates the proportion that an NF (*e.g.,* AMF) instance is busy processing messages. For example, 70% of utilization of an AMF instance means that 70% of the time it is working, while 30% of the time it is idle expecting for requests by UE.

Repository. This module stores workload patterns and performance models. This chapter considers as workload a daily pattern representing users (*e.g.,* UE) needing to establish sessions. Performance models can be constructed from data sets collected from measures, building black-box systems with inputs, controls, and outputs. For instance, system inputs can be the number of users, system controls can be scaling actions of NFs, and system outputs can be performance metrics estimated that depend on current inputs and the applied controls. Instead of selecting this black-box approach to systems performance modeling, this thesis uses theoretical models, particularly PEPA-based models, for evaluating performance of concurrent systems (Hillston, 1994). However, theoretical models require real data to specify parameters, such as service rates that define the average times spent by an NF to complete its tasks.

In this chapter, Equation 5-1 is used to define a daily pattern of workload, which represents the spatial-temporal variation of the traffic in a mobile network (Xu et al., 2017). Fig. **5-3** plots this workload, showing the number of UE versus the time of the day. It is to highlight that $ue(t)$ has an stochastic behavior defined by the term $U[0, b)$, which adds a random number with uniform distribution between 0 and b.

$$ue(t) = \frac{a_0}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) + U[0, b) \tag{5-1}$$

As depicted in Fig. **5-2**, the NS for session establishment is composed by AMF, SMF and NRF. AMF is the entry point for session establishment requests from UE. SMF creates a session management context that allows UE to use 5G applications. NRF performs the discovery of available SMF instances. Additionally, each NF has a load balancer (LB) that distributes the incoming requests to the available NF instances. It is assumed that LBs operate at high rates, and their effect on performance is negligible. Therefore, they will not

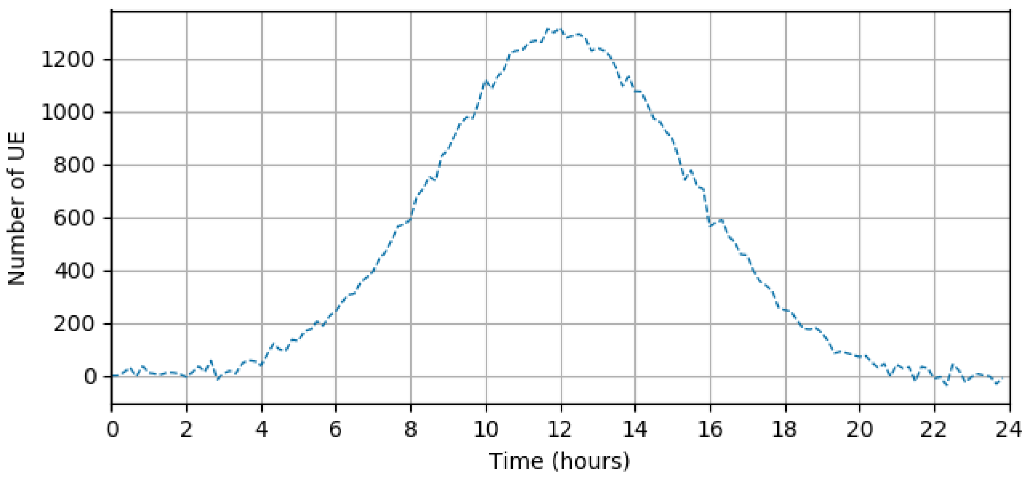


Figure 5-3.: A daily pattern of workload.

include in the modeling. The following sequential components form the PEPA models of the NS for session establishment.

Component *UE* is modeled as a sequential component which interacts with AMF and requests for session establishment at a rate r_{ue}. Between successive requests, UE interposes some idle time defined by r_{idle}.

$$UEIdle \overset{def}{=} (idle, r_{idle}).SessEstablishment$$

$$SessEstablishment \overset{def}{=} (session\_establishment, r_{ue}).UEIdle$$

Component *AMF* accepts and processes session establishment requests from UE at a rate r_p. Then, AMF discovers an available SMF instance and requests it for the session management context creation. Finally, the AMF replies to UE once the session establishment procedures finish. An instance of AMF is modeled as follows.

$$AMF \overset{def}{=} (session\_establishment, r_p).NFDiscovery$$

$$NFDiscovery \overset{def}{=} (discovery, r_{req}).SMCreation$$

$$SMCreation \overset{def}{=} (smc\_creation, r_{req}).SEReply$$

$$SEReply \overset{def}{=} (session\_establishment\_reply, r_{reply}).AMF$$

Component *NRF* exposes the operation *discovery*. Using this operation, an AMF finds the URL of an available SMF instance. An NRF instance has a sequential component corresponding to the discovery operation carried out at a rate r_{req}, which is followed by a local activity performed at a rate r_d.

$$NRF \overset{def}{=} (discovery, r_d).NRFLocal$$

$$NRFLocal \overset{def}{=} (local_{nrf}, r_d).NRF$$

Component *SMF* offers an operation for creating a session management context that allows UE to use applications. Thus, a SMF instance is modeled as a sequential component corresponding to the creation of a session management context carried out at a rate r_{req}, which is followed by a local activity performed at a rate r_{smc}.

$$\begin{aligned} SMF &\overset{def}{=} (smc\_creation, r_{smc}).SMFLocal \\ SMFLocal &\overset{def}{=} (local_{smf}, r_{smc}).SMF \end{aligned}$$

The system equation for the session establishment procedure can be written as follows.

$$\begin{aligned} &UEIdle[N_{UE}] \underset{\{session\_establishment\}}{\bowtie} ((AMF[N_{AMF}] \\ &\underset{\{discovery\}}{\bowtie} NRF[N_{NRF}]) \underset{\{smc_creation\}}{\bowtie} SMF[N_{SMF}]) \end{aligned} \tag{5-2}$$

Where N_{UE} is the number of UE, and N_{AMF}, N_{NRF}, and N_{SMF} are the number of available instances of AMF, NRF, and SMF, respectively.

Environment. In Coop-Scaling, the environment is a representation of the NS (see Fig. **5-2**). By using the workload patterns and performance models, the environment and representation of states (S) of such environment are observations as follows.

$$S = \{t_i, ue_i, \rho_{[amf]i}, \rho_{[smf]i}, \rho_{[nrf]i}, art_i, nfconf_i\} \tag{5-3}$$

Where t_i is the time of sample i; ue_i is the workload at that sample; $\rho_{[amf]i}$, $\rho_{[smf]i}$, and $\rho_{[nrf]i}$ are the capacity utilization experimented by AMF, SMF, and NRF at sample i, respectively; art_i is the average response time at sample i, and $nfconf_i$ is the scaling configuration in the NS at sample i.

In RL, the reward (R) stipulates what an agent must to accomplish. In Coop-Scaling, the average response time and capacity utilization of NFs are used to define three types of rewards, which in turn define three types of environments because different rewards change the state space defined by Equation 5-3. In addition, these environments imply a certain degree of cooperation even among independent agents.

1. *The environment gives rewards according to QoS metrics of average response time.* The reward is +1 if the average response time is below a target, whereas reward is 0 in other cases. R1 (Equation 5-4) defines this environment and requires that, for example, a fourth agent communicates the average response time metric.

$$R_1 = \begin{cases} 1, & art < art_{maximum} \\ 0, & in\ other\ case \end{cases} \tag{5-4}$$

2. *The environment gives rewards according to shared QoS metrics of capacity utilization.* The reward is -1 if any NF reaches 100% of its utilization capacity, whereas reward is the value of the average capacity utilization in other cases. R2 (Equation 5-5) defines this environment. Unlike the previous one, this environment requires that the three agents communicate their capacity utilization to calculate the average utilization. In Equation 5-5, $(1/3)\sum \rho_{[nfs]i}$ is the average capacity utilization for the NS given in Fig. **5-2**. In this environment, the own QoS metric and QoS metrics of the other agents encourage learning.

$$R_2 = \begin{cases} -1, & \rho_{[nfs]i} = 1 \\ (1/3)\sum \rho_{[nfs]i}, & 0 \leq \rho_{[nfs]i} < 1 \end{cases} \tag{5-5}$$

3. *The environment gives rewards according to QoS metrics of average response time and capacity utilization.* The reward defined by R3 (Equation 5-6) is the combination of the above cases. In this environment, agents have more QoS data, such as their capacity utilization, shared capacity utilization, and average response time. In Equation 5-6, $(1/3)\sum \rho_{[nfs]i}$ is the average capacity utilization for the NS given in Fig. **5-2**. More information allows them to make better decisions.

$$R_3 = \begin{cases} 1 + (1/3)\sum \rho_{[nfs]i}, & art < art_{maximum},\ and \\ & 0 \leq \rho_{[nfs]i} < 1 \\ -1, & \rho_{[nfs]i} = 1 \end{cases} \tag{5-6}$$

Decision Maker. This module is modeled as a multi-agent system (see Fig. **5-2**) in which each NF has an agent that can carry out actions of horizontal scaling (*e.g.,* stay equal, increase, and decrease service instances). Three types of agents are analyzed.

1. A single AMF agent that can scale, while SMF and NRF are provisioned statically with a determined number of service instances.

2. Independent agents corresponding to AMF, SMF, and NRF that can share QoS metrics about capacity utilization and average response time but carry out actions independently.

3. Cooperative agents that carry out actions jointly.

In the three previous types of agents, the Q-Learning algorithm is used. The learned decision policy of each agent is determined by the value function, $Q(s, a)$, which estimates long-term discounted rewards for each pair $(state, action)$.

An essential part of Q-Learning in finding an optimal policy is the selection of action a_t in state s_t. In Q-Learning, there exists a trade-off between selecting the currently expected optimal action, or selecting a different action in the hope it will yield a higher cumulative reward in the future (Tijsma et al., 2016). Coop-Scaling uses the softmax exploration (Barto et al., 1991) that converts state-actions values into actions probabilities using a Boltzmann distribution (Equation 5-7). Given a current state s and available actions a_i, a Q-Learning agent selects each action a with a probability given by that distribution.

$$p(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum\limits_{k \in actions} e^{Q(s,a_k)/T}} \tag{5-7}$$

Where T is the temperature parameter that adjusts the randomness of decisions. Low temperatures lead to greedy action selection with regards to Q, whereas high temperatures cause all actions to have more similar chances of being chosen. The probability of transition p gives a balance between exploration and exploitation of the Q-Learning agent.

Once an agent chooses an action, it then executes the action, receives an immediate reward r, and moves to the next state s'. In Coop-Scaling, executing an action means to change the scaling configuration. For example, if in the current step t, the configuration is $(AMF, SMF, NRF) = (1, 1, 1)$ (*i.e.,* each NF agent has one service instance), and the action chosen by the AMF agent is *increase*, the next configuration in the step $t + 1$ will be $(AMF, SMF, NRF) = (2, 1, 1)$.

In each time step, the agent updates $Q(s, a)$ by recursively discounting future utilities and weighting them by a positive learning rate α. The agent, through its experience, adjusts the values of the Q-table according to Equation 5-8:

$$Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha)Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \underset{A}{max} Q_t(s_{t+1}, A)) \tag{5-8}$$

Where r_{t+1} denotes the reward received at step $t + 1$, α is a small positive number called learning factor that allows the agent to retain what has been learned, and $0 \leq \gamma \leq 1$ is the discount factor that determines the importance of future rewards. Q-Learning is popular because of its simplicity, and it can be used for scaling NFs as is proved in (Tang et al., 2015, Tobar et al., 2017).

The dimension of Q-table depends on the number of states defined in the environment. Thus, for a single AMF agent, and independent agents that consider n workload samples, the Q-table is of dimension n rows by 3 columns. The number of columns is given by the three actions that each agent can take. On the other hand, for agents taking actions jointly, the dimension of the Q-table is n rows by 3^M columns, where M is the number of different NFs. For the example outlined with three NFs, the dimension is n rows by 27 columns. For a single AMF agent and independent agents, each one has a Q-table that allows it to select the optimal action independently. However, for agents taking action jointly, there is only one Q-table; hence, it has a larger dimension.

Scaling Actuator. This module executes the scaling policies into 5GCN control plane allowing to update the states of the network service handled.

5.2. Evaluation and Results Analysis

This section describes the evaluation of Coop-Scaling by simulation processes. Performance metrics considered are capacity utilization and average response time.

5.2.1. Simulation Settings

Performance models parameters. To estimate performance metrics such as average response time, throughput, and capacity utilization, the PEPA eclipse plugin is used. In this plugin, components *AMF*, *SMF*, and *NRF* are coded, as well as the system equation (Equation 5-2) allowing to carry out steady-state analysis using the fluid approximation (Tribastone and Gilmore, 2011). Values of service rates in the performance models are $r_d = 30$, $r_{smc} = 160$, $r_p = 80$, $r_{reply} = 100$, $r_{idle} = 0.0056$, $r_{req} = 200$, and $r_{ue} = 80$.

MARL-based scaling parameters. RL agents follow the Q-Learning method. Parameters of this method were set as follows. Temperature (T) as 0.4, the discount rate (γ) as 0.8, and learning rate (α) as 0.2.

Workload pattern. The workload pattern is defined by Equation 5-1. Its parameters were set as $\mu = 12$ and $\sigma = 5$, which allow having a variation with a maximum at noon (see Fig. **5-3**). The coefficient a_0 allows setting the variation range of the workload. It was set as $a_0 = 12.000$. Also, for the random samples, $b = 200$ was used.

5.2.2. Performance Assessment of the Network Service

This subsection aims at assessing the behavior of the NS for session establishment (see Fig. **5-2**) without the operation of Coop-Scaling. For this purpose, first, a particular scaling configuration is selected (*e.g.,* $(AMF, SMF, NRF) = (1, 3, 2)$) and evaluated regarding average response time and capacity utilization versus the number of UE. Second, a performance characterization is plotted, taking into account all scaling configurations of the NS regarding the maximum number of UE and average capacity utilization per configuration.

Fig. **5-4** shows the performance that AMF, SMF, and NRF exhibit according to the number of users for the scaling configuration $(AMF, SMF, NRF) = (1, 3, 2)$. The average response

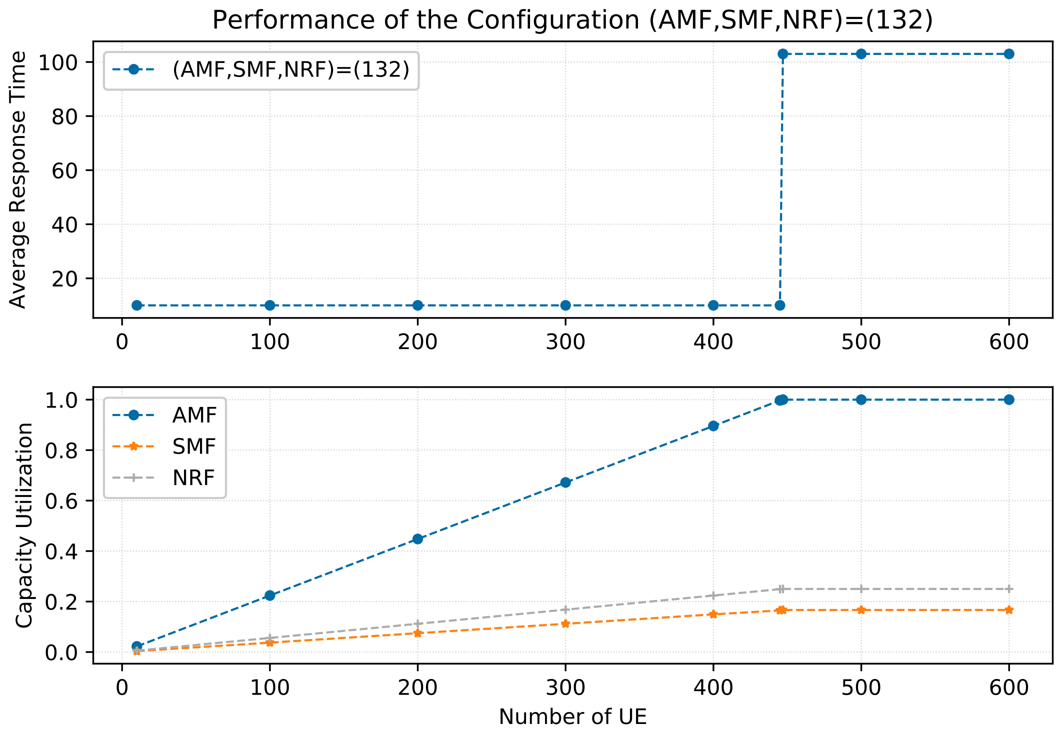


Figure 5-4.: Performance of AMF, SMF, and NRF for configuration $(AMF, SMF, NRF) = (1, 3, 2)$.

time has two performance regions defined by an inflection point of 445 UE. Below this value, the NS presents a good performance, being the response time equal to 10 ms. While this point is exceeded, the performance degrades, reaching a performance of 103 ms. In practice, in the latter region, the NS presents a bad performance because the overall NS is in saturation (*i.e.,* NS reached its maximum capacity) and drops service requests for a number of users greater than 445. From Fig. **5-4**, the module *Environment* of Coop-Scaling can set the performance target, for example, $art_{maximum} = 100ms$.

With respect to the capacity utilization of AMF, SMF, and NRF, it grows from zero to one. The maximum AMF capacity is achieved at the same inflection point where the average response time is degraded. From the capacity utilization curves, it is highlighted two primary facts. First, for a workload of less than 445 UE, SMF, and NRF are oversized, since they have low utilization. Second, for a workload exceeding 445 UE, AMF needs to scale and thus keep the response time below the target (*e.g.,* 100ms). Of course, this analysis is valid for static scaling of the NS for session establishment. In contrast, in Coop-Scaling the agents learn a scaling policy through the RL approach using the Q-Learning method as described in the module *Decision Maker*, whether independent or cooperative.

Taking into account all scaling configurations of the NS for session establishment (see Fig.

5-2), Fig. **5-5** presents a performance characterization for up to three instances in each NF, showing the maximum number of UE versus the average capacity utilization per configuration. A specific configuration represents a combination of the number of service instances of AMF, SMF, and NRF. For example, configuration 221 represents two, one, and one service instances for AMF, SMF, and NRF, respectively.

