UNIVERSITY OF MACEDONIA

SCHOOL OF INFORMATION SCIENCES

DEPARTMENT OF APPLIED INFORMATICS

Software-defined Networking Strategies for Efficient Next-generation Applications

Ph.D. Dissertation

of

Sarantis Kalafatidis

Thessaloniki, September 2022

Abstract

Next-generation Internet Applications (NIA), such as smart-cities and multimedia applications like virtual reality, provide the technological foundation and capabilities necessitated for digital transformation in various economic sectors. With the widespread adoption of these applications, network requirements have become increasingly demanding e.g., they require ultra-low delay for rapid decision-making and high throughput for transferring large amounts of data. In this thesis, we focus on the development of efficient network strategies in order to improve the performance of two specific types of NIA: IoT and cloud-native applications.

We focus on two primary network environments that require adaptability to fulfill the needs of the aforementioned NIA: (i) large-scale wireless-based IoT deployments (e.g., smart cities) and (ii) cloud environments that host microservice-based applications with heterogeneous resource demands. In this context, we investigate the potential advantages derived from a synergy of cloud computing, Software-Defined Networking (SDN), Named-Data Networking (NDN), and the microservices paradigm in enhancing NIA performance through efficient traffic steering. Also, we argue that the centralized control feature of SDNs is an appropriate solution for integrating these technologies and creating a highly adaptive network ecosystem.

The research is concentrated around three key areas: (i) improving IoT applications' adaptability to Wireless Mesh Network (WMN) to ensure consistent and reliable performance; (ii) implementing microservices adaptive load balancing, which implements efficient resource allocation and utilization, enabling applications to handle a larger number of simultaneous requests without performance compromise; and (iii) predicting the impact of individual microservices on network and processing resources, thereby support the decisions of resource allocation control mechanisms.

In order to tackle the aforementioned challenges, this research focuses on

the development and evaluation of realistic applications and SDN network environments over WMN, including as well as real indoor and outdoor testbeds. The objective is to promote a framework where NIA can exhibit greater responsiveness and resource efficiency by adapting to the dynamic WMNs and cloud resource requirements.

Keywords: Next Generation Internet Applications, Traffic Steering, Load Balancing, Microservices, Microservices Profiling, Software-defined Networks; Wireless Mesh Networks; Information-Centric Networking; Named Data Networking; Smart-cities, Resource Consumption Prediction

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Eleftherios Mamatas, for his invaluable guidance, which has greatly influenced both my professional and personal growth. Also, my gratitude extends to my co-supervisors, Prof. Alexander Chatzigeorgiou and Prof. Panagiotis Papadimitriou, for their essential insights and recommendations that enriched the quality of my research. Furthermore, I would like to express special thanks to Prof. Ilias Sakellariou and Dr. Sotiris Skaperas for their indispensable advice during my dissertation. Lastly, my deep appreciation goes to my parents for their unwavering support and encouragement throughout my academic journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

The advent of Industry 4.0 has ushered a new era of technological advances which transforms and improves the operation of production line and supply chain management [3]. A crucial aspect of this trend is the development and utilization of Next-generation Internet Applications (NIA), which leverage pioneering technologies (e.g., teleoperated robots and autonomous vehicles) to enhance productivity, efficiency and connectivity in various sectors, ranging from industrial production and multimedia entertainment to automotive and energy sectors. NIA in Industry 4.0 are characterized by their ability to integrate physical and digital systems, incorporating data analysis and artificial intelligence techniques enabling real-time collaboration and decision-making among a wide range of distributed end-user devices. These applications are revolutionizing traditional industries and paving the way for unprecedented levels of automation, optimization, and innovation.

However, to support the evolving demands of these applications, the network requirements have become increasingly critical [4]. For example, the widespread utilization of IoT devices in large-scale environments such as those of industrial plants or smart-city environments requires: high throughput for transferring a large amount of data e.g., from many devices or sensors, and low latency for rapid decision-making in critical use-case scenarios, such as those of health care and autonomous vehicles. On the other hand, multimedia NIA like augmented and virtual reality, often require specific QoS parameters to ensure

acceptable performance levels which are characterized by bandwidth guarantees, minimizing packet loss, maintaining low latency, and managing traffic based on application requirements.

The primary objective of this PhD thesis is to enhance the performance of Next-generation Internet Applications focusing on two specific types: (i) IoT applications and (ii) cloud-native applications. The aim is to develop efficient strategies and methodologies that address the unique network challenges associated with these application domains, enabling efficient traffic steering across their respective ecosystems. This involves the consideration of the network conditions of the environments in which they are operated, namely (i) large-scale wireless-based IoT deployments (e.g., smart cities) and (ii) cloud environments hosting heterogeneous microservice-based applications. In the following subsections, we delve into an analysis of these application domains, outline the network requirements they entail, describe the proposed solutions, and highlight the key topics of this research.

Network challenges of IoT ecosystems

IoT applications are characterized by their ability to connect and communicate with a wide range of devices, utilizing sensor technology to collect data from the physical environment. These applications involve processing and analysis of massive data, facilitating real-time decision-making, enabling remote automation and control. In this context, IoT applications rely on data collected from IoT devices spread over large areas such as industrial or community regions. To this end, wireless connectivity is essential, as it enables communication over wide coverage, offering cost-effective accessibility, scalability, and flexibility. Consequently, the efficient operation of IoT applications, which are typically associated with critical performance requirements (e.g, low-delay), is linked to the robustness of the wireless network infrastructure. Our research specifically investigates smart cities as representative case-study of large-scale IoT deployments that heavily rely on wireless connectivity.

In recent years, various wireless communication technologies (e.g., WiMax, LTE, 5G) have been proposed to support communication of IoT environments, each of them bringing different advantages to the network [5, 6]. In parallel

with the evolution of 5G networks in the domain of wireless connectivity, Wireless Mesh Networking (WMN) has begun to draw increased interest in numerous IoT case studies, particularly in relation to Smart-City network environments. Examples of such studies include Stratford [5], CityLab [1], SmartSantander [7]. WMN can support an extensive range of IoT deployments through multi-hop communication, adding features such as easy deployment, low-cost infrastructure, as well as reliable and efficient connectivity in dynamic environments, such as those involving mobile users.

In summary, some of the benefits offered by WMN in large-scale IoT ecosystems include: (i) the flexible support for efficient interconnection of numerous distributed end-user devices; (ii) unstructured, self-sustaining, and self-configuring topologies; and (iii) the integration of various networking technologies, such as wired, optic-fiber, cellular, and sensor networks. Furthermore, multi-hop communication over WMN can be a supplementary solution to other wireless networking technologies, for example, providing connectivity in areas where 5G coverage falls short, e.g., when the 5G Radio Access Network (RAN) is distant.

Despite these strengths, the WMN backbone network may incorporate regional communication characteristics that present challenges. These include: (i) areas with large numbers of network nodes, requiring efficient network management; (ii) mobile nodes, causing frequent network instabilities, such as topology rearrangements; and (iii) areas with high signal interference. These issues result in unstable topologies, connection failures, and poor-quality communication, which can negatively affect the performance of IoT applications.

Regarding the network layer, many new approaches have emerged in order to address the above network challenges of IoT ecosystems. A typical approach is the Non-IP protocols which, unlike traditional Internet Protocol (IP) approaches, do not rely on IP addresses for data transmission, providing alternative methods for data communication which can increase efficiency and adaptability in complex IoT network environments. One of these solutions is the Named-Data Networking (NDN) [8], which belongs to Information-Centric Networking (ICN) architecture. NDN is identified as a viable approach for fulfilling the needs of IoT applications, providing network performance improvement and reliability. The architecture of NDN is significant for two main

reasons: (i) it makes the retrieval of content from the network easier, as NDN packets hold data names instead of traditional IP addresses; (ii) it has the capability to reduce communication overhead thanks to its in-network caching features.

However, the utilization of NDN in heavily populated WMN (e.g, Smart-City networks) requires the incorporation of extra mechanisms for efficient end-to-end communication. This requirement arises from the inherent lack of an adaptable routing mechanism in NDN that would promptly identify network changes and modify the NDN nodes accordingly. Targeting efficient end-to-end communication of the IoT application in these environments, we argue that a viable solution may involve the centralized management of the network. This approach would gather data in order to get NDN routing decisions dynamically in response to real-time network conditions.

Cloud-native applications' requirements

Regarding application execution, cloud computing offers economic advantages by providing scalability and flexibility for web application delivery by dynamically allocating resources based on real-time workload demands. Specifically, the employment of virtual machines facilitates the expansion of processing resources to accommodate a vast user base, on-demand, while also offering fault tolerance by maintaining adaptability to dynamic changes (e.g., sudden increase of network flows). However, large-scale web applications introduce complexity due to the heterogeneity of the network and compute requirements, often leading to inefficient management of the resources, especially when deployed in a cloud environment where different types of applications coexist.

A key feature of cloud-native applications is their design based on the microservices architectural paradigm which emerges as a significant strategy for addressing the previously discussed challenges. This approach segments complex monolithic applications or network services into a suite of simple, self-contained microservices that interact with each other. These microservices are commonly presented in containerized form, providing a lightweight, stand-alone, and executable software package that includes everything needed to run as an individual piece of software. This method brings service elasticity as an inherent

feature of applications, allowing for seamless modification of its entities without complex configuration adjustments. As such, resource utilization becomes more predictable since microservices typically have specific resource requirements. A microservice orchestrator (e.g., Kubernetes) does not have to scale the entire application but only the specific microservices experiencing excessive demand. This strategic approach allows large-scale applications or network services to accommodate huge amounts of client requests while maintaining fault tolerance (for example avoid server overloading).

The accelerated growth of automated data-centric procedures, alongside the demands posed by resource-intensive NIA like multimedia applications [9] (e.g., virtual reality) significantly increase the demands of cloud networks. Firstly, there is the challenge of supporting unprecedented levels of Quality of Service (QoS). This aspect is related to the maintenance of ultra-low latency and high throughput, which are critical for many modern applications. Secondly, the dynamic nature of these applications necessitates real-time adaptation of network protocols and resource management, as they have diverse compute and network requirements. Moreover, managing the workload efficiently for large-scale media applications often presents challenges, particularly when these need to be assigned to cloud-based services (e.g., run in multiple replicas of virtual machines or containers), leading to less-than-optimal resource management.

Along these lines, efficient traffic management and workload allocation among microservices play a key role in enhancing the overall performance of the NIA they compose. Here, we argue that implementing network policies, like load balancing, which accounts not only for the network conditions and cloud resources but also for the individual resource needs of each microservice, can lead to efficient resource management that can significantly improve latency and throughput performance. This approach arises from the fact that simple dynamic load balancing policies only focus on a single type of resource (e.g., bandwidth) may benefit bandwidth-intensive microservices but lead to the inefficient use of other resources.

SDN-based Centralized Control as a holistic solution

Given the challenges in NIA performance, which are directly influenced by efficient end-to-end (E2E) communication, we argue that a holistic approach of the aforementioned technologies is required in order to complement the advanced radio capabilities of 5G and WMN technologies. To achieve this, we propose improvements in the communication infrastructure that encompasses corresponding transformations at the higher levels of the network protocol stack (i.e., above the physical layer). Specifically, the proposed solution involves strategies that contribute towards a uniform orchestration of cloud, network (including novel approaches such as Non-IP protocol stacks), and microservices-based applications. We argue that integrating the above, can enhance scalability, dynamically accommodating the varying number of users, and allocate processing resources in real-time offering also adaptability of NIA in the quality conditions of wireless communication.

Software-Defined Networking (SDN) appears as an ideal solution to this shortcoming, providing the necessary elements of intelligent centralized control and network programmability in order to support NIA requirements. Especially:

- Regarding to the IoT applications over WMN, the utilization of SDN brings significant advantages by enabling a real-time global view of the network topology, providing centralized routing decision-making, and facilitating dynamic adaptation of NDN to the wireless conditions. In summary, the integration of SDN-NDN-WMN brings the following benefits: (i) the ability to reconfigure the NDN network in alignment with the dynamic changes e.g., when a new NDN node enters the network; (ii) increased network flexibility, which allows for the use of different paths according to the state of the network (e.g., based on network quality measurements); (iii) improved reliability and fault tolerance through rapid detection of network failures e.g., when an intermediate NDN node goes down.

- In relation to the cloud-native applications, SDN enables network performance enhancement by allowing the implementation of various network policies aligned with current workloads' network and compute requirements. For instance, dynamic SDN-based load balancing mechanisms

can reduce latency and improve throughput by adapting the network to workload changes (e.g., a sharp increase of network flows number). Furthermore, the SDN can improve application performance by making the appropriate routing decisions for incoming requests. That is, determining the most efficient cloud server to handle the request, based on a variety of monitored data, including network and resource consumption statistics, and the unique resource demands of each microservice e.g., some may be throughput other delay sensitive.

Nevertheless, there remains a need for SDN to improve its understanding and adaptability towards a better awareness of specific demands of individual applications (e.g., current computing resources' demands, such as CPU and memory consumption, for each application) and we argue that utilization of such mechanisms (like load-balancing) can be improved through efficient application profiling.

The need for predictive Microservices Profiling

In order to improve the traffic steering efficiency across cloud servers that host NIA, through adaptation to individual NIA demands, an important issue is the accurate and timely prediction of applications' impact on the network and compute resources. An important research challenge in this context involves the efficient management of cloud computing resources, where the problem is not last to address, since multiple applications coexist and share the same computational resources. Here, we argue that the "single-responsibility principle" of microservices, where each microservice fulfills a specific function of the overall application framework, could favor the resource usage prediction as microservices may have a more expectable impact on processing resources, compared to monolithic applications.

Consequently, by understanding these individual impacts, we enable application profiling mechanisms where the prediction models can achieve accurate resource usage prediction, thereby promoting efficient resource management. Such a mechanism, through its predictive capabilities, can effectively guide the orchestration and load-balancing systems. It would serve as a reliable path-finding tool, directing these systems to allocate resources adaptable, based

on the specific needs of each microservice. This not approach only improves resource utilization but also ensures the seamless execution of microservices with diverse demands, therefore enhancing overall application performance.

Key Research Aspects

Considering the background outlined above, this PhD dissertation is dedicated to advancing the traffic steering for NIA applications by exploiting the potential advantages that can be derived from cloud computing, Software-Defined Networking (SDN), Named-Data Networking (NDN) and the microservices paradigm. This research pays special attention to the adaptability facilitated by the centralized control feature of SDNs, considering it as a crucial element for effectively integrating the aforementioned technologies. This approach enables a highly adaptive network ecosystem conducive to NIA performance. In this context, our research focuses on three main key aspects:

- **Enhancing the efficiency and reliability of IoT applications through improved protocol control in wireless mesh networks.** Recognizing the dynamic nature of WMN conditions, this dissertation seeks to develop mechanisms that enhance the resilience and flexibility of applications running over wireless mesh networks, thereby ensuring consistent and reliable performance.

- **Enabling microservices adaptive load balancing**. Given the modular nature of microservices, it's essential to devise intelligent load-balancing strategies that ensure efficient resource allocation and usage, thereby enabling applications to handle an increased number of simultaneous requests without compromising performance.

- **Predicting the impact of various microservices on network and processing resources**. To further improve the efficiency of resource allocation, this research investigates the performance of prediction models that can accurately forecast the resource demands of individual microservices, in order to provide efficient utilization of resources.

Addressing the above issues, this research involves the implementation and experimentation of realistic SDN network environments and real indoor and

outdoor testbeds. The main goal is to foster a paradigm where online applications are more responsive and resource-efficient, through their adaptability to volatile wireless conditions (i.e., of WMN) and resource demands.

1.2 Research Objectives and Contributions

Figure 1.1 provides an overview of this thesis including the problem statement, research objectives, and contributions. More precisely, our main goal is to improve the performance of NIA applications by enabling their adaptability in two critical contexts. Firstly, we aim to improve the adaptability of IoT applications operating over the dynamic conditions of Wireless Mesh Networks (WMNs). Secondly, we seek to enhance the performance of cloud-native applications by addressing the specific resource requirements of each microservice that constitutes it.

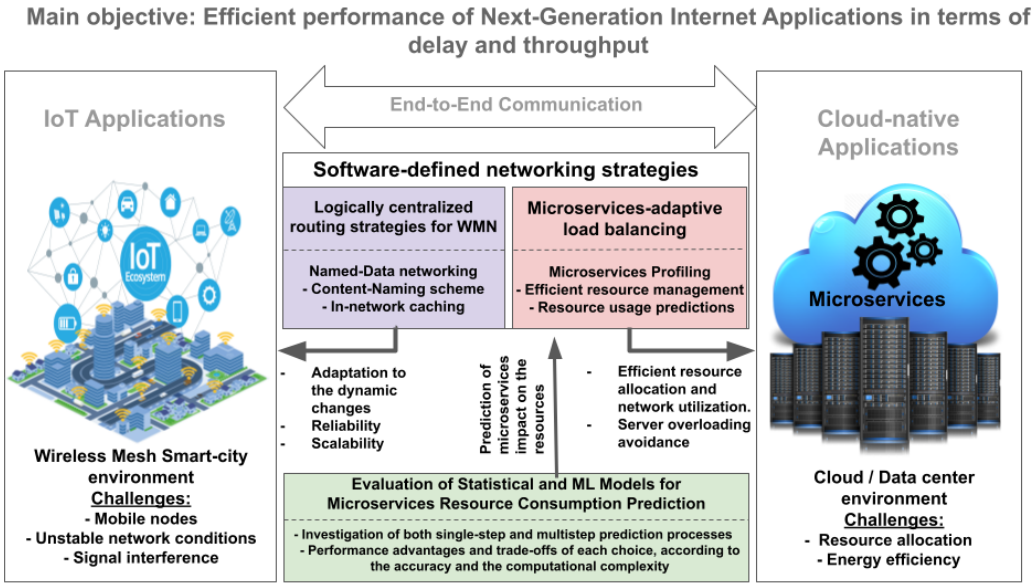

Figure 1.1: Research challenges and contributions

In order to achieve this goal, our work introduces three major contributions: Firstly, we have developed, tested, and compared two novel SDN strategies for Named Data Networking (NDN) based routing i.e., a dynamic approach and a static one based on clustering of WMN quality measurements. Secondly, we developed and experimented with a new SDN platform, designed for

microservice-adaptive load balancing, which takes into account the impact of microservice on network and compute resources. Lastly, we have conducted a study comparing statistical and machine learning models' capabilities to predict resource usage of individual microservices. A more detailed description of the above contributions follows.