Fig. **5-5** reveals that various configurations allow handling variations in the workload but at the expense of greater or lesser capacity utilization. For example, up to approximately 900 users can be supported using configurations 233, 311, and 211. However, in configuration 233, AMF, SMF, and NRF are oversized, presenting an average capacity utilization less than 60%. In contrast, configuration 211 presents an average capacity utilization close to 100%. Configuration 233 has eight NF service instances, whereas configuration 211 uses only four NF service instances, being the latter configuration less costly in terms of the number of NF instances. In this example, maybe configuration 311 is the best option for handling up to 900 users. This chapter analyzes these facts when MARL-based scaling is used as described in module *Decision Maker*.

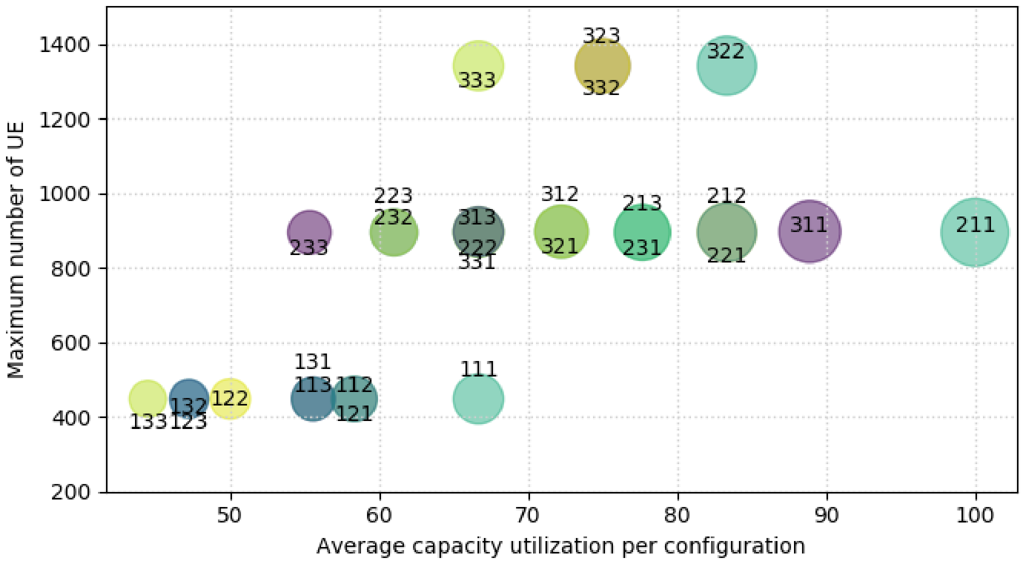


Figure 5-5.: Overall performance characterization.

5.2.3. Simulation Process

It was carried out experiments that allow evaluating the performance of three types of agents (an AMF agent, independent agents, and cooperative agents), who learn a policy of horizontal scaling in each of the environments defined in module *Environment* of Coop-Scaling. In environment 1, which gives rewards according to QoS metrics of average response time, the evaluation ran 300 episodes. It was run the same number of episodes for environment 2, which gives rewards according to shared QoS metrics of capacity utilization. In environment 3, which combines rewards from the previous two environments, it was run 200 episodes. An episode is the execution of the RL process on all samples of the workload. An episode is considered successful if the average response time was below the target (*e.g.,* 100ms) for all workload samples. Each experiment measured successful episodes, average capacity utilization, and average response time. Tables **5-1** and **5-2** record the values of these metrics.

Table **5-1** refers to the learning time, which is the time until the agents converge to the maximum accumulated reward. The learning time allows analyzing how quickly the agents learn, as well as the capacity utilization and the average response time in learning. The learning time for environments 1 and 2 corresponds to the first 200 episodes, whereas it corresponds to the first 100 episodes for environment 3. In turn, Table **5-2** presents the performance once the agents have learned. It corresponds to the last 100 episodes in the experiments carried out. After learning, it is analyzed how well the agents behave regarding successful episodes, capacity utilization, and average response time.

Tables **5-1** and **5-2** record the relationship (in percentage) of successful episodes to the total episodes, whether in learning time or after learning. Thus, a higher percentage of successful episodes indicates better scaling decisions made by agents. The average capacity utilization is the average among all episodes of the average capacity utilization of the three functions (AMF, SMF, and NRF). This metric is measured between zero and one, being better the closer to one, given that the capacity of the functions would be better utilized. The average response time is a QoS metric, which indicates the time at which the control plane of 5GCN responds to session establishment requests made by UE. The average response time is limited by a target value (*e.g.,* 100 ms). Although the three types of agents achieve average response times below this target, it will be preferable to the one that is lower, since it represents in turn greater number of successful episodes.

Table 5-1.: Results of the evaluations in learning time.

Type of agents	Evaluation 1			Evaluation 2			Evaluation 3		
	Successful episodes (%)	Average capacity utilization	Average response time (ms)	Successful episodes (%)	Average capacity utilization	Average response time (ms)	Successful episodes (%)	Average capacity utilization	Average response time (ms)
Single AMF agent	48.6	0.3962	15.6	57.2	0.3973	14.4	77.0	0.3955	12.1
Independent agents	19.2	0.4090	24.1	13.1	0.4215	25.5	49.4	0.4062	16.6
Coop-Scaling	30.9	0.3963	23.6	17.6	0.4235	25.7	44.4	0.4025	19.9

In addition to Tables **5-1** and **5-2**, Figs. **5-6** to **5-8** plot the accumulated reward and the average response time versus the number of episodes of learning. The accumulated reward allows determining when the agent has learned, by convergence of this value to the maximum. Furthermore, the curves of average response time (Fig.s **5-6** to **5-8**) allow

Table 5-2.: Results of the evaluations after learning.

Type of agents	Evaluation 1			Evaluation 2			Evaluation 3		
	Successful episodes (%)	Average capacity utilization	Average response time (ms)	Successful episodes (%)	Average capacity utilization	Average response time (ms)	Successful episodes (%)	Average capacity utilization	Average response time (ms)
Single AMF agent	55.8	0.3971	14.7	64.7	0.3965	13.1	80.5	0.3979	11.3
Independent agents	26.0	0.4030	21.0	25.9	0.4096	20.6	60.1	0.4111	14.4
Coop-Scaling	90.5	0.3621	10.5	66.5	0.4017	12.2	90.5	0.3785	10.5

observing the tendency of the different types of agents in making scaling decisions after learning (convergence), with the best decisions being those that are in the lower part. The analysis of the evaluation results for the three environments is presented below.

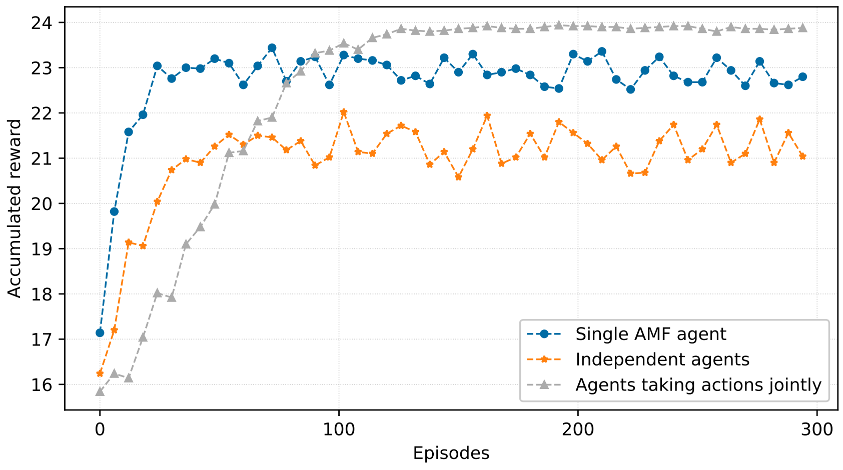
5.2.4. Environment 1, which gives Rewards according to QoS Metrics of Average Response Time

The behavior of the agents at learning time and after learning is described as follows.

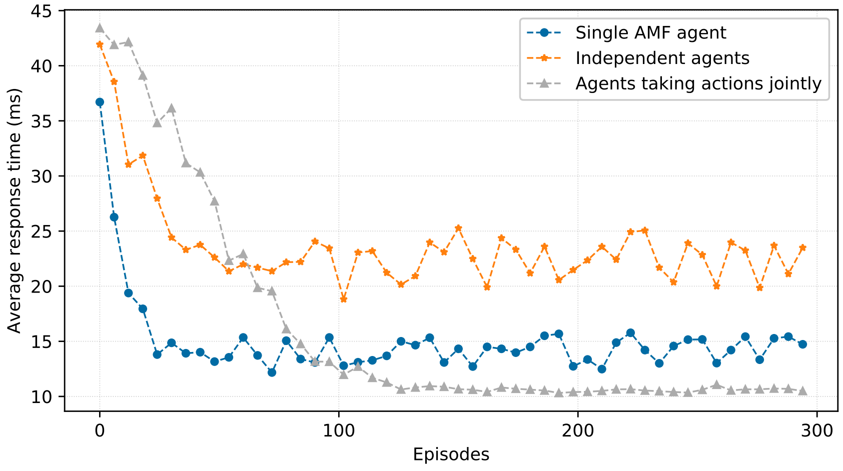
Agents at learning time. Table **5-1** shows that a single AMF agent reaches 53.8% of successful episodes, an average capacity utilization of 0.3970 and the average response time is 15.1ms. Independent agents reach 18.0% of successful episodes, an average capacity utilization of 0.4097 and the average response time is 23.7ms. Cooperative agents reach 52.0% of successful episodes, an average capacity utilization of 0.3890 and the average response time is 18.8ms.

At learning time, a single AMF agent achieves a higher number of successful episodes compared to independent and cooperative agents, because SMF and NRF have been oversized. Oversized SMF and NRF (*i.e.*, SMF and NRF have three service instances) mean that only the AMF agent is worried about the making-decision process for scaling, choosing between one and three services instances depending on the workload pattern. The Decision Maker (the AMF agent) is simpler and has more probability of taking correct actions at learning time. The evaluations results show that a single AMF agent achieves 35.8% and 1.8% more of successful episodes than independent agents and cooperative agents, respectively. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.*, 100ms). However, a single AMF agent achieves the less average response time, which is related with the greater successful episodes. Regarding the average capacity utilization, the three types of agents behave similarly, using a capacity close to 0.4 that is relatively low (probably greater use of AMF, SMF, and NRF instances), which shows the cost of searching for more successful episodes.

Fig. **5-6**a shows that a single AMF agent converges from episode 24, independent agents converge from episode 54, while cooperative agents converge from episode 126. Therefore, a single AMF agent and independent agents converge faster than cooperative agents. The slower convergence of cooperative agents is due to the greater state space that implies more steps to learn. For the average response time, the worst behavior is for independent agents, who present a more considerable value of this metric. Independent agents take actions separately, entailing a low rate of successful episodes.



(a) Accumulated reward.



(b) Average response time.

Figure 5-6.: Accumulated reward and average response time in Environment 1.

Agents after learning. Table **5-2** shows that a single AMF agent reaches 60.4% of successful episodes, an average capacity utilization of 0.3965 and the average response time is 14.4ms. Independent agents reach 21.1% of successful episodes, an average capacity utilization of 0.4065 and the average response time is 22.6ms. Cooperative agents reach 89.2% of successful episodes, an average capacity utilization of 0.3671 and the average response time is 10.5ms.

After learning, cooperative agents taking actions jointly outperform a single AMF agent and independent agents. They achieve 28.8% and 68.1% more successful episodes than a single AMF agent and independent agents, respectively. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.,* 100ms). However, cooperative agents achieve the lower average response time that is related to the greater successful episodes. Regarding the average capacity utilization, cooperative agents have the lowest average capacity utilization, showing probably greater use of AMF, SMF, and NRF instances.

Fig. **5-6**b also shows that cooperative agents overcome a single AMF agent and independent agents because the curve of cooperative agents is below the other two. For the average response time, the worst behavior is for independent agents, who present a more considerable value of this metric. Independent agents take actions separately, entailing a low rate of successful episodes.

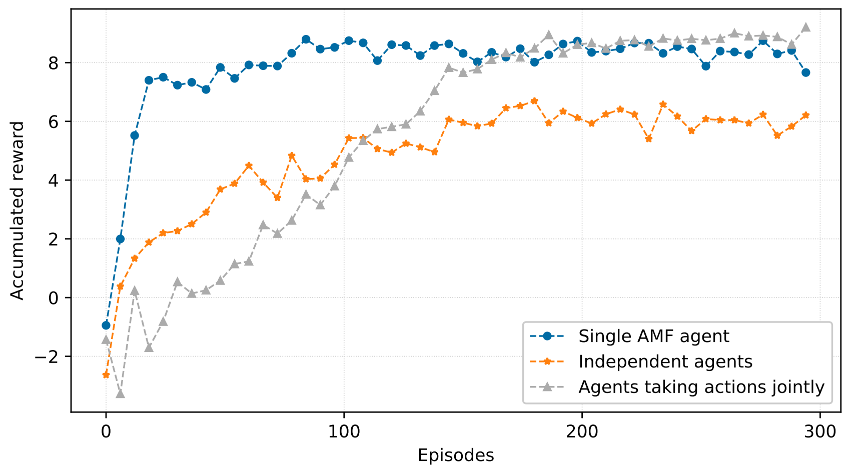
5.2.5. Environment 2, which gives Rewards according to shared QoS Metrics of Capacity Utilization

The behavior of the agents at learning time and after learning is described as follows.

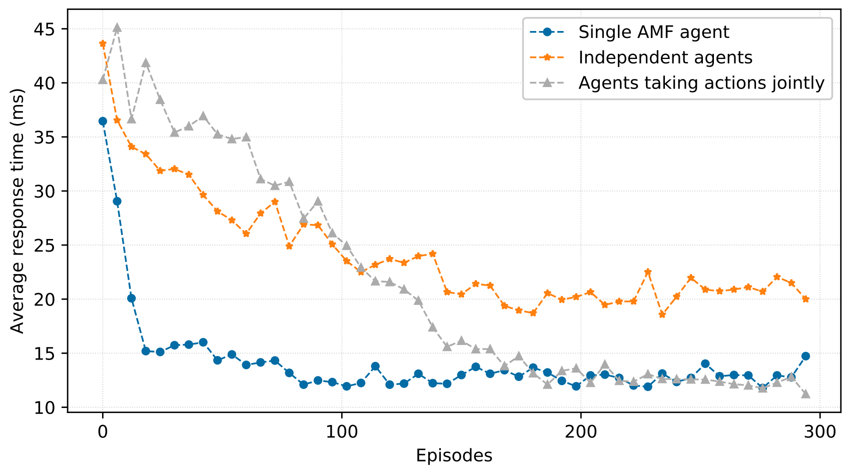
Agents at learning time. Table **5-1** shows that a single AMF agent reaches 57.2% of successful episodes, an average capacity utilization of 0.3973 and the average response time is 14.4ms. Independent agents reach 13.1% of successful episodes, an average capacity utilization of 0.4215 and the average response time is 25.5ms. Cooperative agents reach 17.6% of successful episodes, an average capacity utilization of 0.4235 and the average response time is 25.7ms.

At learning time, a single AMF agent achieves a higher number of successful episodes compared to independent and cooperative agents, because AMF and SMF have been oversized. A single AMF agent achieves 44.1% and 39.6% more of successful episodes than independent agents and cooperative agents, respectively. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.,* 100ms). However, a single AMF agent achieves the less average response time, which is related with the greater successful episodes. Regarding the average capacity utilization, the three types of agents behave similarly, using a capacity close to 0.4 that is relatively low (probably greater use of AMF, SMF, and NRF instances), which shows the cost of searching for more successful episodes.

The convergence (see Fig. **5-7**a) for a single AMF agent is from episode 84, independent agents converge from episode 144, while cooperative agents converge from episode 180. Therefore, a single AMF agent and independent agents converge faster than cooperative agents. For the average response time, the worst behavior is for cooperative agents due to its slower convergence. This behavior implies more incorrect episodes at learning time.



(a) Accumulated reward.



(b) Average response time.

Figure 5-7.: Accumulated reward and average response time in Environment 2.

Agents after learning. Table **5-2** shows that a single AMF agent reaches 64.7% of successful episodes, an average capacity utilization of 0.3965 and the average response time is 13.1ms. Independent agents reach 25.9% of successful episodes, an average capacity utilization of 0.4096 and the average response time is 20.6ms. Cooperative agents reach 66.5% of successful episodes, an average capacity utilization of 0.4017 and the average response time is 12.2ms.

After learning, cooperative agents behave better than a single AMF agent and independent agents. They achieve 1.8% and 36.6% more successful episodes than a single AMF agent and independent agents, respectively. However, the improvement compared to a single AMF agent is low. This small difference is due to the fact that only capacity utilization information is taken into account. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.,* 100ms). However, cooperative agents achieve the lower average response time that is related to the greater successful episodes. Concerning the average capacity utilization, cooperative agents have more utilization than a single AMF, which probably shows a lower cost regarding the number of instances used.

Fig. **5-7**b also shows that cooperative agents overcome a single AMF agent and independent agents because the curve of cooperative agents is below the other two. For the average response time, the worst behavior is for independent agents, who present a more considerable value of this metric. Independent agents take actions separately, entailing a low rate of successful episodes.

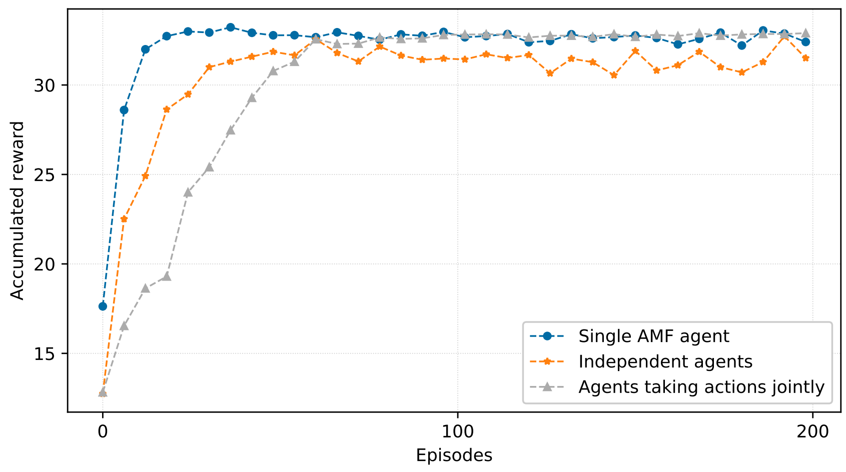
5.2.6. Environment 3, which gives Rewards according to QoS Metrics of Average Response Time and Capacity Utilization

The behavior of the agents at learning time and after learning is described as follows.