- **Logically-Centralized SDN-Based Strategies for Wireless Mesh Smart-City Networks**. The main objective is to improve the performance of end-to-end communication between NDN producers-consumers over multi-hop WMN. Towards this goal, we consider and investigate NDN as a suitable protocol as: (i) provides flexibility to the network facilitating content retrieval from the network using data names instead of IP addresses, and (ii) reduces the communication overhead, offering reliability, thanks it's in-network caching capabilities. Considering the above, we developed and experimented with of a reactive and a proactive SDN-based solution that facilitates NDN adaptability in unstable wireless mesh networking conditions. In particular, our reactive approach exploits the advantages of SDN-NDN integration enabling efficient NDN Interest-Data exchange (in terms of network delay). Our solution encompasses: (i) a global view of network topology in real-time; (ii) centralized decision-making (including content-based decision-making and best-path selection) and (iii) dynamic NDN adaptation to network changes. The proactive NDN path selection strategy is based on evaluating available wireless links in terms of RSSI and delay. Specifically, in this approach, we collect historical network monitoring data and classify their performance to select the appropriate NDN path. The evaluation of our methods involves experimentation over real a WMN, providing and discussing the trade-offs between the two methods by investigating both control overhead and data message exchange performance, considering two different types of applications.

- **Microservices-Adaptive Load Balancing**. Our target is to extend microservice-based NIA performance efficiency in terms of response times, application-layer throughput, energy consumption and fairness while improving resource utilization of the cloud environment. In this context,

we propose, develop and evaluate a novel SDN-based load balancing facility that focuses on a specific use case scenario, that is, services consisting of microservices with heterogeneous, but simpler resource-demands compared to the service they constitute. Our solution adapts its load-balancing process to the particular requirements of microservices, based on dynamic microservice profiling. In summary, our approach offers: (i) microservice-awareness through online profiling that quantifies the level of importance of each resource type, based on simple prediction techniques. (ii) microservice-level adaptability of bespoke SDN-based load balancing policies, considering both cloud (i.e., CPU and memory) and network aspects (i.e., bandwidth allocation and flow sizes, in terms of duration time) and (iii) real experimentation of our approach with an assumed use-case that consists of microservices different resource requirements, demonstrating efficient and balanced server resource allocation, as well as improved application performance.

- **Performance Analysis of Machine Learning and Statistical Models in Predicting Microservices Resource Usage**. The main objective is to enhance the methods we use to estimate the impact of microservices on network and processing resources, leveraging the single-purpose functionality of microservices. More precisely, our experimental evaluation assesses the effectiveness of various statistical models in predicting resource usage for diverse types of microservices, with a particular emphasis on short time intervals. Our research lies in determining the model that demonstrates superior performance in resource forecasting across various traffic patterns: (i) generally, when considering the type of microservice; (ii) when based on the specific resource type; and (iii) when both the microservice and resource type are taken into account. In this context, our contributions include: (i) the evaluation of the performance of classical machine learning and statistical-based techniques over a diverse set of measured parameters and microservice requirements, spanning from typical time-series approaches, e.g., ARMA, ARIMA, and Kalman Filter to machine learning, e.g., LSTM and Random Forest; (ii) the investigation both single-step and multi-step prediction processes;

(iii) valuable insights and discussion of the performance advantages and trade-offs of each choice, according to the accuracy and the computational complexity; (iii) the potential advantages of rolling time-series procedures in order to adapt to dynamic changes. The evaluation is grounded in actual measurements of containerized microservices with varying resource requirements, such as those that are CPU or network intensive. Also, we consider distinct traffic patterns for this evaluation e.g., gradually decrease and increase, sharp increase, stable and random.

1.3 Chapters outline

The structure of this thesis is presented as follows. Each chapter depicts the specific mechanisms and strategies related to the three research challenges (mentioned in the overview subsection) and the primary contributions outlined earlier. In each chapter, we present a brief introduction, the motivation of our solution and contributions to the problem under consideration, followed by a description of the current state of related works in the literature and theoretical background, if necessary. Next, we describe the design of the proposed systems and the technical options of our solutions. Then, the experimental methodology and the associated setups are presented i.e., employed to validate the operations and performance gains of each approach. Finally, a series of experimental results are provided that, through their analysis, highlight the benefits and performance trade-offs of each investigated subject.

In **Chapter 2**, two technological solutions based on SDN are discussed, that offer performance gains and reliability (in terms of delay and packet loss respectively) in applications operating over wireless mesh networks. The first solution is dynamic i.e., is updated on the current state of the wireless network (e.g. topology rearrangements) and appropriately adjusts the paths of network nodes dynamically. The second solution is static, i.e. the paths are pre-selected based on classification of wireless links considering real measurements in terms of delay and signal strength (RSSI). The basic capabilities of the proposed strategies are detail described, that is, the functionality of the SDN-based system and its corresponding mechanisms. The experimental evaluation includes real experimentation over WMN and a special emphasis is placed

on the experimental setup and methodology, considering an analysis with our results based on real measurements.

In **Chapter 3**, we present our proposal for microservices-adaptive SDN-based load balancing. We analyze the functionality of the proposed platform, highlighting its unique capability for microservices-based adaptability which is grounded on a profiling activity building an improved view of each microservice, in terms of particular resource demands and resource status. More precisely, the Microservices Profiler component dynamically predicts the level of microservices' impact on each resource type and determines the typical flow size characterizing the latter. Then the proposed solution and its corresponding mechanisms are described in detail including: (i) the SDN controller (ii) the Monitoring Subsystem (iii) and the load balancing algorithm. Our experimental methodology involves load-balancing among microservices with heterogeneous, but simpler resource demands compared to the service they constitute and are based on a realistic use-case scenario. Our results revealed the significant performance advantages of our solutions, in both efficient cloud resource utilization as well as application performance in terms of response time, throughput, energy consumption and fairness.

Chapter 4 is structured around the primary goal of augmenting the methods we employ to assess the impact of microservices on network and processing resources. Our empirical study evaluates the effectiveness of various statistical models in forecasting resource usage for diverse types of microservices. Firstly, we analyze the problem statement of microservices resource usage prediction and provide corresponding approaches from the literature. We then move on to analyze our experimental methodology, detailing the specific microservices evaluated, the traffic patterns utilized, and the prediction models under consideration. Our research efforts focus on determining the most effective model for resource forecasting across varied traffic patterns, whether it's based on the type of microservice, the specific resource type, or a combination of both. We investigate the performance of both classical machine learning and statistical techniques over a wide range of parameters and microservice requirements, while also considering both single-step and multi-step prediction processes. Finally, we analyze the results of each model yielding valuable insights about the potential benefits and trade-offs of each choice, considering both accuracy

and computational complexity.

In conclusion, **Chapter 5** provides a summary of our research results, and potential future research directions stemming from our work.

1.4 Publications

The scientific findings of this thesis have been published in IEEE peer-reviewed journals and international conference proceedings. Below we give the complete list of our publications.

Refereed International Journals:

1. S. Kalafatidis, and L. Mamatas. "Microservices-Adaptive Software-Defined Load Balancing for 5G and Beyond Ecosystems." *IEEE Network* 36.6 (2022): 46-53.

2. S. Kalafatidis, S. Skaperas, V. Demiroglou, L. Mamatas, and V. Tsaoussidis. "Logically-Centralized SDN-Based NDN Strategies for Wireless Mesh Smart-City Networks". *Future Internet* (2022), 15(1), 19.

3. L. Mamatas, V. Demiroglou, S. Kalafatidis, S. Skaperas, and V. Tsaoussidis. "Protocol-Adaptive Strategies for Wireless Mesh Smart City Networks". *IEEE Network* (2022).

Refereed International Conference Proceedings:

1. S. Kalafatidis, V. Demiroglou, L Mamatas, and V. Tsaoussidis. "Experimenting with an SDN-Based NDN Deployment over Wireless Mesh Networks". *In IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 1-6). IEEE.

2. S. Kalafatidis, et al. "Experiments with SDN-based Adaptable Non-IP Protocol Stacks in Smart-City Environments". *2022 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2022. (Demo Paper)

3. I. Papakonstantinou, S. Kalafatidis, and L. Mamatas. "A Techno-Economic Assessment of Microservices." *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020.

Chapter 2

Logically-Centralized SDN-Based NDN Strategies for Wireless Mesh Smart-City Networks

2.1 Introduction

Smart-cities networks are characterized by a large number of heterogeneous end-user devices with spatial diversity, deployed over city-wide network regions. In this framework, the smart-city ecosystems adopt the Internet of Things (IoT) technology to utilize the smart-city applications (e.g., e-health [10] and environmental quality notification [5]), which are typically associated with critical performance requirements, e.g., low delay, reduced communication overhead, and high resilience. Wireless technology appeared as a key feature to meet the aforementioned requirements, increasing the communication range of distant infrastructure-free IoT deployments. As a consequence, various wireless communication technologies (such as WiMax, LTE, 5G, and beyond) have been proposed to support communication in smart city environments, each of them bringing different advantages to the network [5, 6].

Wireless Mesh Networking (WMN) seems to attract more attention in several case studies of smart cities (e.g., Stratford [5], CityLab [1], SmartShatader [7])

as is capable to support a wide range of wireless smart community areas in a multi-hop manner, with ease-deployable and low-cost infrastructure, while at the same time guarantees robust and efficient connectivity in dynamic environments (e.g., mobile users support). In summary, some of the advantages provided by WMN in smart cities are (i) flexibility supporting efficient interconnection of multiple distributed end-user devices; (ii) unstructured self-sustaining and self-configuring topologies, and (iii) integration of different networking technologies such as wired, optic-fiber, cellular and sensor networks [6]. Moreover, the multi-hop communication over the WMN can work as a complementary solution to other wireless networking technologies, e.g., to provide connectivity in areas where 5G coverage is not sufficient or achieve low delay in case the 5G RAN is far. However, the WMN backbone network may incorporate regional communication characteristics, such as areas with (i) static nodes, requiring efficient management of network resources [11]; (ii) mobile nodes, causing frequent instabilities in the network (e.g., topology rearrangements), and (iii) high signal interference [1].

On the other hand, Named-Data Networking (NDN) [8], an Information-Centric Networking (ICN) [12] architecture, has been proposed as a promising approach to match the IoT application requirements [13] in smart-city environments [14]. In particular, NDN architecture: (i) facilitates content retrieval from the network, as NDN packets contain data names instead of IP addresses and (ii) contributes to reduced communication overhead thanks to the in-network caching [15]. A typical example of NDN implementation on top of smart-city environments is the work [11] in which the authors examine the efficiency and effectiveness of NDN principles in CityLab testbed, using containerized NDN service placement in wireless nodes.

However, deploying NDN in such dense WMN requires employing additional mechanisms to enable efficient multi-hop NDN communication. On the one hand, an important aspect is the selection of nodes, in which, the NDN protocol will be deployed [16], to avoid unnecessary resource consumption. For example, Figure 2.1 depicts that in a multi-hop communication between two NDN nodes (producer-consumer), not all of the available intermediate nodes are required. On the other hand, a second major challenge is to adjust the NDN paths appropriately.

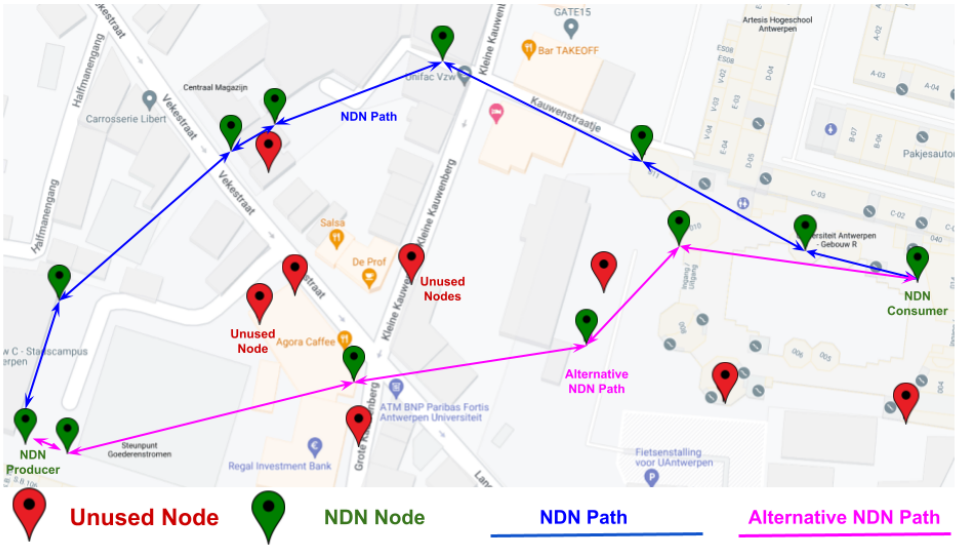

Figure 2.1: Example of NDN communication over CityLab testbed [1].

Considering the latter, smart-city networks provide a diverse set of communication conditions, spanning from stable to unstable conditions. For instance, a specific region of a smart-city deployment may suffer from unstable wireless conditions, such as frequent topology rearrangements, wireless node failures, or high signal interference in crowded areas [1], affecting the NDN performance. In this context, efficient NDN packet forwarding over WMN networks with unstable conditions and low-quality wireless links is remaining an open issue [16].

Here, we argue that Software-Defined Network (SDN) may provide the missing features of intelligent centralized control and programmability to facilitate efficient NDN operation in challenging communication environments, such as the smart-cities WMN [5, 17]. For example, in decentralized (e.g., non-SDN) NDN routing solutions, each NDN router is only aware of the state of its own paths and ignores other paths in the network. The SDN-NDN integration enables the centralized network view making easier the path selection/finding considering more sophisticated forwarding decisions based on several metrics (e.g., network state, RTT, hop counts, cash contents) [18]. Furthermore, the state of NDN routing nodes is maintained in low complexity as the routing decisions, and the network configuration functions are taken over by the SDN controller. Thus, the integration of SDN with NDN over WMNs may bring the following features: (i) global view of the network topology and centralized

monitoring data collection; (ii) allows the decision-making (including NDN best-path selection) based on the "global" network view; and (iii) the dynamic NDN configuration in groups of network nodes. In this context, we proposed an SDN controller which utilized a WMN routing protocol, providing a reactive solution for NDN path selection.

Despite the strengths provided from the SDN and NDN integration for dynamic routing, there is a limitation of the communication management overhead of the SDN. For example, the centralized of the SDN controller communication with the distributed network nodes brings communication overhead, and presents an important tradeoff between the cost and the benefits of designing SDN forwarding strategies [18]. Nevertheless, there are several studies that target reducing the SDN overhead, especially over low-power SDN wireless networks, e.g., [19]. Additionally, a reactive solution, such as [20], may handle dynamic changes in the communication performance, this comes with the cost of increased management overhead, due to the frequent (and potentially unnecessary) NDN path changes. The latter implies that this approach could be inefficient over network typologies consisting of static wireless nodes [1]. Motivated by this fact, we discuss the limitations of a one-fits-all NDN path selection strategy, claiming that an appropriate solution should take into consideration the respective network environment, even in the context of a smart-city environment.

2.2 Contributions and Chapter Organization

In this work, we discuss the trade-offs between two logically-centralized NDN strategies over WMN smart-city networks. More precisely, we consider the following SDN-based NDN path selection solutions, which they target efficient end-to-end NDN communication in terms of performance (i.e., delay): (i) a *reactive approach*, which is an extension of our previous work in [20] (this approach is utilized also in our work [21]) and re-adjusts the NDN path based on the BATMAN protocol; and (ii) a *proactive approach*, that a priori defines the NDN path, based on a combination of partitional clustering and similarity-based measures.

The key contributions of the specific work are summarized as follows:

- Implementation of an SDN-based system that supports two NDN path selection strategies (i.e., a reactive one and a cluster-based proactive), built on top of real WMN networks.

- Real experimentation over the w-iLab.t testbed [22], evaluating the performance impact of our SDN-based solutions considering experimental scenarios with stable and unstable conditions.

- Collection of real measurements related to RSSI, Delay, and Packet loss from indoor [22] WMN network. We record the performance of links among real wireless nodes and provide the measurements as released open-data [23].

2.2.1 Chapter Organization

The rest of the Chapter is organized as follows. The next section contrasts our proposal against the related works. In Section 2.4, we detail the characteristics and design choices of our approach, providing an extensive experimentation analysis in Section 2.5. Finally, our conclusions and directions for future work are presented in Section 2.6.

2.3 Background and Related Works

The smart community networks may benefit from the ICN approach as it can increase the network performance, reducing communication overhead, thanks to the in-network caching capabilities, as shown in [11]. However, ICN architectures, including NDN, lack of inherent mechanisms for supporting efficient NDN operation in challenging communication environments, such as WMN. In this context, content delivery and packet forwarding over such networks, with unstable wireless links, remains an open issue [16]. In this work, we study two different strategies for appropriate NDN path selection over WMN environments: (i) a dynamic approach with a reactive operation and (ii) a proactive one based on clustering evaluation of the quality of the wireless links.

Here, we discuss a number of NDN-based solutions sharing similar design characteristics with the aforementioned strategies. Moreover, we divide the representative-related works into dynamic/reactive and cluster-based approaches, comparing them with the basic technical characteristics of our strategies (i.e., SDN-based centralized control and real experimentation over WMN), as illustrated in Table 2.1.

Table 2.1: Related Work Comparison

Approach	Works	Centralized	SDN	Wireless Mesh	Real Experimentation
Reactive	SRSC [24]	✓	✓	X	X
	Multipath Forwarding [25]	✓	✓	X	✓
	Software-Defined NDN [26]	✓	✓	X	X
	SDN-NDN over WMN [27]	✓	✓	✓	✓
	Our reactive strategy	✓	✓	✓	✓
Cluster-based	Cluster-based NDN Routing [28]	X	X	X	X
	LCRN [29]	X	X	X	X
	NDN-based IoT [30]	X	X	X	X
	PiGeon [11]	✓	X	✓	✓
	Our proactive strategy	✓	✓	✓	✓

Recent approaches have been trying to resolve NDN routing and forwarding limitations with SDN [31]. In [24], authors designed and evaluated an SDN-based routing scheme for CCN/NDN (SRSC) which fully exploits the NDN principles, and, thus, the Controller and the nodes communicate using NDN messages (exchanging control and information messages). Particularly, the controller makes the routing decisions, and the NDN nodes act as forwarding devices only, as in our case. In SRSC, the Controller only informs an NDN node of the entire path to the content and afterward, the NDN nodes communicate with each other (hop-by-hop), to create the specific path. In contrast, we selected the controller to communicate independently with each on-path node and configure the NDN network, because of the unstable mesh topology.