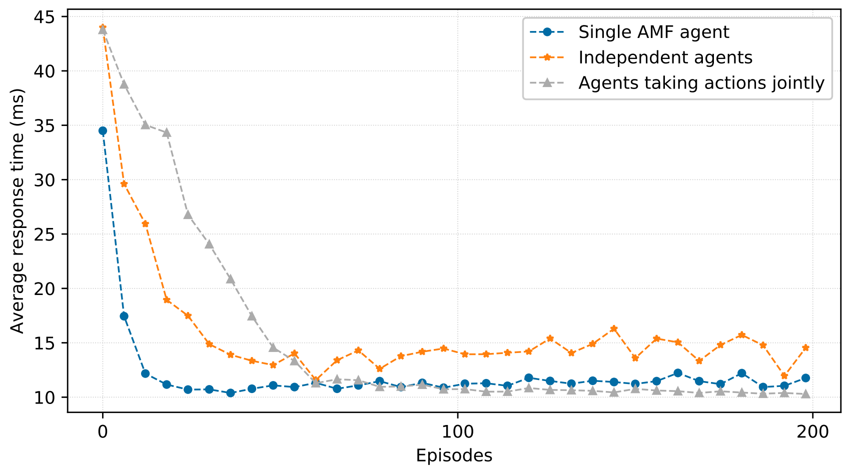
Agents at learning time. Table **5-1** shows that a single AMF agent reaches 77.0% of successful episodes, an average capacity utilization of 0.3955 and the average response time is 12.1ms. Independent agents reach 49.4% of successful episodes, an average capacity utilization of 0.4062 and the average response time is 16.6ms. Cooperative agents reach 44.4% of successful episodes, an average capacity utilization of 0.4025 and the average response time is 19.9ms.

At learning time, a single AMF agent achieves a higher number of successful episodes compared to independent and cooperative agents, because AMF and SMF have been oversized. A single AMF agent achieves 27.6% and 32.6% more of successful episodes than independent agents and cooperative agents, respectively. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.,* 100ms). However, a single AMF agent achieves the less average response time, which is related with the greater successful episodes. Regarding the average capacity utilization, the three types of agents behave similarly, using a capacity close to 0.4 that is relatively low (probably greater use of AMF, SMF, and NRF instances), which shows the cost of searching for more successful episodes.

The convergence (see Fig. **5-8**a) for a single AMF agent is from episode 24, independent agents converge from episode 48, while cooperative agents converge from episode 60. Therefore, a single AMF agent and independent agents converge faster than cooperative agents. For the average response time, the worst behavior is for cooperative agents due to its slower convergence. This behavior implies more incorrect episodes at learning time.



(a) Accumulated reward.



(b) Average response time.

Figure 5-8.: Accumulated reward and average response time in Environment 3.

Agents after learning. Table **5-2** shows that a single AMF agent reaches 80.5% of successful episodes, an average capacity utilization of 0.3979 and the average response time is 11.3ms. Independent agents reach 60.1% of successful episodes, an average capacity utilization of 0.4111 and the average response time is 14.4ms. Cooperative agents reach 90.5% of successful episodes, an average capacity utilization of 0.3785 and the average response time is 10.5ms.

After learning, cooperative agents behave better than a single AMF agent an independent agents. They achieve 10% and 30.4% more successful episodes than a single AMF agent and independent agents, respectively. Regarding the average response time, the three types of agents handle this metric below the target (*i.e.,* 100ms). However, cooperative agents achieve the lower average response time that is related to the greater successful episodes. Regarding the average capacity utilization, cooperative agents have the lowest average capacity utilization, showing probably greater use of AMF, SMF, and NRF instances.

Fig. **5-8**b also shows that cooperative agents overcome a single AMF agent and independent agents because the curve of cooperative agents is below the other two. For the average response time, the worst behavior is for independent agents, who present a more considerable value of this metric. Independent agents take actions separately, entailing a low rate of successful episodes.

5.2.7. Observations

Previous evaluations raise the following observations.

- The approach of scaling AMF only while SMF and NRF are oversized, although it achieves a high value in the number of successful episodes, it does not represent the most appropriate alternative because SMF and NRF could be underused in some time.
- Independent agents have the worst behavior compared to a single AMF and cooperative agents, evidenced in fewer successful episodes and higher response time value. This poor performance is because the state space changes when an agent takes its action.
- Cooperative agents consider the actions of each other. This joint operation allows them to behave better than a single AMF agent and independent agents.
- The achievement of a more significant number of successful episodes is inversely related to the lower capacity utilization. This relationship implies that RL agents try to use more NF instances to achieve better behavior.

5.3. Final Remarks

This chapter presented and evaluated Coop-Scaling, a cooperative scaling mechanism based on MARL in the control plane of 5GCN. Coop-Scaling enforces agents to cooperate by sharing information and taking actions jointly. Results show that the scaling using agents taking action jointly are more effective than the scaling of a single agent or independent agents.

Experiments considered an NS for session establishment making up of three NFs (AMF, SMF, and NRF) and up to three instances for each NF. The RL agents used the Q-Learning method.

6. Scalability and Performance Analysis in 5G Core Network Slicing

Network slicing is a central concept in the 5G mobile networks, which aims at running multiple end-to-end logical networks (*i.e.,* encompassing core and access) as independent business operations on shared infrastructure (NGMN Alliance, 2016, Subedi et al., 2021). 5G network slicing envisions to support different use cases, such as eMBB, mIoT, and URLLC (Rost et al., 2017, Tullberg et al., 2016, Zhang et al., 2017, Barakabitze et al., 2020). Slicing 5GCN involves specifying NSs according to the functional and quality requirements of use cases above mentioned (Campolo et al., 2017). To specify 5G core NSs before deployment tasks, the Network Service Providers (NSPs) need to analyze mandatorily under varying workloads the performance and scalability of 5GNSLs. This analysis is pivotal for NSPs dimension their capacity (*e.g.,* size of physical/virtual infrastructure) (Chiha et al., 2020).

The work (Prados-Garzon et al., 2017b) presented an approach to evaluate the performance and scalability of a vMME by using Queuing Networks and measuring the average response time. As this approach is 4G-oriented, it does not model the 5GCN that defines new NFs, such as AMF, SMF, and AUSF (3GPP, 2018c). Also, this approach does not measure some essential performance indices, such as processor utilization and throughput. The works Trivisonno et al. (2018), Campolo et al. (2018), and Schneider et al. (2019) used different techniques, such as simulations, emulations, and QPNs, to evaluate performance in 5GNSLs. However, these works do not analyze the scalability of 5GNSLs. The work (Qu et al., 2021) modeled 5GNSLs as NSCs, where a source node generates service requests that traverse through several VNFs in sequence towards a destination node. This work is for the data plane, and control plane NFs are not involved.

The above works highlight the necessity of investigating the scalability of 5GNSLs located at the core network deeply. Furthermore, NSPs should adopt modeling and evaluation formalisms to perform such an in-depth scalability and performance investigation.

This chapter proposes a method based on PEPA (Hillston, 1993) for modeling, evaluating, and analyzing the scalability and performance of 5GNSLs in the core network systematically. PEPA has been used to model and evaluate distributed systems. The work (Almutairi and Thomas, 2020) modeled a web-based sales system in the presence of denial of service attacks.

Evaluation results regarding throughput and population level show how the attacks negatively impact the orders of customers. The work (Sanders et al., 2020) proposed the Imperial PEPA Compiler, an alternative to the PEPA Eclipse plugin (Tribastone et al., 2009), for robustness analysis of resource allocation. The proposed tool overcomes some limitations, such as the size and complexity of models imposed by the plugin. The work (Hillston et al., 2011) introduced an approach that uses continuous PEPA models to represent large systems with multiple replications in components, such as clients, servers, and devices.

In this chapter, PEPA is used to introduce new composite structures intended to model and evaluate 5GCN procedures. This chapter illustrates how to use the proposed method by two case studies: the session establishment 5GNSL and the user registration in a V2X 5GNSL. The case studies focus on dimensioning the capacity of slices regarding users to attend and QoS requirements. The accuracy of the method is validated with the LQN modeling formalism. Results show that the method is useful to model 5GNSLs core procedures and dimension the capacity of slices. The validation results corroborate that the PEPA-based method measures performance, in terms of throughput, average response time, and processor utilization, with negligible difference regarding a traditional approach like LQN.

Fig. **6-1** outlines the PEPA-based method for scalability and performance analysis of 5GCN. Analysis results allow defining and evaluating optimization strategies considering variations in the number of concurrent users.

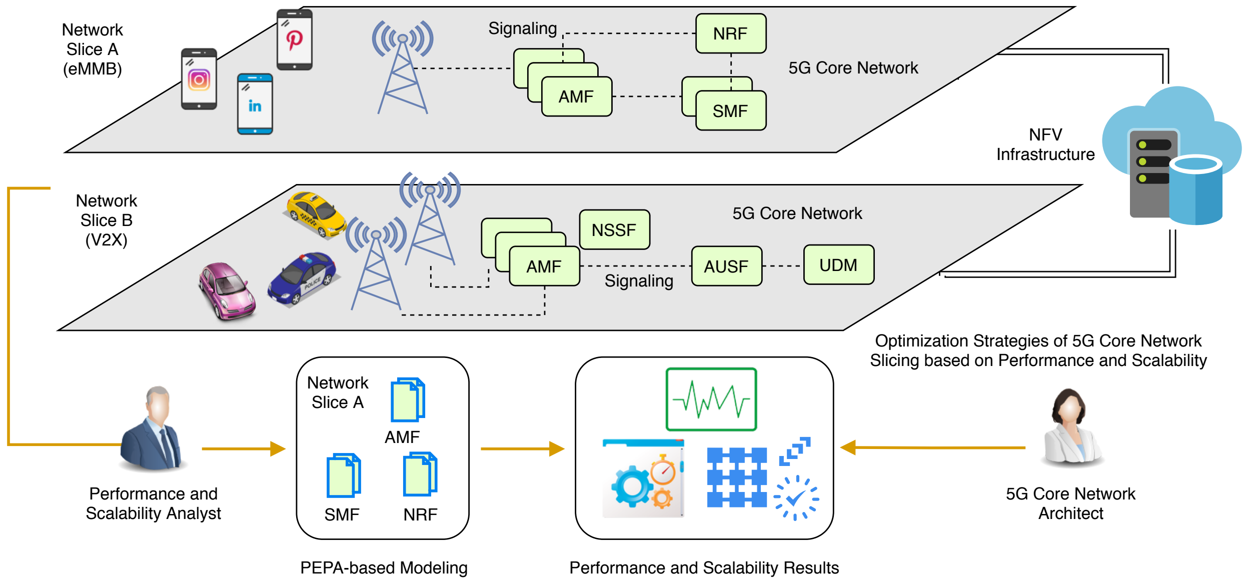


Figure 6-1.: PEPA-based method for scalability and performance analysis in 5GNSLs.

The contributions presented in this chapter are:

- A method that introduces new composite structures based on PEPA and intends to model and evaluate 5GCN procedures.

- The scalability analysis in 5GNSLs considering concurrent users and the multiplicity of NFs.
- Two use cases that illustrate how to use the method for modeling, evaluating, and analyzing the performance and scalability in 5GNSLs.
- The dimensioning of 5GNSLs regarding concurrent users and QoS.

The remainder of this chapter is organized as follows. Section 6.1 details the PEPA-based approach. Section 6.2 and Section 6.3 evaluate the scalability and performance of the session establishment 5GNSL and the user registration V2X 5GNSL, respectively. Section 6.4 provides some conclusions.

6.1. Scalability and Performance Analysis Method

This section describes the PEPA-based method that allows modeling, evaluating, and analyzing the performance and scalability of 5GNSLs regarding response time, throughput, and processor utilization. Thus, this method facilitates the dimensioning of 5GNSLs in the core network, which leads to avoiding the wasting of resources.

6.1.1. Modeling Fundamentals

Some assumptions are made for modeling 5GNSLs close to reality. These assumptions are related to 5GCN features, such as the communication pattern and the high number of concurrent users.

- For reducing complexity, 5GCN control plane procedures are modeled in a high-abstraction level without losing representativeness in their overall behavior.
- UE and NFs are represented by sequential PEPA-components because 5GCN control plane procedures operate sequentially and distributively.
- It is modeled the communication between NFs by the client-server pattern (Oluwatosin, 2014) with synchronous and asynchronous (Hillston et al., 2011) mode due to NFs can act as server and client. For example, UE sends requests to AMF (server). In turn, AMF (client) sends requests to SMF.
- It is assumed that each NF instance has a pool of threads to handle in parallel the myriad of concurrent incoming requests in the 5GCN control plane.

6.1.2. Modeling Patterns

Communication. This method supports the modeling of synchronous and asynchronous communication. The synchronous mode is represented by two sequential actions:
$(req_{service}, v).(rep_{service}, v)$

The first one defines the service request. The second action represents the response. Here, $1/v$ is the expected duration of request and response actions in time units. Considering the running example (see Fig. **2-3**), AMF requests the discovery service of SMF from NRF. The communication between AMF and NRF is modeled as $(req_{discovery}, v).(rep_{discovery}, v)$. Since the communication between NFs is almost instantaneous, v is set to an enough high value like $100,000$. The asynchronous mode is modeled by a request action $(req_{service}, v)$. In contrast to the synchronous mode, the NF acting as the client is not blocked to receive or send other operations.

Processing on a VM. A VM hosting an NF uses processing to process a request. For modeling such processing, it is used the pattern defined in (Hillston et al., 2011, Williams and Clark), in which a two-state sequential component models a single processing unit. The first state enables an action to obtain exclusive access to the resource, whereas the second state performs all the actions deployed on the processor. For example, the processing in NRF can be modeled as in Listing 6.1. $Nrfp_1$ (first state) gets access to the processor using the action (get_{nrfp}, r_p). $Nrfp_2$ (second state) performs the action $(discover, r_d)$. This action performs the discovery service by NRF.

$$Nrfp_1 \overset{def}{=} (get_{nrfp}, r_p).Nrfp_2$$
$$Nrfp_2 \overset{def}{=} (discover, r_d).Nrfp_1$$

Listing 6.1: NRFP

6.1.3. Functioning

Fig. **6-2** depicts the functioning, actors, and tasks of the method. The Slicing Architect defines the service function chains and their signaling procedures (*e.g.,* session establishment). Furthermore, the Architect uses the performance analysis results to select the appropriate scaling configuration (*i.e.,* type and the number of NF instances) to handle the expected workload.

The Performance Analyst creates PEPA models to represent NFs, specifying in the PEPA editor the equations that define the cooperation between NFs. It is noteworthy that the Analyst can reuse existing PEPA models and slicing equations available in the *PEPA Models Repository.* Furthermore, since NF models include service rates, the Analyst retrieves them from the *NF Performance Repository.* The Analyst also performs steady-state analysis of slices for measuring performance indices, such as average response time, throughput, and processor utilization. The PEPA Analysis Tool enables steady-state analysis. The 5G Network Administrator uses the service chain specifications to deploy them.

The NF Developer implements NFs and runs performance tests to determine the average service rates of operations, such as registering or discovering an NF in and from a repository. These rates are useful to model the expected duration of the operations. For example, a Developer can measure the service rates by generating N calls to the service by calculating the average time taken to generate responses. This Developer also records pairs of values that relate operations of NFs with service rates in the *NF Performance Repository*.

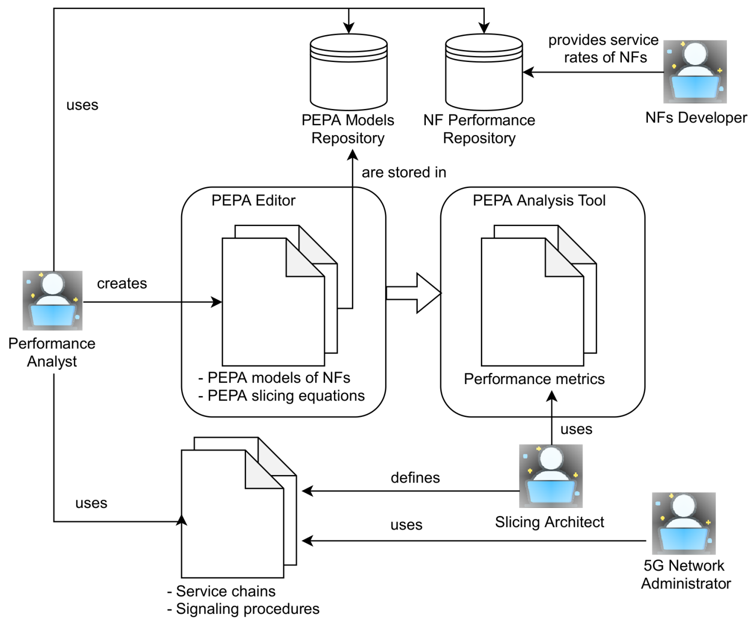


Figure 6-2.: Method functioning - actors and tasks.

The state-space underlying PEPA models can exponentially grow when the number of components increases (state-space explosion problem), which brings a high computational cost. Note that this problem is present in other analysis techniques, such as Petri nets, where the quantitative results depend on the numerical solution of the underlying Continuous-Time Markov Chain (CTMC) (Donatelli et al., 1995). Conversely, traditional queuing networks do not suffer from this problem due to their low computational cost that adapts well to the size of the system under study (Tribastone, 2010). However, queuing networks consider the exclusive possession of a resource (*e.g.,* network node) and do not model nested service requests that characterize client-server architectures. To address the state explosion problem, in this chapter, it is used the fluid approximation proposed in (Tribastone and Gilmore, 2011) that results in a set of Ordinary Differential Equations (ODEs), reducing the computational cost for solving PEPA models.

6.2. Case Study: Session Establishment 5GNSL

This case study aims three-folds. First, presenting the use of the method by modeling the session establishment 5GNSL. Second, dimensioning this 5GNSL to satisfy QoS requirements and avoid bottlenecks by its evaluation and analysis in terms of average response time, throughput, processor utilization, and scalability when NF instances and concurrent users vary. Third, validating the method with LQN.

The PEPA Eclipse plugin is used to develop the method and performing a steady-state analysis of the session establishment 5GNSL. This plugin was deployed in an Intel Core i5 PC (1.7 GHz) with 4 GB of RAM.