In [25], the authors proposed an SDN-enabled Controller for multipath forwarding in NDN. The SDN controller analyzes the global view of the NDN network and makes appropriate forwarding decisions, according to the router states, the available forwarding paths, and the cached contents. The particular

centralized solution improves the performance of NDN compared to a distributed multipath forwarding strategy, which relies on *a priori* forwarding information and is inappropriate for networks with dynamic topologies, as in our case. Such solutions have been evaluated over a real-world WAN network, so frequent path changes and unstable topologies were not investigated in depth.

Authors in [26] introduced an integrated SDN-NDN framework and modified NDN's FIB design to address FIB overflow. In particular, FIB overflow is affected by (i) a large number of different contents and (ii) long-lived FIB entries. In our work, we maintain the default NDN's FIB design and address those issues by creating short-lived FIB entries in the NDN network.

In [27], authors target to improve the NDN performance through efficient content management, considering features of wireless communication. Their proposed solution has been evaluated in a real WMN environment and exhibits the most common design features with our dynamic approach (e.g., SDN-based centralized control over real WMN). Nevertheless, the main difference between our approach with [27] is decision-making regarding the dynamic conditions of WMN. In particular, our work aligns the functionality of NDN with the decisions of a dynamic WMN routing protocol, offering a rapid response of the NDN network to frequent topology changes (e.g., due to link failures).

Most of the aforementioned works are not validated in real-world test beds. Moreover, there is no prior work that addresses the challenging communication issues of deploying NDN in real-world mesh networks, using the SDN approach, as illustrated in Table 2.1.

As our proactive strategy, there are also other works that utilize clustering methods for NDN routing. In [28], authors propose an NDN routing protocol for wireless networks using clustering to reduce the number of nodes considered for route discovery. Moreover, in [29], a routing protocol is presented based on clustering to reduce routing overhead among the network nodes. In [30], the authors studied the integration of NDN-based IoT with Edge computing, introducing an algorithm that employs clustering to improve NDN routing. In contrast with our cluster-based approach, the above works are not evaluated over real network environments, and their routing decisions do not take into account WMN network characteristics (e.g., the quality of the wireless links).

In work, [11] authors use clustering for NDN service placement and in-

vestigate ICN advances by conducting experiments over a smart-city WMN test-bed [1]. The decision-making for the service placement takes into account compute resources (i.e., CPU) and bandwidth among the network nodes. In contrast to this work, we investigate the best NDN path selection considering RSSI and delay metrics among the network nodes, as we target efficient NDN communication over WMN in terms of delay and reliability.

From the literature review, we observe that most cluster-based routing works (as the above) mainly focus on excluding monitoring data or characteristics of the network (e.g., considered network nodes) that affect the routing decision, aiming to reduce the data communication latency or costs. Compared to these works, we approach cluster-based routing differently, using a multi-cluster approach as an evaluation quality measure to select the best NDN path (i.e., the clustering results are taken into account for the path selection). The employment of a clustering-based approach, which intuitively requires stationary data structures, corresponds with the results of an extensive experimental analysis over a real wireless network with static nodes (described in Section IV).

2.4 Proposed SDN-Based System

In this section, we present the proposed SDN system and the implemented reactive and proactive approaches, which target efficient NDN operation over WMNs. The main objective of both approaches is the selection of the appropriate paths between the NDN producer and consumer. In summary, we consider:

1. A reactive NDN path selection strategy that aligns NDN paths with the dynamic routing decisions of the WMN protocol. This approach is based on distributed decision-making information (i.e., each node chooses the best route for each destination among its neighbors) which is collected from the SDN Controller to configure the NDN nodes.

2. A proactive NDN path selection strategy based on evaluating available wireless links in terms of RSSI and delay. Specifically, in this approach, we collect historical network monitoring data and classify their performance to select the appropriate NDN path.

Their detailed description follows next.

2.4.1 Reactive NDN Path Selection Strategy

Here, we present our proposed SDN-based system and its corresponding mechanisms, aiming to the flexible and adaptive NDN operation in unstable WMN regions of smart-city backbone networks. Its main functionalities are (i) centralized monitoring of the wireless mesh network; (ii) dynamic best-path decision-making, and (iii) NDN configuration according to the selected routes. The system consists of two functional entities: (i) *the SDN Controller* which is the centralized control point of the network and (ii) *the Network Nodes*, which support NDN communication over WMN. A detailed description of the system components follows.

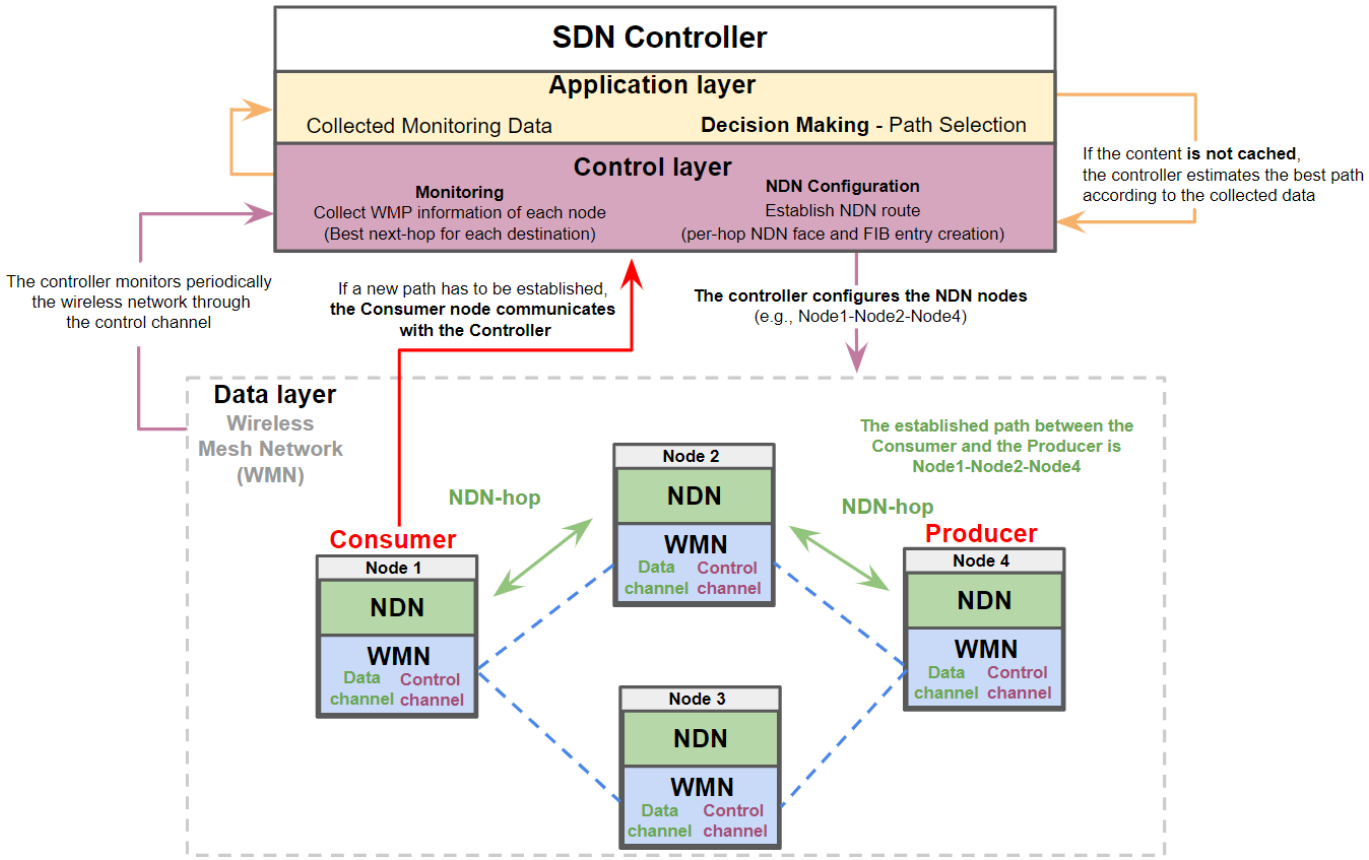

Figure 2.2: SDN-based Experimentation System.

SDN Controller

The Controller is the key component that enables the integration of the NDN with the dynamic decisions of wireless routing protocol [32] and as it performs centralized monitoring of the wireless nodes to configure the NDN paths. Its main functionalities are (i) information collection about the network state and rapid detection of network changes, enabling the global view of the entire network in real-time; (ii) definition of the best route for each content request between the NDN-Consumer and the NDN Producer, and (iii) to NDN route establishment including per-hop NDN face and FIB entry creation. The overview of the Controller's operation is presented in Figure 2.2.

The monitoring data collection of WMNs is performed through the WMNs routing protocol, i.e., the blue dashed lines of Figure 2.2. That is the centralized information collection from the distributed wireless nodes, regarding the discovery of the neighbors of each node and the routes that occur between the hops of the network. The routing costs among the links are based on the quality metrics of BATMAN [32] routing protocol (Layer 2), which is utilized in our implementation. The Controller is also located in the WMN network and communicates with the Network Nodes over IP.

The Controller manages the content requests through the reactive operation, following these four actions: (i) receives Consumer's updates for each new Interest packet; (ii) defines the best wireless routing path among the Consumer and the Producer, according to the collected monitoring data; (iii) establishes the selected NDN route, and (iv) triggers the Consumer to send the particular Interest packet.