6.2.1. Modeling

The session establishment is a primary task of 5GCN that allows end-users to use session-based 5G applications. A session is the user activity carried out between the instant the user launches and closes a network application. Within a session, the application sends or receives all necessary data from performing tasks, such as download a web page, streaming a video, or make a call. The Inter-Arrival Time (IAT) is the time interval between the start of two consecutive sessions (Prados-Garzon et al., 2017b).

Recall the service chain (Fig. **2-2**) and the message sequence (Fig. **2-3**) of the session establishment 5GNSL. Modeling this slice comprises *1)* Defining service rates for UE, AMF, SMF, and NRF; these rates model the duration of actions in PEPA components. *2)* Composing new PEPA structures to model UE, AMF, SMF, and NRF; these structures use the PEPA operators and the modeling patterns to create sequential components. *3)* Modeling the interaction between PEPA components by the cooperation operator; this operator is also useful to define the overall system model (*i.e.,* the model of the session establishment 5GNSL).

Service rates. Table **6-1** presents the service rates (r) in UE, AMF, SMF, and NRF, as well as in processors of VMs that host these NFs. $1/r$ defines, in time units, the average execution demands for actions in NFs or processing on VMs.

PEPA components. It is modeled UE, AMF, SMF, NRF, and the processing of VMs that host them as PEPA components. The states are noted with the name of corresponding NF (first letter in uppercase) and a sequential number (*e.g.,* Amf_1). The action types are noted

Table 6-1.: Service rates for modeling the session establishment 5GNSL.

Service rate	Description
r_{iat}	$1/r_{iat}$ (inter-arrival time) is the time that UE interposes between successive requests of session establishment
r_{pr}	$1/r_{pr}$ is the average time demand for preparing the service workflow
r_r	$1/r_r$ is the average time demand for preparing a response message by AMF
r_d	$1/r_d$ is the average time that takes the discovery service performed by NRF
r_{sc}	$1/r_{sc}$ is the average time that takes the session creation service performed by SMF
r_p	$1/r_p$ is the average execution demand on VM's processors
v	v is a high service rate that models almost instantaneous service calls and replies between NFs

in lowercase (*e.g.,* discovery). Action types are noted by subscripts to add details. For example, the notation to access to the processor of AMF, call to an operation served by NRF, and service request and response for discovery is get_{amfp}, $call_{nrf}$, req_d, and rep_d, respectively.

Component *UE* models users requesting the session establishment to the network slice (see Listing 6.2). The behavior of UE is cyclic and interposes an IAT between successive requests. UE has two states. The first one (Ue_1) gets access to the processor using the action (get_{uep}, r_p), and then, performs the thinking action $(think, r_{iat})$. The second state (Ue_2) models the synchronous actions request and response that UE and AMF interchange to establish a session. These actions are (req_{se}, v) and (rep_{se}, v).

$$Ue_1 \overset{def}{=} (get_{uep}, r_p).(think, r_{iat}).Ue_2$$
$$Ue_2 \overset{def}{=} (req_{se}, v).(rep_{se}, v).Ue_1$$

Listing 6.2: UE

Component *AMF* is the entry point for session establishment requests from UE. The action sequence is defined for a set of states Amf_i, $i = \{1, ..., 12\}$ as follows. Amf_1 models the service request. Amf_2 is a local action that prepare the service workflow. Amf_3 uses the choice operator and allows for the fork of the action sequence depending on regular or emergency requests. An occurrence probability of 0.95 modifies the rate for regular requests, whereas a probability of 0.05 does for emergency requests. Amf_4 to Amf_8 are for regular requests, Amf_4 starts a call to NRF for the discovery service. Amf_5 models the synchronous communication between AMF and NRF. Amf_6 starts a call to SMF for the creation of a session management context. Amf_7 represents the synchronous communication between AMF and SMF. Amf_8 prepares a message response to UE. Amf_9 to Amf_{11} are specific for emergency requests. In this case, AMF uses a statically configured SMF and does not require the discovery service by NRF. Amf_9 starts a call to SMF. Amf_{10} models the synchronous communication between AMF and SMF. Amf_{11} prepares a message response to UE. Finally, the fork joins in Amf_{12}, which models the session establishment response. Listing 6.3 presents the PEPA-based model for AMF.

$$\begin{aligned}
&Amf_1 \overset{def}{=} (req_{se}, v).Amf_2\\
&Amf_2 \overset{def}{=} (get_{amfp}, r_p).(prepare, r_{pr}).Amf_3\\
&Amf_3 \overset{def}{=} (choose, 0.95 \times v).Amf_4\\
&\qquad\quad +(choose, 0.05 \times v).Amf_9\\
&Amf_4 \overset{def}{=} (get_{amfp}, r_p).(call_{nrf}, v).Amf_5\\
&Amf_5 \overset{def}{=} (req_d, v).(rep_d, v).Amf_6\\
&Amf_6 \overset{def}{=} (get_{amfp}, r_p).(call_{1smf}, v).Amf_7\\
&Amf_7 \overset{def}{=} (req_{1sc}, v).(rep_{1sc}, v).Amf_8\\
&Amf_8 \overset{def}{=} (get_{amfp}, r_p).(respond_1, r_r).Amf_{12}\\
&Amf_9 \overset{def}{=} (get_{amfp}, r_p).(call_{2smf}, v).Amf_{10}\\
&Amf_{10} \overset{def}{=} (req_{2sc}, v).(rep_{2sc}, v).Amf_{11}\\
&Amf_{11} \overset{def}{=} (get_{amfp}, r_p).(respond_2, r_r).Amf_{12}\\
&Amf_{12} \overset{def}{=} (rep_{se}, v).Amf_1
\end{aligned}$$

Listing 6.3: AMF

Component *NRF* is modeled using actions that represent discovery requests and replies, which are performed in cooperation with AMF. These actions are (req_d, v) and (rep_d, v), for request and reply, respectively. In addition, a state (Nrf_2) defines the access to the processor and the local operation that performs the discovery service. Nrf_2 is modeled using the sequential actions (get_{nrfp}, r_p) and $(discovery, r_d)$, for access to the processor and the discovery service, respectively. Listing 6.4 presents the PEPA-based model for NRF.

$$\begin{aligned}
&Nrf_1 \overset{def}{=} (req_d, v).Nrf_2\\
&Nrf_2 \overset{def}{=} (get_{nrfp}, r_p).(discover, r_d).Nrf_3\\
&Nrf_3 \overset{def}{=} (rep_d, v).Nrf_1
\end{aligned}$$

Listing 6.4: NRF

Component *SMF* models the service for creating a session management context that allows UE to use 5G applications. SMF is modeled similarly to NRF. Four additional actions are needed, two actions are now necessary to model the request and two ones for the response. This addition is due to the fork in AMF that handles regular and emergency service.

The state (Smf_2) defines the access to the processor and the local operation that performs the session creation service. Smf_2 is modeled using the sequential actions (get_{smfp}, r_p) and ($createSession, r_{cs}$), for access to the processor and the session creation service, respectively. Listing 6.5 presents the PEPA-based model for SMF.

$$\begin{aligned}
&Smf_1 \overset{def}{=} (req_{1sc}, v).Smf_2 + (req_{2sc}, v).Smf_2 \\
&Smf_2 \overset{def}{=} (get_{smfp}, r_p).(createSession, r_{cs}).Smf_3 \\
&Smf_3 \overset{def}{=} (rep_{1sc}, v).Smf_1 + (rep_{2sc}, v).Smf_1
\end{aligned}$$

Listing 6.5: SMF

Components *UEP*, *AMFP*, *NRFP*, and ***SMFP*** model the processing entities on which UE, AMF, NRF, and SMF execute, respectively. These processing entities are modeled following the pattern of processing given in the Subsection 6.1.2, in which each component hast two states. The first one gets access to the processor. The second state performs the actions deployed on the processor. As the model for *NRFP* was already presented in Listing 6.1, Lists 6.6, 6.7, and 6.8 present the PEPA-based models for *UEP*, *AMFP*, and *SMFP*, respectively.

$$\begin{aligned}
&Uep_1 \overset{def}{=} (get_{uep}, r_p).Uep_2 \\
&Uep_2 \overset{def}{=} (think, r_{iat}).Uep_1
\end{aligned}$$

Listing 6.6: UEP

$$\begin{aligned}
&Amfp_1 \overset{def}{=} (get_{amfp}, r_p).Amfp_2 \\
&Amfp_2 \overset{def}{=} (prepare, r_{pr}).Amfp_1 + (call_{nrf}, v).Amfp_1 \\
&\qquad +(call_{1smf}, v).Amfp_1 + (respond_1, r_r).Amf_1 \\
&\qquad +(call_{2smf}, v).Amf_1 + (respond_2, r_r).Amf_1
\end{aligned}$$

Listing 6.7: AMFP

$$\begin{aligned}
&Smfp_1 \overset{def}{=} (get_{smfp}, r_p).Smfp_2 \\
&Smfp_2 \overset{def}{=} (createSession, r_{cs}).Smfp_1
\end{aligned}$$

Listing 6.8: SMFP

<u>*Overall system model*.</u> The operator of cooperation (Subsection 2.1.8) is used to model the session establishment 5GNSL. List 6.9 presents the system model by using the new 5G-oriented PEPA constructions.

$$\begin{aligned}
&((((Ue_1[N_{ue}] \underset{L_1}{\bowtie} Amf_1[N_{amf} \cdot N_{amfp} \cdot N_t]) \\
&\underset{L_2}{\bowtie} Nrf_1[N_{nrf} \cdot N_{nrfp} \cdot N_t])
\end{aligned}$$

$$
\begin{aligned}
&\underset{L_3}{\bowtie} Smf_1[N_{smf} \cdot N_{smfp} \cdot N_t])\\
&\underset{L_4}{\bowtie} (((Uep_1[N_{uep}] \underset{\emptyset}{\bowtie} Amfp_1[N_{amf} \cdot N_{amfp}])\\
&\underset{\emptyset}{\bowtie} Nrfp_1[N_{nrf} \cdot N_{nrfp}]) \underset{\emptyset}{\bowtie} Smfp_1[N_{smf} \cdot N_{smfp}]))
\end{aligned}
$$

$$
\begin{aligned}
L_1 &= \{req_{se}, rep_{se}\}\\
L_2 &= \{req_d, rep_d\}\\
L_3 &= \{req_{1sc}, rep_{1sc}, req_{2sc}, rep_{2sc}\}\\
L_4 &= \{get_{uep}, think, get_{amfp}, prepare, call_{nrf},\\
&\qquad call_{1smf}, call_{2smf}, respond_1, respond_2,\\
&\qquad get_{nrfp}, discover, get_{smfp}, createSession\}\\
\emptyset &= \{\}
\end{aligned}
$$

Listing 6.9: Overall system model

Where N_{ue} is the number of UE. N_{amf}, N_{nrf}, and N_{smf} are the number of available instances of AMF, SMF, and NRF, respectively. N_{amfp}, N_{smfp}, and N_{nrfp} are the number of processors that are allocated to each instance of AMF, SMF, and NRF, respectively. It is noteworthy that each processor can handle a set of concurrent threads, which is noted by N_t. Thus, the product $N_{nf} \cdot N_{nfp} \cdot N_t$ represents the total number of threads of an NF. Moreover, the product $N_{nf} \cdot N_{nfp}$ is the total number of processors allocated to an NF. Also note that if a UE has a single processor, N_{ue} and N_{uep} are equal.

6.2.2. Performance Metrics

To measure the performance and scalability of 5GNSLs, it is used the metrics processor utilization, throughput, average response time, and scalability.

Utilization. This metric measures the total utilization of the processor when it is performing actions at the steady-state (Williams and Clark). For example, the utilization of NRFP (Listing 6.1) is its population level when NRFP performs the action $(discover, r_d)$.

Throughput. This metric measures the frequency at which an action is performed at the steady-state (Williams and Clark). For instance, the throughput for the session establishment 5GNSL is measured in the action (rep_{se}, v) of AMF (Listing 6.3) which represents the session establishment responses per time unit.

Average response time. This metric measures the average time that a component spends passing throughout a particular set of states (Williams and Clark). For example, in the session establishment 5GNSL, the average response time corresponds to the time that AMF spends to discover SMF, create the session context, and prepare the response to UE.

Scalability Metric. This metric measures the scalability (Equation 6-1) as the ratio between the productivity of a system at two different scale factors k_2 and k_1 (Jogalekar and Woodside, 2000). If $\psi(k_1, k_2) < 1$, the productivity of the system at scale k_2 is less than at scale k_1. When $\psi(k_1, k_2) = 1$, the productivity of system at scales k_1 and k_2 is equal. If $\psi(k_1, k_2) > 1$, the productivity of system at scale k_2 is higher than at scale k_1.

$$\psi(k_1, k_2) = \frac{F(k_2)}{F(k_1)} \tag{6-1}$$

Where, $F(k)$ is the productivity of a system at the scale k, given by Equation 6-2.

$$F(k) = \frac{\lambda(k) \cdot f(k)}{C(k)} \tag{6-2}$$

Where $\lambda(k)$ is the average throughput achieved at scale k, $C(k)$ is the running cost per second of the system at scale k, and $f(k)$ (Equation 6-3) is a value function determined by evaluating the performance of the scaled system. The value function given in (Jogalekar and Woodside, 2000) is considered to compare the average response time at scale k, $\overline{T}(k)$, with a target value $\widehat{T}$:

$$f(k) = \frac{1}{1 + \overline{T}(k)/\widehat{T}} \tag{6-3}$$

It is noteworthy that $f(k)$ rewards the productivity. This reward tends to one if $\overline{T}(k)$ is relatively much smaller than $\widehat{T}$. It is near to 0.5 if $\overline{T}(k)$ is close to $\widehat{T}$. It tends to zero if $\overline{T}(k)$ is relatively much higher than $\widehat{T}$.

6.2.3. Results and Analysis

Table **6-2** presents the service rates used in the experimental evaluation. In particular, for $1/r_{iat}$, it is used 600 seconds as IAT per subscriber, which is a typical value found in (Oliveira et al., 2014) from an analysis of the usage pattern of mobile data traffic. The other service rates are set by performing measurements in a 5GCN prototype available in (Arteaga, 2020). In the experiments, it is set $N_t = 36$ as the number of threads per processor, which is in the range supported by modern multi-task processors, and varied the number of concurrent users from 1 to $100,000$ to analyze configurations extensively.

Fig. **6-3** presents the throughput results for the basic configuration of the session establishment 5GNSL, $(N_{nrf}, N_{smf}, N_{amf}) = (1, 1, 1)$. These results show how the throughput

Table 6-2.: Service rates used for numerical experimentation.

Service Rate	Value	Description
r_{iat}	1/600	Rate between subsequent requests
r_{pr}	1/0.001	Rate of initial preparing by AMF
r_r	1/0.02	Rate of reply by AMF
r_d	1/0.03	Rate of discovery by NRF
r_{sc}	1/0.025	Rate of session creation by SMF
r_p	1/0.00001	Rate of processing on VMs
v	1/0.00001	Rate of almost real-time communication

saturation point depends on the number of processors that the NF instance uses. In particular, configuration (1,1,1) can handle up to 35, 70, and 140 session establishment requests per second with one, two, and four processors per NF instance, respectively. Once the basic configuration achieves these saturation points, the session establishment 5GNSL starts to drop incoming requests. Summarizing, the results mentioned above confirm the correlation between throughput and number of processors; few processors lead to support low throughput and, in turn, many processors support high throughput.

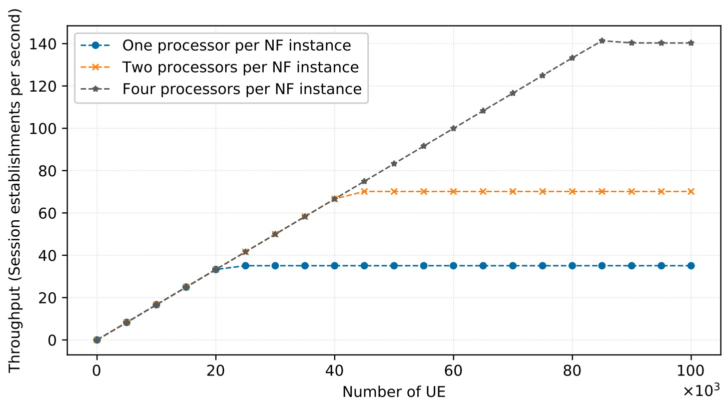


Figure 6-3.: Throughput of session establishment 5GNSL for the basic configuration.

Fig. **6-4** presents the throughput for configurations (1,1,1), (2,1,1), (2,2,1), and (2,2,2). In this evaluation, it is used two processors in each VM that hosts a VNF. As expected, in every configuration, the throughput increases with the number of users until it reaches its maximum. In this set of configurations, the increase in the number of NF instances produces an improvement in the throughput. Configurations (2,1,1), (2,2,1), and (2,2,2) achieved an improvement in the maximum throughput around 14%, 39%, and 101%, respectively, regarding the maximum throughput offered by configuration (1,1,1). To sum up, the throughput supported increases with the number of NF instances.

Fig. **6-5** presents the average response time results for the basic configuration of the session

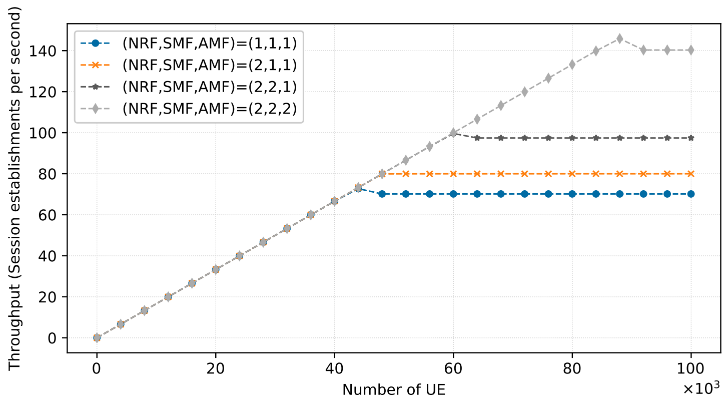


Figure 6-4.: Throughput of session establishment 5GNSL for different configurations.

establishment 5GNSL $(N_{nrf}, N_{smf}, N_{amf}) = (1, 1, 1)$, which is measured when AMF goes through states Amf_2 to Amf_{12} (see AMF component). These results show that this time increases with the number of concurrent users up to a saturation point. This saturation point depends on the number of processors; in particular, configuration (1,1,1) can handle up to 23,000, 42,000, and 85,000 concurrent users with one, two, and four processors before the session establishment 5GNSL starts to drop incoming requests. If a Slice Architect sets, for instance, the target average response time to 200 ms, the session establishment 5GNSL can handle nearly 80,000 concurrent users using four processors per NF instance. Summarizing, the results mentioned above confirm the correlation between the average response time and the number of processors; many processors lead to support a high number of concurrent users in a short time.

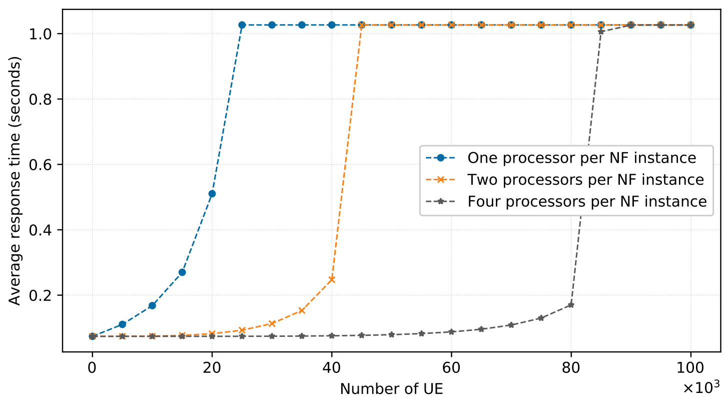


Figure 6-5.: Average response time of session establishment for the basic configuration.

Fig. **6-6** presents the average response time results for configurations (1,1,1), (2,1,1), (2,2,1), and (2,2,2) when VMs that host the VNF instances use two processors. According to these results, the average response time improves as NFs scale-out. Specifically, this time is lower in configurations (2,1,1) and (2,2,1) than in basic configuration around 13% and 28%, respectively. However, the maximum average response time in configuration (2,2,2) is as high as in configuration (1,1,1) due to the entry point of the slice is AMF. When AMF is scaled-out, it sends a high number of requests to the other NFs. Thus, SMF and NRF must also be scaled-out to avoid bottlenecks.

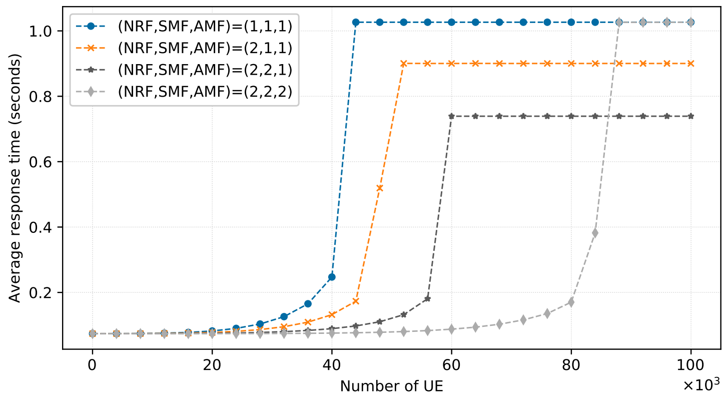


Figure 6-6.: Average response time of session establishment for different configurations.

Fig. **6-7** shows the processor utilization of session establishment 5GNSL for configuration $(N_{nrf}, N_{smf}, N_{amf}) = (1,1,1)$ and two processors allocated per NF instance. Utilization is measured as the population of the second state of processing (*e.g., $Amfp_2$*). NRFP reaches its maximum utilization (1.97) for 45,000 concurrent users, explaining the overload of the session establishment 5GNSL from this point (see Figs. **6-3** and **6-4**). This slice is overloaded from 45,000 users, although AMFP and SMFP are underused, 1.45, and 1.75, respectively. In summary, these results show that if in a sequential chain, an NF becomes a bottleneck, the request processing chain fails thoroughly.

Fig. **6-8** presents the scalability results when configurations $k_1 = (1,1,1)$, $k_2 = (2,1,1)$, $k_3 = (2,2,1)$, $k_4 = (2,2,2)$ use two processors in each VM that hosts a VNF. The scalability results reveal some facts. First, the basic configuration is useful to attend less than 40,000 users because $\psi(k_1, k_2)$, $\psi(k_1, k_3)$, and $\psi(k_1, k_4)$ are less than 1; similar productivity with less cost which is related to the number of NF instances that constitute a configuration. Second, the configuration (2,1,1) is appropriate to attend between $40,000$ and $45,000$ concurrent users because $\psi(k_1, k_2) > 1$; this configuration costs less than upper configurations. Third, the configuration (2,2,1) meets between $45,000$ and $54,000$ concurrent users because $\psi(k_1, k_3) > 1$; this configuration costs less than (2,2,2) configuration. Fourth, the configuration (2,2,2) is useful to attend more than 54,000 concurrent users.

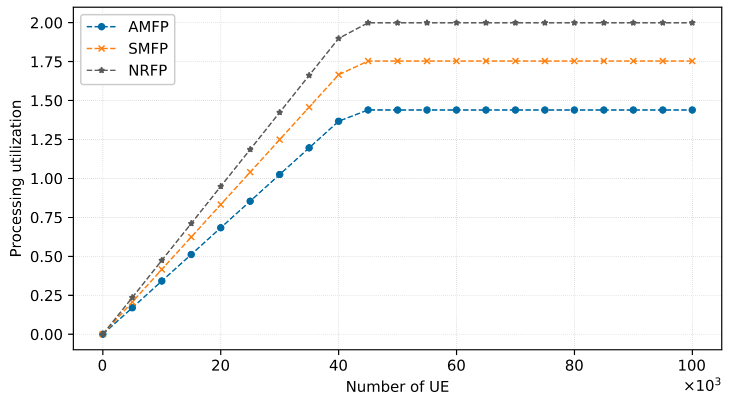


Figure 6-7.: Processor utilization of AMFP, SMFP, and NRFP for the basic configuration.

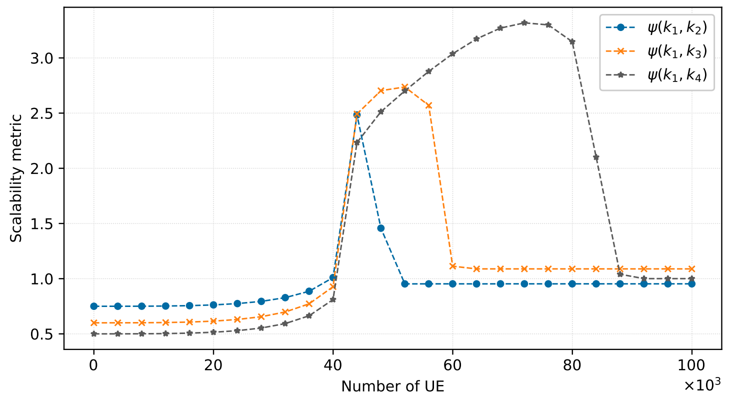


Figure 6-8.: Scalability metric of session establishment 5GNSL for different configurations.

Summarizing, the main insights from the results above described are; first, the method allows NSPs to dimension 5GNSLs. Second, a single NF in saturation state generates a failure in a sequential service chain; thus, for NSLs containing this kind of chain, it is necessary to scale the number of NF instances coordinately. Second, horizontal scaling allows improving the performance of the slice regarding the maximum number of concurrent users, throughput, and average response time. However, a more significant number of instances implies an increasing cost. Therefore, there must be a balance between performance (QoS to be supported) and the number of NF instances.

6.2.4. Validation with Layered Queuing Network

The accuracy of the PEPA-based model is validated by LQN (Franks et al., 2009) since LQN is a well established and accepted model for performance evaluation of distributed systems, such as the core network in 5GNSLs. Accuracy is the difference between the measured obtained by the model and the got by LQN (see Equation 6-4 that follows the definition of percentage relative error proposed in (Tribastone, 2010)). LQN is a model for Extended Queuing Networks, which allows analyzing client-server architectures with nested multiple resource possession in which successive depths of nesting define the layers. Information for constructing LQN models is available in (Woodside, 2013). PEPA can be mapped to LQN by the process-algebraic proposed in (Tribastone, 2010). This process was followed to model in LQN the session establishment 5GNSL.

$$Error(\%) = \left| \frac{PEPA_{metric} - LQN_{metric}}{LQN_{metric}} \right| \times 100 \tag{6-4}$$

Fig. **6-9** presents the LQN model for the session establishment 5GNSL, including tasks, processors, entries, calls, and demands. Tasks (large parallelograms) are interacting entities (*e.g.,* software services) that carry out operations defined by their entries (services). PEPA components in NFs are mapped as Tasks. A task has a host processor (ovals) that models the computational resource used to carry out service operations; PEPA processing entities are mapped as LQN processors. Each processor has a queue, a discipline for executing its tasks (*e.g.,* First-In-First-Out), and a multiplicity (noted as $< N_{nf} >$); this multiplicity represents the number of NF instances in the horizontal scaling. UE is mapped as a Reference task that does not receive any request. In this validation, UE is a load generator that cyclically creates requests for the AMF task.

A task has one or more entries (smaller parallelograms), representing different operations it may perform. Operations served by NFs (*i.e.,* session establishment, session management context creation, and discovery) are mapped as entries. The session establishment entry performs six activities (rectangles) that model the fork between regular and emergency service requests and operate in sequence, as indicated by the arrows. The activities *discover SMF* and *create smc* call the entries in NRF and SMF, respectively. A call may be synchronous, asynchronous, or forwarding. Synchronous communication in PEPA is mapped as synchronous calls between tasks UE, AMF, SMF, and NRF. LQN demands by the total average amounts of host processing and the average number of calls for service operations required to complete an entry. Service rates are mapped as time demands.

For solving the LQN model of the session establishment 5GNSL, it is used the analytical solver LQNS proposed by Franks et al. (2013). In LQN, the response time of an entry is the time spent answering a single request; it includes the running time and the blocking time (waiting for a processor or waiting for nested lower services to complete). The utilization of a single-threaded task is the fraction of time that such task is busy (executing or blocked), meaning not idle. A multi-threaded task may have several services underway at once, and

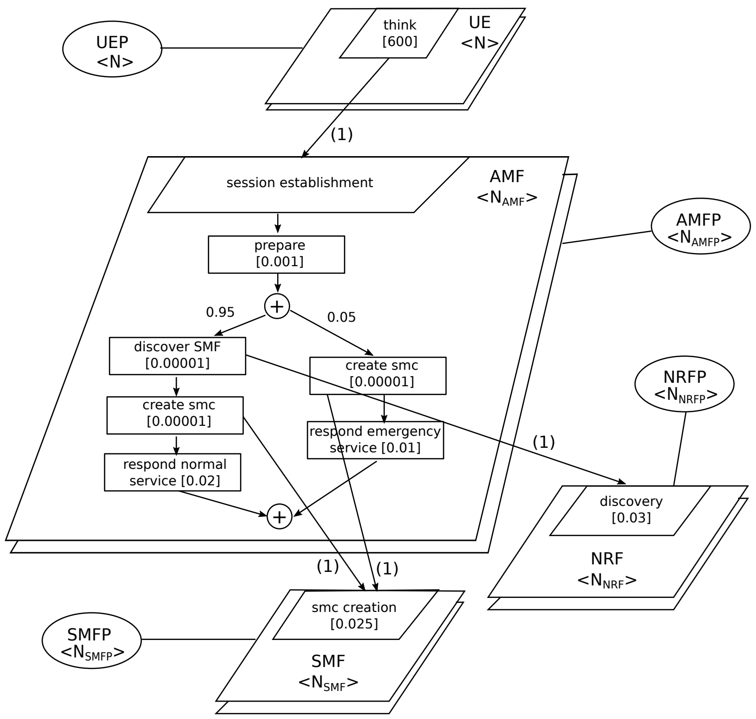


Figure 6-9.: LQN model of the session establishment 5GNSL.

its utilization is the mean number of busy threads. The throughput is the number of service requests of an entry served by a task in a time unit. Summarizing, average response time, throughput, and processor utilization match with the performance metrics defined for the PEPA modeling (see Subsection 6.2.2).

Table **6-3** presents the validation results averaged over five independent runs for configurations $(N_{nrf}, N_{smf}, N_{amf})$ = (1,1,1), (2,1,1), (2,2,1), and (2,2,2) for 20,000 UEs and 80,000 UEs. The evaluation results corroborate the accuracy of the method. It measures performance metrics with negligible difference in comparison to LQN. The most significant differences occur in the average response time measures, mainly when the number of users is close to the overload point, as in 80,000 UE in configuration (2,2,2). These differences are because the fluid approximation used in the PEPA-based model presents a more pronounced transition to overload than the LQN Solver. Readers can found an exhaustive comparison between PEPA and LQN in (Tribastone, 2010, 2013).

6.3. Case study: User Registration V2X 5GNSL

This case study aims two-folds. First, presenting the use of the method by modeling the user registration V2X 5GNSL. Second, analyzing an auto-scaler functionality to dimension

Table 6-3.: Model validation (* 20,000 UE, ** 80,000 UE, relative error in %).

Config.	Throughput			Average response time			AMFP Utilization			SMFP Utilization			NRFP Utilization		
	PEPA	LQN	Error	PEPA	LQN	Error	PEPA	LQN	Error	PEPA	LQN	Error	PEPA	LQN	Error
(1, 1, 1)*	33.3	33.2	0.30	0.0746	0.0827	9.79	0.7004	0.6998	0.09	0.8331	0.8323	0.10	0.9498	0.9489	0.09
(1, 1, 1)**	70.2	70.1	0.14	1.0263	1.0265	0,02	1.4745	1.4742	0.02	1.7538	1.7534	0.02	1.9993	1.9989	0.02
(2, 1, 1)*	33.3	33.2	0.30	0.0746	0.0782	4.60	0.7004	0.6998	0.10	0.8332	0.8324	0.10	0.9498	0.9489	0.10
(2, 1, 1)**	79.9	80.0	0.13	0.9003	0.9002	0.01	1.6808	1.6810	0.01	1.9992	1.9994	0.01	2.2790	2.2793	0.01
(2, 2, 1)*	33.3	33.2	0.30	0.0746	0.0757	1.45	0.7004	0.6998	0.09	0.8331	0.8324	0.08	0.9498	0.9489	0.09
(2, 2, 1)**	94.9	95.1	0.21	0.7581	0.7570	0.15	1.9962	1.9991	0.15	2.3742	2.3777	0.15	2.7066	2.7106	0.15
(2, 2, 2)*	33.3	33.2	0.30	0.0746	0.0745	0.13	0.7004	0.6998	0.10	0.8332	0.8324	0.10	0.9498	0.9489	0.10
(2, 2, 2)**	133.3	133.2	0.08	0.0746	0.1523	51.0	2.8021	2.8014	0.02	3.3328	3.3318	0.03	3.7995	3.7983	0.03

the capacity of this 5GNSL. It is also used the PEPA Eclipse plugin running on an Intel Core i5 PC (1.7 GHz) with 4 GB of RAM.

6.3.1. Modeling

Consider the V2X 5GNSL designed by Campolo et al. (2018). V2X aims at supporting advanced driving assistance services, such as cooperative driving based on sensors data and driving intentions (3GPP, 2018b). Also, consider the analysis of the registration procedure that is supported by the service layer depicted in Fig. **6-10**. For registration to the network slice, the Vehicular User Equipment (VUE) provides the network slice selection information to AMF. Then, AMF cooperates with the Network Slice Selection Function (NSSF) to perform the slice selection. Once NSSF has validated the subscriber information, VUE is authenticated cooperatively by AUSF and the Unified Data Management (UDM). After successful authentication, VUE can use network slice services. It is noteworthy that a V2X network slice needs multiple NF instances to avoid overloading, which can increase service latency. In this case study, it is reused the specification of the PEPA-based model of NRF (Listing 6.4) for the discovery of AUSF. NRF allows associating the most appropriate authentication method at run time, such as EAP-AKA or EAP-AKA', which is useful for 3GPP and non-3GPP accesses.

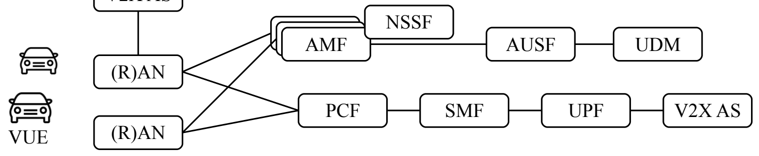


Figure 6-10.: Service layer of the V2X 5GNSL.

As in the previous case study, the modeling includes: *1)* the service rates of the operations performed by NFs (*i.e.,* VUE, AMF, NSSF, NRF, AUSF, and UDM) and the processors of

VMs, *2)* the PEPA components of NFs and processors; and *3)* the system model of the user registration V2X 5GNSL. In particular, the registration procedure is modeled without the Policy Charging Function (PCF). It is considered that PCF affects the registration performance negligibly due to charging rules are relatively straightforward compared to other core functions. Thus, it is assumed that the operation of PCF does not significantly impact the performance of the slice registration process.

Service rates. Table **6-4** presents the service rates (r) in VUE, AMF, NSSF, NRF, AUSF, and UDM. It also introduces the service rates in processors of VMs that host such NFs.

Table 6-4.: Service rates for modeling PEPA actions in the V2X 5GNSL.

Service rate	Description
r_{iat}	$1/r_{iat}$ (inter-arrival time) is the time that VUE interposes between successive requests of registration.
r_r	$1/r_r$ is the average time demand for preparing a response message by AMF.
r_s	$1/r_s$ is the average time that takes the selection of a slice.
r_d	$1/r_d$ is the average time that takes the discovery service performed by NRF.
r_a	$1/r_a$ is the average time that takes authentication in AUSF.
r_{rd}	$1/r_r$ is the average time that takes the recovery of vue data.
r_p	$1/r_p$ is the average execution demand on VM's processors.
v	v is a high service rate that models almost instantaneous service calls and replies between NFs; *e.g.,* AMF calling the AUSF's authentication service.

PEPA components. Next paragraphs describe the model of VUE, AMF, NSSF, AUSF, and UDM as PEPA components and the model of the processors of VMs. In this case study, NRF (Listing 6.4) and NFRP (Listing 6.1) models are not presented because they are similar to those described in the previous case study.