Moreover, our system maintains information related to NDN in-network cashing. Especially, our Controller associates the content prefix with the corresponding utilized NDN paths and the content's Freshness Period to determine whether the content is cached. More precisely, as the Data packet will travel through the reverse path, the Controller node stores the path that has cached the particular Data packet and estimates the caching remaining time. This can be accomplished by maintaining information about the particular data packet (e.g., Freshness Period) as well as the corresponding caching information (e.g., utilized cache size, remaining entries, caching policy).

The Controller is located in the wireless mesh network and communicates with the wireless nodes over wireless links. Here, a major challenge is to guarantee reliable communication between the SDN controller and the WMN nodes, for example, in case of disruptive communication. Following our previous works [33, 34], we assume two wireless communication channels: (i) a control channel utilizing long-range but low-data-rate wireless communication, targeting the reliable connection between the Controller and the wireless nodes and (ii) a data channel utilizing a high-data-rate and short-range channel, supporting application data transfer. Although, the deployment of a reliable control channel is a complex aspect, and deserves an independent study outside the scope of this paper. We briefly discuss this approach here, to underline the complexity of the above task, which may become a major disadvantage considering the reactive strategies.

Network Nodes

The infrastructure of our proposed system consists of interconnected wireless nodes that support NDN communication. The NDN and WMN functionalities are independent and are integrated with the Controller (i.e., the Controller monitors the WMN to configure the NDN). Here, we describe the *NDN and WMN functionalities* to illustrate the system's network nodes operation.

In a nutshell, NDN is a future Internet architecture that follows the ICN principles and accomplishes named content retrieval by employing two types of packets (e.g., the Interest and the Data packets). In NDN, Consumers send an Interest packet in the network to fetch the corresponding Data packet that contains the requested content. Although a Data packet is originally generated from a Producer node, it may be retrieved from intermediate nodes' caches, as NDN supports in-network caching.

Each NDN node uses three components: the Content Store (CS), the Pending Interest Table (PIT), and the Forwarding Information Base (FIB). When a new Interest packet is received, the NDN node first checks if the requested content exists in the CS (i.e., it is cached), and in that case, it responds directly with the Data packet. Alternatively, if the prefix matches a specific PIT entry (which contains the already sent Interest prefixes associated

with the respective faces), then the incoming face is added to the particular entry (meaning that when the Data is fetched it will be forwarded also to that face). Otherwise, a new PIT entry is created and the Interest is forwarded to the next hop according to the FIB information [15].

In our NDN deployment, face creation and prefix registration are triggered from the Controller node. Since NDN communication is Consumer-driven, our system performance heavily relies on the Consumer node behavior. Thus, this plane targets to fetch the data efficiently with the minimum communication delay, by exploiting the NDN features.

The NDN consumer communicates with the SDN controller, if a new Consumer-Producer path has to be established, i.e., in cases: (i) the consumer requests a specific content for the first time or (ii) freshness period for a content (already requested) has expired. In all other cases, the requested content is retrieved to the consumer from the network, using the in-network cashing capability of NDN.

Here, we give an example of the NDN and SDN interaction of our system, as illustrated in Figure 2.2. The NDN Consumer (node1) intends to fetch the content with the prefix *e.g., sensor/temperature*. Thus, it sends a request to the Controller to inform them about the specific content. If the content is not cached in the network, the Controller finds the best path to the NDN Producer (e.g., Node1-Node2-Node4) and establishes the NDN routes. Finally, the Controller triggers the Consumer to transmit the Interest packet through the created path.

2.4.2 Proactive NDN Path Selection Strategy

The proactive strategy is also SDN-based and it is grounded on the implementation described in Figure 2.2. Specifically, each node in the network collects information from its neighbors related to RSSI and delays, periodically. The collected monitoring data is transferred to the SDN controller through the control channel. Then, the controller performs the clustering and chooses the best path by adjusting the NDN on the wireless nodes accordingly. Compared to the reactive strategy, in this implementation, the SDN controller forwards all the content requests on the selected path until it re-estimates the best path

again. For this reason, we consider this strategy proactive.

More precisely, the proactive strategy is based on partitional clustering and similarity-based distance measures. The main goal is to determine the best NDN path in terms of performance and reliability for end-to-end NDN communication (e.g., between an NDN consumer and NDN producer). In summary, our implementation has the following objectives: (i) the data collection among the network nodes, regarding the RSSI and delay, in a central node of the network; (ii) the clustering of the measurements, and (iii) the determination of the end-to-end NDN best paths among the wireless links.

In Figure 2.3, we illustrate the NDN path selection according to the clustering results, considering 4 clusters, sorted from best to worst based on the average value within the cluster. The color gradations indicate the cluster each link belongs to, i.e., the colors green, blue, black, and red symbolize the clusters from best to worst, respectively. Then, each provided path is characterized by its worst link, e.g., the worst path contains at least one link clustered as the worst link. For example, in Figure 2.3, the best path is that containing the nodes 1-2-3-5-8, since all the including links are clustered as best-links (notated with green color).

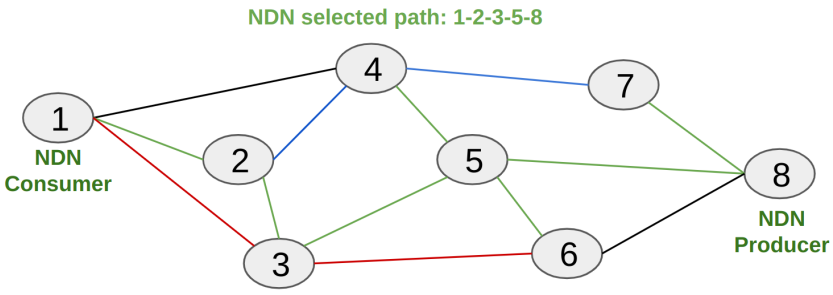

Figure 2.3: NDN path selection using the proactive strategy.

Here we describe the clustering method, we choose the dynamic time warping (DTW) [35] to estimate the similarity between the time series. Exploiting the nonlinear and time-independent nature of DTW, compared to typical measures (e.g., Euclidean distance), which provides a more robust and "global" (less sensitive to local time-series shifts) similarity estimation, improving the clustering performance. Moreover, we utilize the k-medoids [36] algorithm, since it is a representative object technique, more resilient to outliers [37] than

k-means. The fundamental steps of the proposed clustering approach, are given below:

- Step 1: Calculate DTW, pairwisely, between all the available links.
- Step 2: Apply the k-medoids algorithm to the vector that describes the pairwise similarity of all the available links (computed in Step 2). The number of clusters is pre-defined (in Section 2.5. We chose 4 clusters).
- Step 3: Sort the clusters, according to a specific Quality of Service (QoS) metric, for example, RSSI or delay.

2.5 Experimental Evaluation

In this section, we assess our proactive and reactive approaches for NDN path selection, targeting to the efficient operation of NDN over WMN. Our evaluation includes real experimentation over an indoor wireless environment, considering small-scale SDN-NDN deployments, utilizing the w-iLab.1 Fed4FIRE+ testbed. In summary, the evaluation metrics include (i) the NDN performance in terms of delay; (ii) the SDN communication management overhead; (iii) the frequency of dynamic path changes (path selection), and (iv) NDN reliability (i.e., percentage of Interest-Data exchange failures). More precisely, we consider two experimental scenarios:

- Scenario 1, which evaluates the reactive approach over different types of network conditions, i.e., stable and unstable WMN conditions.
- Scenario 2, which assesses the effectiveness of a proactive (clustering-based) approach in this context, also comparing its outputs with these of the reactive approach.

2.5.1 Experimentation Setup

Here, we present the most important experimentation details of both scenarios. The NDN implementation is based on the containerized Named data Forwarding Daemon (NFD) [38], developed in Docker containers [39] since it is lightweight, easily reconfigurable, and facilitates the NDN deployment on any

hardware. Moreover, we use the better approach to mobile ad-hoc networking (BATMAN) [32] wireless mesh routing protocol.

The multi-hop network topologies consist of wireless network nodes (Intel NUC devices) of the w-iLab.1 testbed. The nodes are equipped with an AR9462 wireless network adapter that we used to construct our wifi mesh network. We used the ath9k driver and configured these devices to run at 2.4 GHz with 20 MHz channel width in mesh mode. Moreover, we use a separate 802.11 wireless channel for the control messages (i.e., for the communication of the controller with the NDN nodes,) which is in mesh mode and uses the ath10 driver.

Regarding the performance measurements of the wireless links (i.e., RSSI and delay) utilized in the clustering process, we use wireless connectivity (among all nodes of the network) based on a Peer Link Management protocol, which is used to discover neighboring nodes and keep track of them. Here, the neighbor discovery is only limited to the signal range of each node. The evaluation is based on *Ping* tool by sending a batch of 100 ICMP packets every 3 s. As illustrated in Figure 2.4, the results present: *total* completion time—the total time to communicate of the 100 packets (we use a 2-s threshold waiting for each batch to complete. If the time is exceeded the particular batch is considered undelivered); *PLR*—the percentage of ICMP packets loss (i.e., out of 100); *avg*—the average Round Trip Time (RTT) of the ICMP packets; *sd*—and the Standard deviation RTT of the ICMP packets. The *RSSI* measurements are recorded from a second interface on each wireless node (the first interface was in mesh mode and was used for sending the ICMP packets), which is in monitor mode and uses the same driver. The performance measurements were collected from the w-iLab.1 testbed [22], utilizing the above method, are provided in [23].

Finally, the use case under consideration is an IoT scenario, where an IoT NDN Consumer requests sensor measurements from an IoT NDN Producer, that generates emulated sensor measurements. Here, focus on two IoT applications with different requirements: (i) short flows with delay demands, assuming a typical sensor measurement application, i.e., the produced data packets have a limited size (350 Bytes) and represent raw sensor measurements (e.g., temperature, humidity, etc); and (ii) long flows with throughput demands, assuming photo file transfers with a limited size (1 Mb). In the short flows, the emulation of Consumer and Producer applications are based on the *ndnpeek*

and ndnpoke tools, respectively, and for the long flows emulation, we use the *ndncatchunks and ndnputchunks* tools to transfer files as data segments. Moreover, to measure NDN connectivity failure attempts, we have disabled NDN retransmissions for the short-flow application.

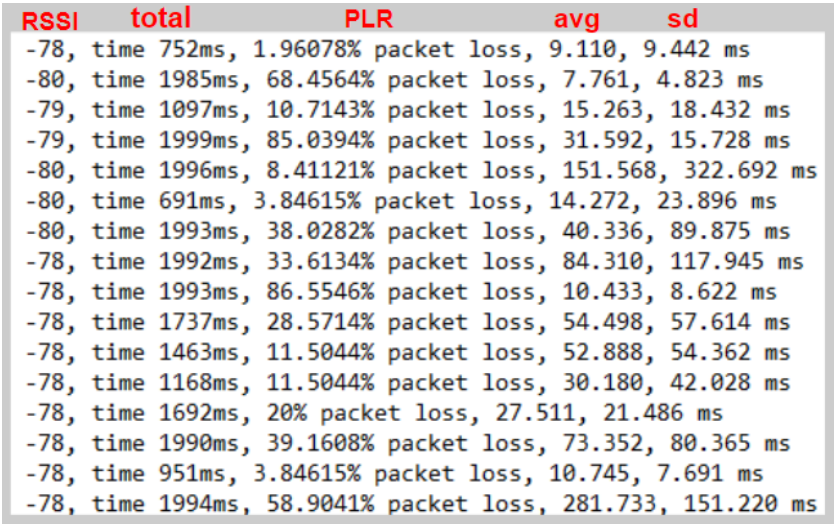

Figure 2.4: Evaluation of the wireless links.

2.5.2 Scenario 1—Evaluation of the Reactive Processes

Here, we demonstrate the functionality of the reactive strategy, targeting the flexibility and adaptability improvements that may bring, rendering NDN deployments over dynamic WMN topologies feasible.

In this scenario, we focus on the first IoT application scenario, i.e., short flows with delay demands. More precisely, we consider 150 different IoT contents generated from the Producer node, while the Consumer node transmits 1000 interest packets one by one to fetch the generated IoT contents from the Producer node. The content requests follow the Zipf distribution [40], (with $\alpha = 1.5$). We set the freshness period to 10 s, which is the time that the cached content is valid.

First, we discuss the results over stable networking conditions. In practice, we consider a multi-hop network topology consisting of seven wireless network nodes from the 9th floor of the w-iLab.1 Fed4FIRE+ [22] office lab, illustrated in Figure 2.5. The experiment assumes one Consumer (Node9-13), one Producer node (Node9-21), and the Controller (Node9-3) participating in the same

wireless topology, i.e., the latter performs centralized control of the entire network. Although the selected devices are located in close proximity, we adjust the transmission power (TP) of the wireless nodes to 3 dBm and managed to shape a multi-hop scenario. The monitoring data of each node include information about the BATMAN neighbors and originators enabling the global view of the network, and, are sent to the Controller through the A.L.F.R.E.D. tool [41]. The Controller configures the NDN nodes by transferring messages to the NFD container over TCP. These messages include information about the next hop and the particular content prefix and, thus, enable NDN face creation and prefix registration. Moreover, the nodes are not located nearby, thus, their connectivity is accomplished through the WMN. Results are collected over 10 repetitions of the experiment. Furthermore, the evaluation includes three metrics: (i) the performance of the NDN network in terms of communication delay (i.e., the Interest-Data exchange procedure between the Consumer and the Producer); (ii) the path establishment delay, i.e., the elapsed time at which the Controller makes the best path decision and configures the NDN network, and (iii) the performance of SDN decision making over the WMN. A detailed description of the metrics used, follows.

- RTT_{NDN} is obtained from the NDN Consumer node and represents the round trip time (RTT) between the interest packet transmission and the data packet reception,

$$RTT_{NDN} = RTT_{NDN_c} + RTT_{NDN_p},$$

  where RTT_{NDN_c} corresponds to the RTT values in cache-hit cases and RTT_{NDN_p} is the measured RTT in case of Producer-fetched content. Note that, the Consumer can fetch the requested content either from the Producer node or from the on-path nodes, due to the in-network caching.

- *Total Delay:* denotes the elapsed time between the Consumer's request to the Controller until the Data packet reception, including the path establishment delay.

- *Best Path Changes (BPC)*: represents the total number of best path changes in the whole duration of the experiment.

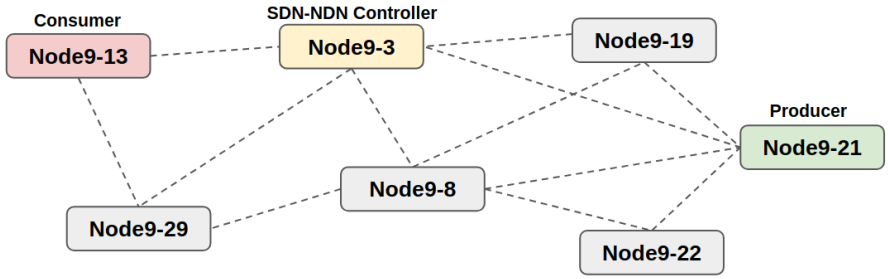

Figure 2.5: WMN topology, considering the 9th floor of the w-iLab.1 testbed.

We now move on to discussing our results. In Figure 2.6, we present the outputs of the reactive approach, considering the RTT_{NDN} and the *Total Delay* metrics. The low average RTT_{NDN} value, indicates that a reactive approach may ensure the seamless operation of the NDN in volatile network topologies, without compromising the NDN performance. On the other hand, the high average *Total Delay*, especially in contrast to the average RTT_{NDN}, reveals the additional network control overhead introduced by the Controller, as detailed in Section 2.4.1, which could be avoided in the case of proactive path selection strategies. Note that the significant difference between RTT_{NDN_p} and RTT_{NDN_c} (≈87 msec) is attributed to the locally occurred cache hits in the Consumer node (without requiring any network transmission) while fetching data from the remote Producer requires interest and data packet transmissions over the 3-hop topology.

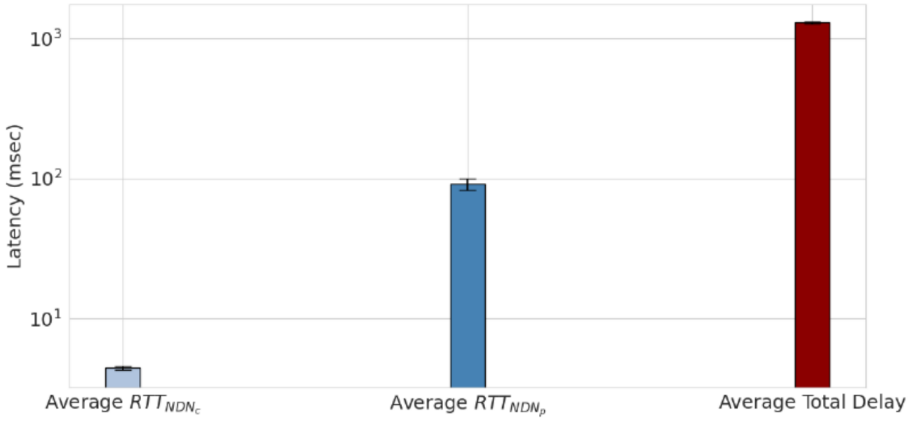

Figure 2.6: Average RTT_{NDN_c}, $RTT_{NDN_p}$and *Total Delay*.

Figure 2.7 illustrates the average best paths hops and the total *Best Path*

Changes (BPC), per round (experiment repetition). As is shown, the average number of hops (in each round) is 3, confirming the validity of the experimental methodology. Additionally, the *Best Path Changes* deviation (i.e., ranging from 7 to 19) illustrates the wireless links volatility of the test-bed's setup and highlights the proposed system's capability to capture frequent network changes and establish the appropriate paths.

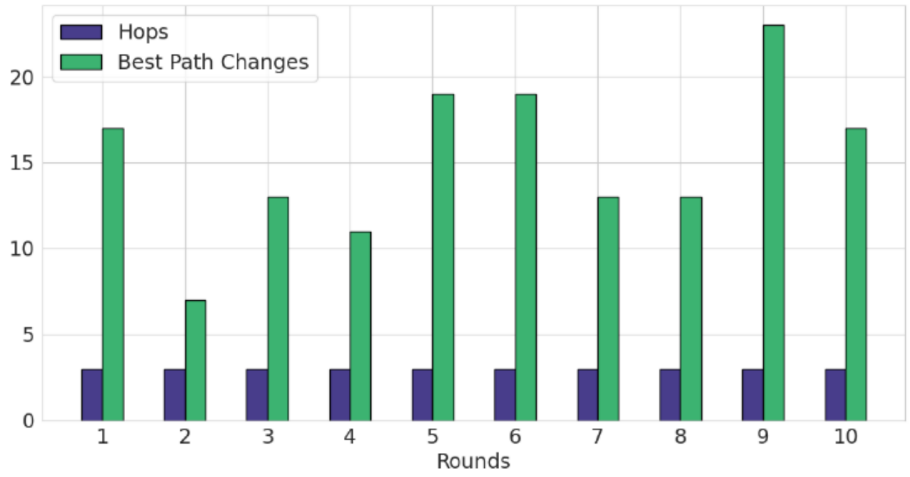

Figure 2.7: Number of Hops and *Best Path Changes (BPC)* pers round.