Component *VUE* models VUE requesting the registration to the V2X 5GNSL (see Listing 6.10). Between successive requests, VUE interposes an IAT defined by r_{iat} representing the duration that VUE remains registered to a previous network slice. VUE has two states. The first one (Vue_1) gets access to the processor using the action (get_{vuep}, r_p), and then, performs the action $(stayRegistered, r_{iat})$. The second state (Vue_2) models the synchronous communication between VUE and AMF using the actions of request (req_{reg}, v) and response (rep_{reg}, v).

$$Vue_1 \overset{def}{=} (get_{vuep}, r_p).(stayRegistered, r_{iat}).Vue_2$$
$$Vue_2 \overset{def}{=} (req_{reg}, v).(rep_{reg}, v).Vue_1$$

Listing 6.10: VUE

Component *AMF* processes registration requests from VUE. The action sequence is defined for a set of states Amf_i, $i = \{1, ..., 9\}$ as follows. Amf_1 models the service request. Amf_2 starts a call to NSSF for slice selection. Amf_3 models the synchronous communication between AMF and , captionpos=bNSSF. Amf_4 starts a call to NRF for discovering the appropriate AUSF. Amf_5 represents the synchronous communication between AMF and NRF. Amf_6 starts a call to AUSF for authentication of VUE. Amf_7 models the synchronous communication between AMF and AUSF. Amf_8 prepares a message response to send back to VUE once the registration procedure finishes. Amf_9 models the registration reply to VUE. Listing 6.11 presents the PEPA-based model for AMF.

$$Amf_1 \overset{def}{=} (req_{reg}, v).Amf_2$$
$$Amf_2 \overset{def}{=} (get_{amfp}, r_p).(call_{nssf}, v).Amf_3$$
$$Amf_3 \overset{def}{=} (req_{ss}, v).(rep_{ss}, v).Amf_4$$
$$Amf_4 \overset{def}{=} (get_{amfp}, r_p).(call_{nrf}, v).Amf_5$$
$$Amf_5 \overset{def}{=} (req_d, v).(rep_d, v).Amf_6$$
$$Amf_6 \overset{def}{=} (get_{amfp}, r_p).(call_{ausf}, v).Amf_7$$
$$Amf_7 \overset{def}{=} (req_{auth}, v).(rep_{auth}, v).Amf_8$$
$$Amf_8 \overset{def}{=} (get_{amfp}, r_p).(respond, r_r).Amf_9$$
$$Amf_9 \overset{def}{=} (rep_{reg}, v).Amf_1$$

Listing 6.11: AMF

Component *NSSF* models the network slice selection service offered by NSSF using three states. The state $Nssf_1$ defines the request from AMF. The state $Nssf_2$ represents the access to the processor and the local operation that performs the network slice selection. The state $Nssf_3$ models the response to AMF. Listing 6.12 presents the PEPA-based model for NSSF.

$$Nssf_1 \overset{def}{=} (req_{ss}, v).Nssf_2$$
$$Nssf_2 \overset{def}{=} (get_{nssfp}, r_p).(selectSlice, r_s).Nssf_3$$
$$Nssf_3 \overset{def}{=} (rep_{ss}, v).Nssf_1$$

Listing 6.12: NSSF

Component *AUSF* handles authentication requests. The action sequence is defined for a set of states $Ausf_i$, $i = \{1, ..., 5\}$. $Ausf_1$ models the service request. $Ausf_2$ starts a call to UDM for retrieving administrative data of VUE. $Ausf_3$ models the synchronous communication between AUSF and UDM for data retrieving. $Ausf_4$ represents the local operation that authenticates VUE. $Ausf_5$ is the service reply. Listing 6.13 presents the PEPA-based model for AUSF.

$$
\begin{aligned}
&Ausf_1 \overset{def}{=} (req_{auth}, v).Ausf_2\\
&Ausf_2 \overset{def}{=} (get_{ausfp}, r_p).(call_{udm}, v).Ausf_3\\
&Ausf_3 \overset{def}{=} (req_{rd}, v).(rep_{rd}, v).Ausf_4\\
&Ausf_4 \overset{def}{=} (get_{ausfp}, r_p).(authenticateVue, r_a).Ausf_5\\
&Ausf_5 \overset{def}{=} (rep_{auth}, v).Ausf_1
\end{aligned}
$$

Listing 6.13: AUSF

Component *UDM* models the data recovery service. UDM is modeled using actions that represents data recovery requests and replies, which are performed in cooperation with AUSF. These actions are (req_{rd}, v) and (rep_{rd}, v), for request and reply, respectively. In addition, a state (Udm_2) defines the access to the processor and the local operation that performs the data recovery service. Udm_2 is modeled using the sequential actions (get_{udmp}, r_p) and $(recoverData, r_{rd})$. Listing 6.14 presents the PEPA-based model for UDM.

$$
\begin{aligned}
&Udm_1 \overset{def}{=} (req_{rd}, v).Udm_2\\
&Udm_2 \overset{def}{=} (get_{udmp}, r_p).(recoverData, r_{rd}).Udm_3\\
&Udm_3 \overset{def}{=} (rep_{rd}, v).Udm_1
\end{aligned}
$$

Listing 6.14: UDM

Components *VUEP*, *AMFP*, *NSSFP*, *NRFP*, *AUSFP*, and *UDMP* model the processing entities on which VUE, AMF, NSSF, NRF, AUSF, and UDM execute. The pattern of processing (see Subsection 6.1.2) is used to model these entities, in which each component hast two states. The first state gets access to the processor, whereas the second state performs the actions deployed on the processor. Listings 6.15, 6.16, 6.17, 6.18, and 6.19 present the PEPA-based models for VUEP, AMFP, NSSFP, NRFP, AUSFP, and UDMP, respectively.

$$
\begin{aligned}
&Vuep_1 \overset{def}{=} (get_{vuep}, r_p).Vuep_2\\
&Vuep_2 \overset{def}{=} (stayRegistered, r_{iat}).Vuep_1
\end{aligned}
$$

Listing 6.15: VUEP

$$
\begin{aligned}
&Amfp_1 \overset{def}{=} (get_{amfp}, r_p).Amfp_2\\
&Amfp_2 \overset{def}{=} (call_{nssf}, v).Amfp_1 + (call_{nrf}, v).Amfp_1\\
&\qquad\quad +(call_{ausf}, v).Amfp_1 + (respond, r_r).Amfp_1
\end{aligned}
$$

Listing 6.16: AMFP

$$
\begin{aligned}
&Nssfp_1 \overset{def}{=} (get_{nssfp}, r_p).Nssfp_2\\
&Nssfp_2 \overset{def}{=} (selectSlice, r_s).Nssfp_1
\end{aligned}
$$

Listing 6.17: NSSFP

$$
\begin{aligned}
&Ausfp_1 \overset{def}{=} (get_{ausfp}, r_p).Ausfp_2\\
&Ausfp_2 \overset{def}{=} (call_{udm}, v).Ausfp_1\\
&\qquad\quad +(authenticateVue, r_a).Ausfp_1
\end{aligned}
$$

Listing 6.18: AUSFP

$$
\begin{aligned}
&Udmp_1 \overset{def}{=} (get_{udmp}, r_p).Udmp_2\\
&Udmp_2 \overset{def}{=} (recoverData, r_{rd}).Udmp_1
\end{aligned}
$$

Listing 6.19: UDMP

Overall system model. The entire system of the user registration V2X 5GNSL (see List 6.20) is modeled by using the operator of cooperation (Subsection 2.1.8).

$$
\begin{aligned}
&((((((Vue_1[N_{vue}] \underset{L_1}{\bowtie} Amf_1[N_{amf} \cdot N_{amfp} \cdot N_t]) \underset{L_2}{\bowtie}\\
&Nssf_1[N_{nssf} \cdot N_{nssfp} \cdot N_t]) \underset{L_3}{\bowtie} Nrf_1[N_{nrf} \cdot N_{nrfp} \cdot N_t])\\
&\underset{L_4}{\bowtie} Ausf_1[N_{ausf} \cdot N_{ausfp} \cdot N_t]) \underset{L_5}{\bowtie}\\
&Udm_1[N_{udm} \cdot N_{udmp} \cdot N_t]) \underset{L_6}{\bowtie}\\
&(((((Vuep_1[N_{vuep}] \underset{\emptyset}{\bowtie} Amfp_1[N_{amf} \cdot N_{amfp}]) \underset{\emptyset}{\bowtie}\\
&Nssfp_1[N_{nssf} \cdot N_{nssfp}]) \underset{\emptyset}{\bowtie} Nrfp_1[N_{nrf} \cdot N_{nrfp}]) \underset{\emptyset}{\bowtie}\\
&Ausfp_1[N_{ausf} \cdot N_{ausfp}]) \underset{\emptyset}{\bowtie} Udmp_1[N_{udm} \cdot N_{udmp}]))
\end{aligned}
$$

$$
\begin{aligned}
L_1 &= \{req_{reg}, rep_{reg}\}\\
L_2 &= \{req_{ss}, rep_{ss}\}\\
L_3 &= \{req_d, rep_d\}\\
L_4 &= \{req_{auth}, rep_{auth}\}\\
L_5 &= \{req_{rd}, rep_{rd}\}\\
L_6 &= \{get_{vuep}, stayRegistered, get_{amfp}, call_{nssf}, call_{nrf},\\
&\quad call_{ausf}, respond, get_{nssfp}, selectSlice, get_{nrfp},\\
&\quad discover, get_{ausfp}, call_{udm}, authenticateVue,\\
&\quad get_{udmp}, recoverData\}
\end{aligned}
$$

$\emptyset = \{\}$

Listing 6.20: Overall system model

In List 6.20, N_{vue} is the number of VUE, and N_{amf}, N_{nssf}, N_{nrf}, N_{ausf}, and N_{udm} are the number of available instances of AMF, NSSF, NRF, AUSF, and UDM, respectively. Also, N_{amfp}, N_{nssfp}, N_{nrfp}, N_{ausfp}, and N_{udmp} are the number of processors that are allocated to each instance of AMF, NSSF, NRF, AUSF, and UDM, respectively. It is noteworthy that each processor can handle a set of concurrent threads, which is noted by N_t. Thus, the product $N_{nf} \cdot N_{nfp} \cdot N_t$ represents the total number of threads of an NF. The product $N_{nf} \cdot N_{nfp}$ is the total number of processors allocated to an NF. Also, if a VUE has a single processor, N_{vue} and N_{vuep} are equal.

6.3.2. Results and Analysis

In this experiment, it is considered the automatic scaling of the V2X 5GNSL to face workload variations of Fig. **6-11**, assuming the performance policy: "Each NF (*i.e.,* AMF, NSSF, NRF, AUSF, and UDM) must operate without overloading between 40 and 70 percent of its processor utilization". The following scaling policy may be defined to meet the performance policy afore-mentioned: "The scaling system increases by one instance when the utilization is above the upper threshold (70%). On the other hand, it decreases in one instance when the utilization is less than the lower threshold (40%)". It is assumed that each NF instance uses two processors. Table **6-5** presents the service rates used in this evaluation.

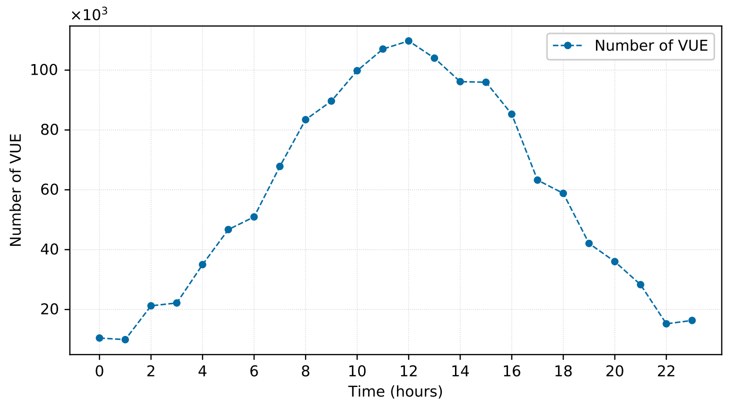


Figure 6-11.: Variable workload (number of VUE) in a period of 24 hours.

Fig. **6-12** shows that applying the scaling policy allows keeping the processor utilization in the user registration V2X 5GNSL into the target range most of the time. At the start and end of the workload pattern, when the number of users is relatively low, the utilization is

Table 6-5.: Service rates used for numerical experimentation.

Service rate	Value	Description
r_{iat}	1/600	Rate between subsequent requests.
r_r	1/0.02	Rate of reply by AMF.
r_s	1/0.025	Rate of slice selection by NSSF.
r_d	1/0.015	Rate of discovery by NRF.
r_a	1/0.03	Rate of authentication by AUSF.
r_{rd}	1/0.001	Rate of data recovery by UDM.
r_p	1/0.00001	Rate of processing on VMs.
v	1/0.00001	Rate of almost instantaneous communication.

less than 40%, which indicates that all NFs need just one instance (see Fig. **6-13**). At some hours (*e.g.,* 4, 5, and 9), the utilization is higher than 70% because the scaling system is reactive. However, since 70% is a conservative value, the NFs of the user registration V2X 5GNSL are not overloaded. It is noteworthy that UDM uses only one instance for the entire experimentation time due to its high rate to process data recovery requests ($r_{rd} = 1/0.001$).

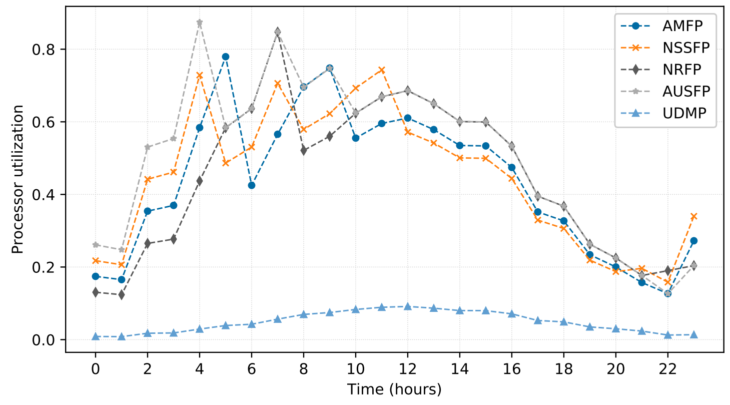


Figure 6-12.: Processor utilization as the scaling policy is applied.

Fig. **6-13** plots the number of NF instances required to comply with the performance policy in the user registration V2X 5GNSL. It is noteworthy that NFs with lower service rates, such as NSSF and AUSF, are scaled to a greater extent. In particular, NSSF and AUSF need up to four instances, NRF up to three instances, NSSF up to two instances, and UDM only one (it has a very high service rate).

Fig. **6-14** presents the average response time of NFs used in the user registration V2X 5GNSL, revealing that this time is low when AMF, NSSF, NRF, AUSF, and UDM are not overloaded. The performance and scalability policies considered in this case study are simple, but they still allow for low response times; the maximum average response time is around $185ms$. A performance policy that sets a more demanding objective in response time will need a more complex scaling mechanism than the here presented.

Fig. **6-15** shows the throughput of the user registration V2X 5GNSL. As expected, the maximum throughput (number of VUE registrations per second) is supported when the slice operates with their highest number of instances. In particular, the slice achieved a maximum of 180 registration per second at 12 hours with the following configuration: 3 AMF, 4 NSSF, 4 AUSF, 2 NRF, and 1 UDM.

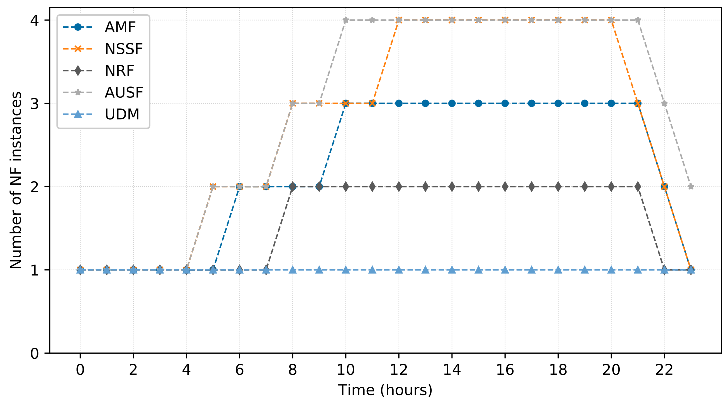


Figure 6-13.: Number of NF instances in horizontal scaling to accomplish the performance policy.

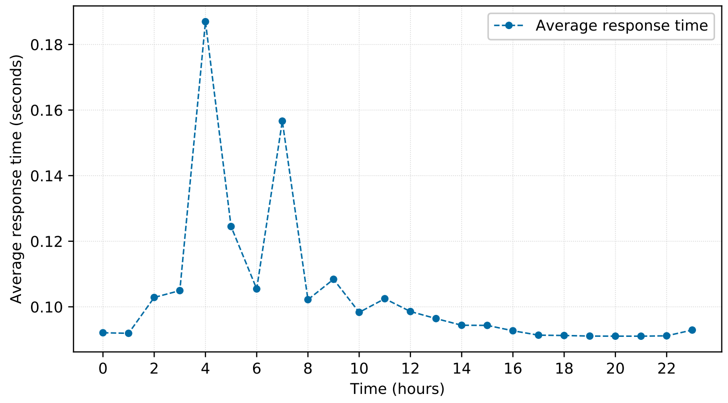


Figure 6-14.: Average response time as the scaling policy is applied.

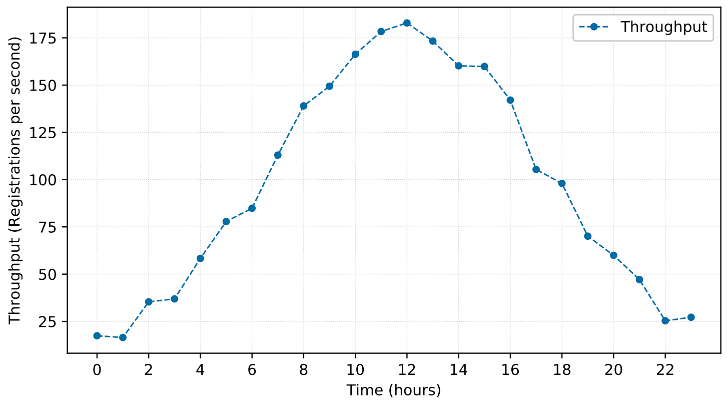


Figure 6-15.: Throughput (registrations per second) as the scaling policy is applied.

Summarizing, the main insights from the results above described are; first, the proposed method allows NSPs to implement automatic scaling in 5GNSLs. Second, the service rates are significant to NFs since, to implement policies, they can determine their number of instances as in the UDM case.

6.4. Final Remarks

This chapter presented a PEPA-based method to analyze the scalability and performance of core NSs in 5GNSL. This method allows specifying 5G core NFs as sequential process components and their interaction forming a cooperation process. New composite structures based on PEPA has been introduced which are useful to model and evaluate 5GCN procedures. This chapter illustrated the use of the proposed method by presenting the modeling and assessment regarding scalability and performance metrics of the session establishment 5GNSL and the user registration process in a V2X 5GNSL. The evaluation results in the two case studies corroborated the usefulness of the method to model 5G core NSs and dimension the capacity of 5GNSLs. Furthermore, the accuracy validation results corroborated that the PEPA-based method measures performance metrics (throughput, average response time, and processor utilization) with negligible difference regarding LQN.

7. Conclusions and Future Work

7.1. Answer for the Fundamental Questions

This thesis addressed the research question: which, how, and when scale NFs in NFV?. To answer this question, this thesis conducted investigations on performance and scalability in two NFV-based networks: EPC and 5GCN. First, this thesis investigated the scalability and performance characterization in a practical NFV-based EPC. Second, a scaling mechanism based on RL for an NFV-based EPC was proposed and evaluated by simulations. Third, a scaling mechanism based on MARL for an NFV-based 5GCN was proposed and evaluated by simulations. Fourth, this thesis analyzed the scalability and performance in the 5GCN theoretically. These investigations are developed in the following chapters.

Chapter 3 presented a scalability and performance characterization of an NFV-based EPC. The corresponding characterization results revealed that horizontal and vertical scaling can improve latency and throughput, enabling to handle variations in the number of concurrent users. Also, this characterization allowed determining which NFs (*e.g.,* MME, SGW, and PGW) must scale and how to scale them (*e.g.,* horizontal, vertical, and elastic). Using the scalability results above-mentioned, Chapter 3 also presented an automatic mechanism to scale an NFV-based EPC, which uses static threshold rules to decide when to trigger scaling actions. The EPC and its scaling mechanism were implemented and tested in a data center, verifying its qualities to manage workload and performance variations.

Chapter 4 presented an adaptive scaling mechanism for managing performance variations in an NFV-based EPC. The mechanism autonomously selects the instance number of the MME, which is the function that carries out the user registration and handover procedures in EPC. The mechanism operates by a RL agent that uses the Q-Learning method and GPs for improving a horizontal scaling policy. Tests based on simulations reveal that this mechanism outperforms scaling based on static threshold rules.

Chapter 5 presented a cooperative scaling mechanism for managing workload variations in the 5GCN control plane. This mechanism makes decisions of scaling using MARL seeking to maintain the average response time below a target. Chapter 5 evaluates three types of agents: *i)* Single AMF agent learning how to scale while other functions such as SMF and NRF remain static and oversized. *ii)* Independent agents (AMF, SMF, and NRF) that learn how to scale independently. *iii)* Cooperative agents that make scaling decisions jointly. The evaluation of the scaling mechanism using these agents was carried out by simulations. Evaluation results reveal that cooperative agents outperform the other two cases.

Chapter 6 presented the scalability and performance analysis in 5GCN by using PEPA. In particular, two 5G signaling procedures are modeled and evaluated regarding throughput, average response time, and scalability metrics. The first procedure is the Session Establishment in a 5G network slice, and the second one is the User Registration in a V2X network slice. PEPA-based models of these procedures were validated against LQN models, showing good accuracy of the performance analysis carried out. Investigation of Chapter 6 revealed the usefulness of scalability analysis to model 5G core NSs and dimension the capacity of 5G network slices.

7.2. Future Work

Chapter 5 showed advantages of using MARL to implement cooperative scaling in an NFV-based 5GCN. In general, RL multi-agents can be equipped with the intelligence necessary to autonomously learn how to scale complex NSs. However, further investigations should be conducted to improve the scaling mechanism presented. The following lines of work are identified.

- It is necessary to study other cooperative models such as game theory. This future work is justified because agents experience tension between achieving their goal (keeping the average response time below a threshold) and capacity utilization, and therefore, the achievement of the performance goal may be at the expense of less usage in NFs.

- Competitive scaling should be investigated considering two types of agents. Some agents could have as objective the achievement of the performance goal, such as the average response time. On the other hand, other agents could seek the maximization of the capacity.

- It will be relevant to implement and validate the cooperative scaling mechanism proposed in this thesis (*e.g.,* Coop-Scaling) in 5G and beyond networks. These networks will be formed by logical network segments generated by new use cases (*e.g.,* tactile Internet and holographic communications), which will require flexible capacity management and reliability. Hence, scaling capabilities based on MARL can bring advantages to 5G and 6G mobile networks.

A. Appendix: Scientific Production

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity thereby presented. The list of the accepted papers is as follows.

- C. H. T. Arteaga, F. Risso and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an NFV-based EPC," 2017 13th International Conference on Network and Service Management (CNSM), Tokyo, Japan, 2017, pp. 1-7, doi: 10.23919/CNSM.2017.8255982.
- C. H. T. Arteaga, F. B. Anacona, K. T. T. Ortega and O. M. C. Rendon, "A Scaling Mechanism for an Evolved Packet Core Based on Network Functions Virtualization," in IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 779-792, June 2020, doi: 10.1109/TNSM.2019.2961988.
- C. H. T. Arteaga, A. Ordoñez and O. M. C. Rendon, "Scalability and Performance Analysis in 5G Core Network Slicing," in IEEE Access, vol. 8, pp. 142086-142100, 2020, doi: 10.1109/ACCESS.2020.3013597.
- C. H. T. Arteaga, O. M. C. Rendon, "Cooperative Scaling for the 5G Core Network Control Plane," To be submitted to Journal of Network and Computer Applications - Elsevier.

There is other peer-reviewed publication that, although not directly related to this thesis, is linked to the design of scalability solutions.

- J. S. Orduz, G. D. Orozco, C. H. Tobar-Arteaga and O. M. C. Rendon, "μvIMS: A Finer-Scalable Architecture Based on Microservices," 2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium), 2019, pp. 141-148, doi: 10.1109/LCNSymposium47956.2019.9000664.

The papers above-mentioned are available in the next pages.

Bibliography

3GPP. General packet radio service (gprs) enhancements for evolved universal terrestrial radio access network (e-utran) access. Release 16 ts 23.401 v16.1.0, 3GPP, Sophia-Antipolis, France, 2018a.

3GPP. Enhancement of 3gpp support for v2x scenarios. Release 15 tr 22.186 v16.0.0, 3GPP, Sophia-Antipolis, France, 2018b.

3GPP. System architecture for the 5g system. Release 15 tr 23.501 v15.3.0, 3GPP, Sophia-Antipolis, France, 2018c.

3GPP. Procedures for the 5g system. Release 15 tr 23.502 v15.2.0, 3GPP, Sophia-Antipolis, France, 2018d.

3GPP. 5g system; access and mobility management services; stage 3. Release 15 ts 29.518 v15.3.0, 3GPP, Sophia-Antipolis, France, 2019.

S. Abrar, C. K. Loo, and N. Kubota. A multi-agent approach for personalized hypertension risk prediction. *IEEE Access*, 9:75090–75106, 2021. doi: 10.1109/ACCESS.2021.3074791.

O. Adamuz-Hinojosa, J. Ordonez-Lucena, P. Ameigeiras, J. J. Ramos-Munoz, D. Lopez, and J. Folgueira. Automated network service scaling in nfv: Concepts, mechanisms and scaling workflow. *IEEE Communications Magazine*, 56(7):162–169, July 2018.

I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche. On the scalability of 5g core network: the amf case. In *CCNC 2018*, pages 1–6, Las Vegas, NV, USA, Jan. 2018a. IEEE.

I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin. Improving traffic forecasting for 5G core network scalability: A Machine Learning approach. *IEEE Network Magazine*, 32(6): 42–49, Nov. 2018b.

O. Almutairi and N. Thomas. Performance modelling of the impact of cyber attacks on a web-based sales system. *Electronic Notes in Theoretical Computer Science*, 353:5–20, 2020. doi: 10.1016/j.entcs.2020.09.016.

M. Álvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3):195–266, April 2012.

P. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin. A cloud native solution for dynamic auto scaling of mme in lte. In *International Symposium on Personal, Indoor, and Mobile Radio Communications*, pages 1–7, Montreal, QC, Canada, February 2018. IEEE. doi: 10.1109/PIMRC.2017.8292270.

C. H. T. Arteaga. A 5g core network prototype, 2020. URL `https://github.com/carloshtobar/A5GCoreNetworkPrototype`. Accessed Feb, 2020.

L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

S. Ayoubi et al. Machine learning for cognitive network management. *IEEE Communications Magazine*, 2017.

N. Baldo, M. Miozzo, M. Requena-Esteso, and J. Nin-Guerrero. An open source product-oriented lte network simulator based on ns-3. In *International conference on Modeling, analysis and simulation of wireless and mobile systems*, pages 293–298, Miami, Florida, USA, November 2011. ACM. doi: 10.1145/2068897.2068948.

D. Balla, C. Simon, and M. Maliosz. Adaptive scaling of kubernetes pods. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5, 2020. doi: 10.1109/NOMS47738.2020.9110428.

A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. V. der Merwe, and S. Rangarajan. Scaling the lte control-plane for future mobile access. In *Conference on Emerging Networking Experiments and Technologies*, pages 1–13, Heidelberg, Germany, December 2015. ACM. doi: 10.1145/2716281.2836104.

A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167 (106984):1–40, 2020.

A. Barto, S. Bradtke, U. of Massachusetts at Amherst. Department of Computer, I. Science, and S. Singh. *Real-time Learning and Control Using Asynchronous Dynamic Programming.* COINS technical report. University of Massachusetts at Amherst, Department of Computer and Information Science, 1991.

S. Becker, G. Brataas, and S. Lehrig. *Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications*, volume I. Springer International Publishing, Cham, Switzerland, 1 edition, 2017.

J. Beckett and R. Bradfield. Power efficiency comparison of enterprise-class blade servers and enclosures. Technical report, Dell, 2011.

D. Bhamare, R. Jain, M. Samaka, and A. Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, November 2016.

A. Bilal, T. Tarik, A. Vajda, and B. Miloud. Dynamic cloud resource scheduling in virtualized 5g mobile systems. In *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, 2016a.

A. Bilal, A. Vajda, and T. Tarik. Impact of network function virtualization: A study based on real-life mobile network data. In *International Wireless Communications and Mobile Computing Conference International (IWCMC)*, pages 541–546, Paphos, Cyprus, 2016b.

C. Bouras, P. Ntarzanos, and A. Papazois. Cost modeling for sdn/nfv based mobile 5g networks. In *8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 87–92, Lisbon, Portugal, Oct. 2016. IEEE.

J. C. Burguillo. *Multi-agent Systems. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation.* Cham, 2018.

S. Burroughs. *Towards predictive runtime modelling of Kubernetes microservices.* PhD thesis, The University of Waikato, 2021.

L. Busoniu, R. Babuska, and B. D. Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, Mar 2008.

T. K. Buyakar, A. K. Rangisetti, A. A. Franklin, and B. R. Tamma. Auto scaling of data plane vnfs in 5g networks. In *Conference on Network and Service Management*, volume 13, pages 1–4, Tokyo,Japan, November 2017. IEEE.

F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea. Performance of network virtualization in cloud computing infrastructures: The openstack case. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 132–137, Luxembourg, Luxembourg, 2014.

C. Campolo, A. Molinaro, A. Iera, and F. Menichella. 5g network slicing for vehicle-to-everything services. *IEEE Wireless Communications*, 24(6):38–45, Dec. 2017. ISSN 1536-1284.

C. Campolo, R. D. R. Fontes, A. Molinaro, C. E. Rothenberg, and A. Iera. Slicing on the road: Enabling the automotive vertical through 5g network softwarization. *Sensors*, 18 (12), Dec. 2018. doi: 10.3390/s18124435.

D. Campos, C. Gutierrez, and O. Caicedo. 5g and beyond: Past, present and future of the mobile communications. *IEEE Latin America Transactions*, 19(10):1702–1736, Apr. 2021.

A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

L. Cao, P. Sharma, S. Fahmy, and V. Saxena. Nfv-vital: A framework for characterizing the performance of virtual network functions. In *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 93–99, San Francisco, CA, USA, 2015.

G. Carella, M. Pauls, L. Grebe, and T. Magedanz. An extensible autoscaling engine (ae) for software-based network functions. In *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Palo Alto, CA, USA, 2016.

R. V. Chandra and B. S. Varanasi. *Python Requests Essentials*. Packt Publishing, 2015. ISBN 1784395412, 9781784395414.

P. Chen and C. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, August 2014.

S. Chen, J. Hu, Y. Shi, Y. Peng, J. Fang, R. Zhao, and L. Zhao. Vehicle-to-everything (v2x) services supported by lte-based systems and 5g. *IEEE Communications Standards Magazine*, 1(2):70–76, 2017. doi: 10.1109/MCOMSTD.2017.1700015.

W. Chen. *Formal Modeling and Automatic Generation of Test Cases for the Autonomous Vehicle*. Theses, Université Paris-Saclay, Sept. 2020.

X. Chen, J. Ding, and Z. Lu. A decentralized trust management system for intelligent transportation environments. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–14, 2020. doi: 10.1109/TITS.2020.3013279.

X. Chen, J. Ding, Z. Lu, and T. Zhan. An efficient formal modeling framework for hybrid cloud-fog systems. *IEEE Transactions on Network Science and Engineering*, 8(1):447–462, 2021. doi: 10.1109/TNSE.2020.3040215.

A. Chiha, M. V. der Wee, D. Colle, and S. Verbrugge. Network slicing cost allocation model. *Network and Systems Management*, 28(3):1–33, 2020. doi: 10.1007/s10922-020-09522-3.

M. Chiosi et al. Network functions virtualisation; an introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 1–16, Darmstadt-Germany, 2012.

T. Choi, T. Kim, W. Tavernier, A. Korvala, and J. Pajunpaa. Agile management and interoperability testing of sdn/nfv-enriched 5g core networks. *Communications and Experimental Trials with Heterogeneous and Agile Mobile networks*, 40:72–88, february 2018.

S. Chowdhury. *Resource Management in Softwarized Networks*. PhD thesis, University of Waterloo, Ontario, Canada, February 2021.

G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. In *Linear algebra, Markov chains, and queueing models*, pages 145–191. Springer, 1993.

Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2016 2021. White paper, 2017. URL `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`. Accessed April 19, 2018.

Cisco. Cisco annual internet report (2018-2023). White paper, 2020. URL `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`. Accessed April 23, 2021.

E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza. Elasticity in cloud computing: a survey. *Annals of telecommunications*, 70:289–309, August 2015.

L. Cui, S. Yang, F. Chen, Z. Ming, N. Lu, and J. Qin. A survey on application of machine learning for internet of things. *International Journal of Machine Learning and Cybernetics*, 9(8):1399–1417, 2018. doi: 10.1007/s13042-018-0834-5.

M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. 2010. ISBN 978-3-86644-569-7.

M. Di Mauro, G. Galatro, F. Postiglione, and M. Tambasco. Performability of network service chains: Stochastic modeling and assessment of softwarized ip multimedia subsystem. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2021. doi: 10.1109/TDSC.2021.3082626.

S. Donatelli, M. Ribaudo, and J. Hillston. A comparison of performance evaluation process algebra and generalized stochastic petri nets. In *6th International Workshop on Petri Nets and Performance Models*, Durham, NC, USA, 1995. doi: 10.1109/PNPM.1995.524326.

S. Dutta, T. Taleb, and A. Ksentini. Qoe-aware elasticity support in cloud-native 5g systems. In *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, 2016.

ETSI. Network functions virtualisation (nfv); architectural framework, 2013. URL `http://www.etsi.org`.

ETSI. Network functions virtualisation (nfv); management and orchestration, 2014a. URL `http://www.etsi.org`.

ETSI. Network functions virtualisation (nfv); nfv performance and portability best practises, 2014b. URL `http://www.etsi.org`.

ETSI-GS-AFI. Autonomic network engineering for the self-managing future internet (afi); generic autonomic network architecture, 2013. URL `http://www.etsi.org`.

Z. Fadlullah et al. State-of-the-art deep learning: Evolving machine intelligence toward tomorrowâ€™s intelligent network traffic control systems. *IEEE Communications Surveys & Tutorials*, 19(4):2432–2455, May 2017.

M. U. Farooq, A. Shakoor, and A. B. Siddique. An efficient dynamic round robin algorithm for cpu scheduling. In *International Conference on Communication, Computing and Digital Systems*, pages 244–248, Islamabad, Pakistan, March 2017. IEEE.

W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *International Symposium on Performance Analysis of Systems and Software*, pages 171–172, Philadelphia, PA, USA, April 2015. IEEE.

R. Fontes, S. Afzal, S. Brito, M. Santos, and C. Esteve Rothenberg. Mininet-WiFi: emulating Software-Defined wireless networks. In *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*, Barcelona, Spain, Nov. 2015.

H. Fourati, R. Maaloul, and L. Chaari. A survey of 5g network systems: challenges and machine learning approaches. *International Journal of Machine Learning and Cybernetics*, 12(2):385–431, 2021. doi: 10.1007/s13042-020-01178-4.

G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35 (2):148–161, Mar-Apr 2009. doi: https://doi.org/10.1109/TSE.2008.74.

G. Franks, P. Maly, M. Woodside, D. C. Petriu, A. Hubbard, and M. Mroz. Layered queueing network solver and simulator user manual. Technical Report 11145, Ottawa, Ontario, Canada, 2013.

J. Garay, J. Matias, J. Unzilla, and E. Jacob. Service description in the nfv revolution:trends, challenges and a way forward. *IEEE Communications Magazine*, 54(3):68–74, March 2016.

S. K. Gowtam Peesapati, M. Olsson, M. Masoudi, S. Andersson, and C. Cavdar. Q-learning based radio resource adaptation for improved energy performance of 5g base stations. In *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2021.

M. Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 1st edition, 2014. ISBN 1449372627, 9781449372620.

J. R. Gunasekaran, P. Thinakaran, N. Chidambaram, M. T. Kandemir, and C. R. Das. Fifer: Tackling underutilization in the serverless era. *arXiv preprint arXiv:2008.12819*, 2020.

W. Hahn and B. Gajic. Gw elasticity in data centers: Options to adapt to changing traffic profiles in control and user plane. In *International Conference on Intelligence in Next Generation Networks*, pages 16–22, Paris, France, April 2015.

B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network functions virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, February 2015.

D. Harutyunyan, R. Behravesh, and N. Slamnik-KrijeÅ¡torac. Cost-efficient placement and scaling of 5g core network and mec-enabled application vnfs. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 241–249, 2021.

H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE Network*, 28:18–26, December 2014.

P. Heidari and A. Kanso. Qos assurance through low level analysis of resource utilization of the cloud applications. In *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 228–235, San Francisco, CA, USA, 2016.

S. Henning and W. Hasselbring. How to measure scalability of distributed stream processing engines? In *Companion of the ACM/SPEC International Conference on Performance Engineering*, pages 85–88, 2021.

J. Hillston. Pepa - performance enhanced process algebra. Technical report, dept. of computer science, University of Edinburgh, March 1993.

J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

J. Hillston, M. Tribastone, and S. Gilmore. Stochastic Process Algebras: From Individuals to Populations. *The Computer Journal*, 55(7):866–881, 09 2011. ISSN 0010-4620. doi: 10.1093/comjnl/bxr094.

W. Hou, H. Wen, H. Song, W. Lei, and W. Zhang. Multi-agent deep reinforcement learning for task offloading and resource allocation in cybertwin based networks. *IEEE Internet of Things Journal*, pages 1–1, 2021. doi: 10.1109/JIOT.2021.3095677.

F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain. Machine learning for resource management in cellular and iot networks: Potentials, current solutions, and open challenges. *IEEE Communications Surveys Tutorials*, 22(2):1251–1275, 2020. doi: 10.1109/COMST.2020.2964534.

ITU. Tmn management functions, 2000. URL `http://www.itu.int`.

ITU. Definitions of terms related to quality of service, 2008. URL `http://www.itu.int`.

ITU. Imt vision – framework and overall objectives of the future development of imt for 2020 and beyond, 2015. URL `http://www.itu.int`.

P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, June 2000. ISSN 2161-9883. doi: 10.1109/71.862209.

W. John et al. Research directions in network service chaining. In *IEEE SDN for Future Networks and Services (SDN4FNS)*, Trento, Italy, 2013.

J. Kempf, B. Johansson, S. Pettersson, H. LÃ¼ning, and T. Nilsson. Moving the mobile evolved packet core to the cloud. In *International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 784–791. IEEE, October 2012.

H.-G. Kim, D.-Y. Lee, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong. Machine learning-based method for prediction of virtual network function resource demands. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 405–413, 2019. doi: 10.1109/NETSOFT.2019.8806687.

B. Kitchenham. Procedures for performing systematic reviews. Joint technical report, Keele University, Keele, UK, 2004.

M. Kouzehgar, M. Meghjani, and R. Bouffanais. Multi-agent reinforcement learning for dynamic ocean monitoring by a swarm of buoys. In *Global Oceans 2020: Singapore – U.S. Gulf Coast*, pages 1–8, 2020. doi: 10.1109/IEEECONF38699.2020.9389128.

L. Kuang, J. Zheng, K. Li, and H. Gao. Intelligent traffic signal control based on reinforcement learning with state reduction for smart cities. *ACM Trans. Internet Technol.*, 21(4), July 2021. ISSN 1533-5399. doi: 10.1145/3418682.

J. Kumar, A. K. Singh, and R. Buyya. Self directed learning based workload forecasting model for cloud resource management. *Information Sciences*, 543:345–366, 2021. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2020.07.012.

L. Li and S. Shen. End-to-end qos performance management across lte networks. In *Network Operations and Management Symposium (APNOMS)*, Taipei, Taiwan, 2011.

X. Liao, W. Li, Q. Xu, X. Wang, B. Jin, X. Zhang, Y. Wang, and Y. Zhang. Iteratively-refined interactive 3d medical image segmentation with multi-agent reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

B. Liu, R. Buyya, and A. N. Toosi. A fuzzy-based auto-scaler for web applications in cloud computing environments. In *International Conference on Service-Oriented Computing*, pages 797–811. Springer, 2018.

Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung. Blockchain and machine learning for communications and networking systems. *IEEE Communications Surveys Tutorials*, 22 (2):1392–1431, 2020. doi: 10.1109/COMST.2020.2975911.

Z. Luo and C. Wu. An online algorithm for vnf service chain scaling in datacenters. *IEEE/ACM Transactions on Networking*, 28(3):1061–1073, 2020. doi: 10.1109/TNET. 2020.2979263.

A. Maca, J. Daza, and O. Caicedo. Implementation and deployment of the horizontal and vertical scaling of b-vepc under aws, 2019. URL `https://github.com/AndresMaca/AWS-VEPC-PUBLIC`. Accessed June, 2019.

Mausam and A. Kolobov. *Planning with Markov Decision Processes.* 2012.

A. Mestres et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, July 2017.

R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba. Network function virtualization, state-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, Firstquarter 2016a.

R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba. Network function virtualization, state-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, Firstquarter 2016b.

H. Moens and F. Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *10th International Conference on Network and Service Management (CNSM)*, Rio de Janeiro, Brazil, 2014.

M. S. Mollel, A. I. Abubakar, M. Ozturk, S. F. Kaijage, M. Kisangiri, S. Hussain, M. A. Imran, and Q. H. Abbasi. A survey of machine learning applications to handover management in 5g and beyond. *IEEE Access*, 9:45770–45802, 2021. doi: 10.1109/ACCESS. 2021.3067503.

J. Moysen and L. Giupponi. From 4g to 5g: Self-organized network management meets machine learning. *Computer Communications*, 129:248–268, 2018.

G. Neelakantam, D. D. Onthoni, and P. K. Sahoo. Reinforcement learning based passengers assistance system for crowded public transportation in fog enabled smart city. *Electronics*, 9(9), 2020.

NGMN Alliance. Description of network slicing concept. Final deliverable, 2016.

T. Nguyen, Y. Yeom, T. Kim, D. Park, and S. Kim. Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors*, 20(16), 2020. doi: 10.3390/s20164621.

N. Nikaein and S. Krea. Latency for real-time machine-to-machine communication in lte-based system architecture. In *European Wireless-Sustainable Wireless Technologies*, pages 1–6, Vienna, Austria, July 2011.

E. Oliveira, A. Viana, K. Naveen, and C. Sarraute. Mobile data traffic modeling: Revealing temporal facets. *Computer Networks*, 112, 10 2014.

H. S. Oluwatosin. Client-server model. *IOSRJ Comput. Eng*, 16(1):2278–8727, 2014.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A. Perrusquía, W. Yu, and X. Li. Redundant robot control using multi agent reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1650–1655, 2020. doi: 10.1109/CASE48305.2020.9216774.

S. G. Popescu, D. J. Sharp, J. H. Cole, K. Kamnitsas, and B. Glocker. Distributional gaussian process layers for outlier detection in image segmentation. In *International Conference on Information Processing in Medical Imaging*, pages 415–427. Springer, 2021.

J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. A. Maldonado, and J. M. Lopez-Soler. Modeling and dimensioning of a virtualized mme for 5g mobile networks. *IEEE Transactions on Vehicular Technology*, 66:4383 – 4395, May 2017a.

J. Prados-Garzon, A. Laghrissi, M. Bagaa, and T. Taleb. A queuing based dynamic auto scaling algorithm for the lte epc control plane. In *Globecom 2018*, pages 1–7, Abu Dhabi, United Arab Emirates, Dec. 2018.

J. Prados-Garzon et al. Modeling and dimensioning of a virtualized mme for 5g mobile networks. *IEEE Transactions on Vehicular Technology*, 66(5):4383–4395, May 2017b.

G. Premsankar, K. Ahokas, and S. Luukkainen. Design and implementation of a distributed mobility management entity on openstack. In *International Conference on Cloud Computing Technology and Science*, pages 487–490, Vancouver, BC, Canada, November 2015.

K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao. Dynamic resource scaling for vnf over nonstationary traffic: A learning approach. *IEEE Transactions on Cognitive Communications and Networking*, 7(2):648–662, 2021. doi: 10.1109/TCCN.2020.3018157.

A. Rajan et al. Understanding the bottlenecks in virtualizing cellular core network functions. In *IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN)*, Beijing, China, 2015.

W. Rankothge, F. Le, A. Russo, and J. Lobo. Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms. *IEEE Transactions on Network and Service Management*, 14(2):343–356, June 2017.

C. E. Rasmussen. Gaussian processes for machine learning. In Springer, editor, *Advanced Lectures on Machine Learning*. Berlin, 2004.

S. J. A. Raza, A. Dastider, and M. Lin. Survivable robotic control through guided bayesian policy search with deep reinforcement learning, 2021.

Y. Ren, T. Phung-Duc, J.-C. Chen, and Z.-W. Yu. Dynamic auto scaling algorithm (dasa) for 5g mobile networks. In *Global Communications Conference (GLOBECOM)*, pages 1–6, Washington, DC, USA, Dec 2016.

P. E. I. Rivera and M. Erol-Kantarci. Qos-aware load balancing in wireless networks using clipped double q-learning, 2021.

P. Rost, C. Mannweiler, D. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker. Network slicing to enable scalability and flexibility in 5g mobile networks. *IEEE Communications Magazine*, 55(5): 72–79, May 2017. doi: 10.1109/MCOM.2017.1600920.

C. Rotter and T. Van Do. A queueing model for threshold-based scaling of upf instances in 5g core. *IEEE Access*, 9:81443–81453, 2021. doi: 10.1109/ACCESS.2021.3085955.

A. P. Sabelhaus and C. Majidi. Gaussian process dynamics models for soft robots with shape memory actuators. In *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*, pages 191–198, 2021. doi: 10.1109/RoboSoft51838.2021.9479294.

M. N. Sadiku and S. M. Musa. *Performance analysis of computer networks*, volume I. Springer, 1 edition, 2013.

W. S. Sanders, S. Srivastava, and I. Banicescu. Robustness analysis of scaled resource allocation models using the imperial pepa compiler. In *2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 60–67, 2020. doi: 10.1109/ISPDC51135.2020.00018.

V. Sanjana. Virtualized evolved packet core for lte networks. Master's thesis, Indian Institute of Technology, Bombay, Mumbay, India, October 2016.

P. Satapathy, J. Dave, P. Naik, and M. Vutukuru. Performance comparison of state synchronization techniques in a distributed lte epc. In *Conference on Network Function Virtualization and Software Defined Networks*, pages 1–7, Berlin, Germany, November 2017.

S. Schneider, A. Sharma, H. Karl, and H. Wehrheim. Specifying and analyzing virtual network services using queuing petri nets. In *IFIP/IEEE IM 2019, Washington, DC, USA, April 09-11, 2019.*, 2019.

D. Sesto-Castilla, E. Garcia-Villegas, G. Lyberopoulos, and E. Theodoropoulou. Use of machine learning for energy efficiency in present and future mobile networks. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2019. doi: 10.1109/WCNC.2019.8885478.

W. Shafik, M. Matinkhah, and M. N. Sanda. Network resource management drives machine learning: a survey and future research direction. *Journal of Communications Technology, Electronics and Computer Science*, 30:1–15, 2020. ISSN 2457-905X.

N. Sharma, K. Rivera, A. Angelopoulou, and S. Mondesire. Predicting large-scaled, cloud-hosted virtual world resource demands for automated load balancing. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 338–343, 2020. doi: 10.1109/CSCI51800.2020.00064.

R. Shea, F. Wang, H. Wang, and J. Liu. A deep investigation into network performance in virtual machine based cloud environments. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1285–1293, Toronto, ON, Canada, 2014.

E. Siivola, S. Weber, and A. Vehtari. Qualifying drug dosing regimens in pediatrics using gaussian processes. *Statistics in Medicine*, 40(10):2355–2372, 2021.

P. Singh, A. Kaur, P. Gupta, S. S. Gill, and K. Jyoti. Rhas: robust hybrid auto-scaling for web applications in cloud computing. *Cluster Computing*, 24(2):717–737, 2021. doi: 10.1007/s10586-020-03148-5.

S. S. Soliman and B. Song. Fifth generation (5g) cellular and the network for tomorrow: cognitive and cooperative approach for energy savings. *Network and Computer Applications*, 85:84–93, May 2017.

K. C. Sreedhar and M. Swathi. A novel approach to doctor's decision making system using q learning. *European Journal of Molecular & Clinical Medicine*, 7(11):4203–4209, 2021.

R. C. Streijl, S. Winkler, and D. S. Hands. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. *Multimedia Systems*, 22(2):213–227, 2016.

P. Subedi, A. Alsadoon, P. W. C. Prasad, S. Rehman, N. Giweli, M. Imran, and S. Arif. Network slicing: a next generation 5g perspective. *EURASIP Journal on Wireless Communications and Networking*, 2021(102):1–26, 2021. doi: 10.1186/s13638-021-01983-7.

N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad. Survey on sdn based network intrusion detection system using machine learning approaches. *Peer-to-Peer Networking and Applications*, 12(2):493–501, 2019. doi: 10.1007/s12083-017-0630-0.

R. Sutton and A. Barto. *Reinforcement learning: An introduction*. London, 2012.

T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz. Ease: Epc as a service to ease mobile core network deployment over cloud. *IEEE Network*, 29(2):78–88, March 2015.

P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang. Efficient auto-scaling approach in the telco cloud using self-learning algorithm. In *IEEE Global Communications Conference (GLOBECOM)*, San Diego, CA, USA, 2015.

Team-SimPy. Discrete event simulation for python. URL `http://simpy.readthedocs.io`.

A. D. Tijsma, M. M. Drugan, and M. A. Wiering. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016. doi: 10.1109/SSCI.2016.7849366.

C. Tobar, F. Risso, and O. Caicedo. An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc. In *International Conference on Network and Service Management (CNSM)*, pages 1–7, Tokyo, Japan, Nov 2017. IEEE.

B. Tola. Dependability modeling, analysis, and provisioning of nfv-supported services. Doctoral dissertation, Norwegian University of Science and Technology, 2021.

M. Tribastone. Relating layered queueing networks and process algebra models. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 183–194, New York, NY, USA, 2010. doi: 10.1145/1712605.1712634.

M. Tribastone. A fluid model for layered queueing networks. *IEEE Transactions on Software Engineering*, pages 1–14, 2013.

M. Tribastone and S. Gilmore. Rigorous software engineering for service-oriented systems. chapter Scaling Performance Analysis Using Fluid-flow Approximation, pages 486–505. Springer-Verlag, Berlin, Heidelberg, 2011.

M. Tribastone, A. Duguid, and S. Gilmore. The pepa eclipse plugin. *SIGMETRICS Perform. Eval. Rev.*, 36(4):28–33, Mar. 2009. ISSN 0163-5999. doi: 10.1145/1530873.1530880.

R. Trivisonno, M. Condoluci, X. An, and T. Mahmoodi. miot slice for 5g systems: Design and performance evaluation. *Sensors*, 18(2), Feb. 2018. doi: 10.3390/s18020635.

H. Tullberg, P. Popovski, Z. Li, M. A. Uusitalo, A. Hoglund, O. Bulakci, M. Fallgren, and J. F. Monserrat. The metis 5g system concept: Meeting the 5g requirements. *IEEE Communications Magazine*, 54(12):132–139, Dec. 2016. doi: 10.1109/MCOM.2016.1500799CM.

M. Vutukuru, N. Sadagopan, P. Satapathy, and J. K. Dave. A virtualized evolved packet core for lte networks, 2016. URL `https://github.com/networkedsystemsIITB/NFV_LTE_EPC`. Accessed April, 2020.

M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *arXiv:1709.08339*, 2017.

S. Wang, X. Zhang, J. Zhang, J. Feng, W. Wang, and K. Xin. An approach for spatial-temporal traffic modeling in mobile cellular networks. In *27th International Teletraffic Congress (ITC27)*, pages 203–209, Ghent, Belgium, 2015.

X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau. Online vnf scaling in datacenters. In *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 140–147, San Francisco, CA, USA, 2016.

C. Watkins. Learning from delayed rewards, 1989.

C. D. Williams and A. Clark. A case study in capacity planning for pepa models with the pepa eclipse plug-in. *Electronic Notes in Theoretical Computer Science*, 318.

M. Woodside. Tutorial introduction to layered modeling of software performance - edition 4.0. Technical Report 1, Ottawa, Ontario, Canada, 2013.

J. Wu, J. Wang, Q. Chen, Z. Yuan, P. Zhou, X. Wang, and C. Fu. Resource allocation for delay-sensitive vehicle-to-multi-edges (v2es) communications in vehicular networks: A multi-agent deep reinforcement learning approach. *IEEE Transactions on Network Science and Engineering*, 8(2):1873–1886, 2021. doi: 10.1109/TNSE.2021.3075530.

W. Xiang, K. Zheng, and X. S. Shen. *5G Mobile Communications*. Springer Publishing Company, Incorporated, 1st edition, 2017.

F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM Transactions on Networking*, 25 (2):1147–1161, April 2017.

B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang. A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262, 2018. doi: 10.1016/j.comnet.2018.01.021.

F. Yuan, A. Sadovnik, R. Zhang, D. Casenhiser, E. J. Paek, S. O. Yoon, and X. Zhao. A simulated experiment to explore robotic dialogue strategies for people with dementia. *arXiv preprint arXiv:2104.08940*, 2021.

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

B. Zhang, P. Zhang, Y. Zhao, Y. Wang, X. Luo, and Y. Jin. Co-scaler: cooperative scaling of software -defined vnf service function chain. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Palo Alto, CA, USA, 2016.

H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung. Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges. *IEEE Communications Magazine*, 55(8):138–145, Aug. 2017. doi: 10.1109/MCOM.2017.1600940.

X. Zhang, Y. Tian, and X. Zheng. Antenna optimization design based on deep gaussian process model. *International Journal of Antennas and Propagation*, 2020.

X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou. Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters. *IEEE Access*, 9:105192–105204, 2021. doi: 10.1109/ACCESS.2021.3100082.

S. Zou, W. Wang, W. Ni, L. Wang, and Y. L. Tang. Efficient orchestration of virtualization resource in ran based on chemical reaction optimization and q-learning. *IEEE Internet of Things Journal*, 2021. doi: 10.1109/JIOT.2021.3098331.

M. Zukerman. *Introduction to Queueing Theory and Stochastic Teletraffic Models*. 2016.