Subsequently, we discuss the outputs of the reactive process, targeting the effectiveness of NDN end-to-end communication, and, therefore, we exclude in-network caching (i.e., NDN cash contents). Here, the network topology consists of 6 network nodes from the 10th floor of the w-iLab.1 Fed4FIRE+ test-bed, as it is depicted in Figure 2.8. Node10-29 hosts the Consumer, node10-20 the Producer, and Node10-32 the Controller, which (as in the previous experiment) belongs to the same wireless topology. According to Figure 2.8, two multihop paths may be formed for the end-to-end communication, i.e., path1 (node10-29, node10-9 and node10-20) and path2 (node10-29, node10-32, node10-34, node10-5 and node10-20). The Transmission Power (TP) of node10-34 and node10-5 is 10 dBm, and the TP of node10-32, node10-9, and node10-29 is 5 dBm. Finally, we consider three types of communication conditions, by adjusting the Producer's TP, as follows: (i) 5 dBm (low TP), (ii) 10 dBm (high TP), and (iii) periodically increase/decrease from 5dBm to 10dBm, every 5 minutes (unstable TP).

Next, in Tables 2.2 and 2.3, we compare the proposed SDN, BATMAN-

based, reactive solution with the conventional NDN approach, i.e., fixed path1 and path2 solutions, over the total amount of 1500 Interest-Data exchanges from the NDN Consumer node.

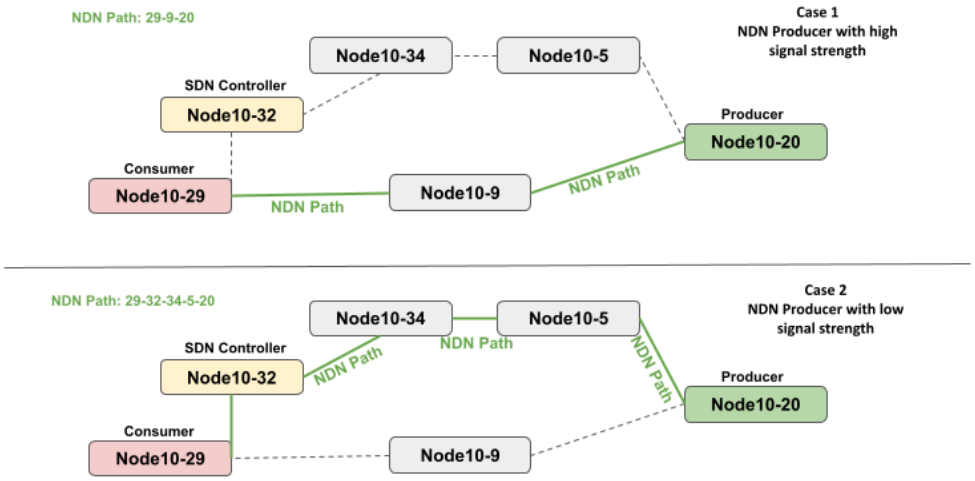

Figure 2.8: WMN topology, considering the 10th floor of the w-iLab.1 test-bed.

Table 2.2 enlists the average delay (msec) values of the Interest-Data exchanges for the fixed NDN paths and the reactive NDN solution. According to the outputs, Path2 provides the lower delay values for both low and unstable TP of the Producer, on the contrary, Path1 minimizes the delay for the case of high TP. Regarding the results of the proposed reactive strategy, the integration of the BATMAN protocol provides comparable performance in all cases, implying that a dynamic approach can successfully select the best paths, according to the network conditions.

Table 2.2: Average of Interest-Data exchanges performance delay (msec)

NDN Path	**Low TP (5 dBm)**	**High TP (10 dBm)**	**Unstable TP (5-10 dBm)**
Fixed Path1	61.62	29.86	53.71
Fixed Path2	45.36	38.62	43.35
Reactive solution	48.58	29.93	48.99

Similarly, in Table 2.3, we evaluate the fixed and dynamic NDN path selection solutions in terms of reliability, i.e., comparing the total amount of Interest-Data exchange failures. The reactive strategy demonstrates its

effectiveness in challenging communication conditions (low and unstable TP), providing the lowest number of interest-data failures, and also, indicating the potential gains of a dynamic mechanism, concerning unstable communication conditions. On the other hand, Path1 has the least failures in high TP cases.

Table 2.3: Total amount of failures over 1500 interest-data exchanges

NDN Path	**Low TP (5 dBm)**	**High TP (10 dBm)**	**Unstable TP (5-10 dBm)**
Fixed Path1	115	11	61
Fixed Path2	70	53	58
Reactive solution	58	40	51

Finally, Table 2.4 presents the results of the reactive solution's path choices regarding path 1 and path 2 (i.e., how many times BATMAN chose each path), considering the different TP adjustments for the Producer node. As it is shown, the reactive solution's path choices are in accordance with the best path selection, derived in Tables 2.2 and 2.3. In other words, our dynamic approach, integrating the functionality of NDN with the decisions of a wireless mesh protocol (i.e., BATMAN), successfully detects the best paths.

In a conclusion, scenario 1 demonstrates that an SDN-based solution may effectively support the NDN operation over WMN, especially considering the reliable performance on challenging communication networks. However, real-time network monitoring increases the network overhead, also adding extra delays to the network performance due to the centralized SDN control management overhead. Hence, in scenario 2 we discuss the potential gains of a proactive routing strategy instead of the operation of dynamic NDN path changes, targeting relatively stable wireless networks.

Table 2.4: Reactive's solution path choices over 1500 interest-data exchanges

NDN Path	**Low TP (5 dBm)**	**High TP (10 dBm)**	**Unstable TP (5-10 dBm)**
Path 1	41	1186	362
Path 2	1469	314	1138

2.5.3 Scenario 2—Evaluation of the Proactive Strategy

In scenario 2, we aim at the effectiveness of the SDN-based proactive procedure in terms of NDN performance over WMN smart-city environments. We use a multi-hop network topology consisting of ten wireless network nodes as illustrated in Figure 2.9 with green circles, including one Consumer (Node10-29) and one Producer node (Node10-20), located at the edges of the network.

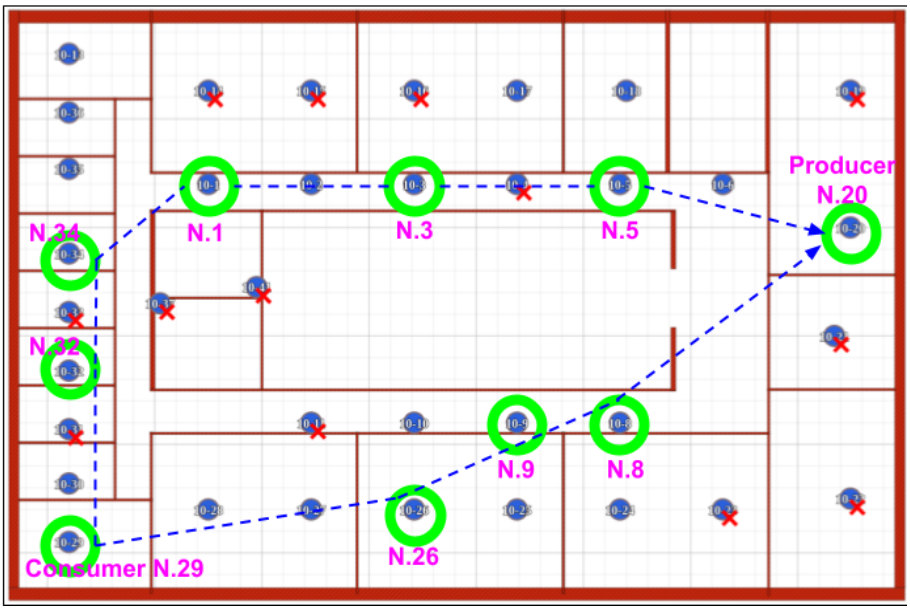

Figure 2.9: Selected topology—w-iLab.1 office lab 10th floor [2].

First, we discuss the clustering results over the experimental network topology. More precisely, the proposed clustering technique considers 4 clusters, classified from best to worst based on their intra-cluster mean value; the best cluster provides the minimum intra-cluster mean value. Figures 2.10–2.12 illustrate the clustering outputs considering as inputs the absolute RSSI, the delay, and the bivariate RSSI-delay values of the links, respectively. The color gradations denote the cluster that each link belongs to, i.e., green, blue, black, and red colors symbolize the clusters from the best to worst, respectively. According to the clustering results: (i) RSSI values mainly depict the distance among the nodes; as expected, since we experiment over an indoor environment with low interference, (ii) no evidence for the high similarity between RSSI and delay values exists, e.g., the path from node-1 to node-20, (iii) RSSI-delay-based clustering presents dissimilarities from both RSSI and Delay clustering.

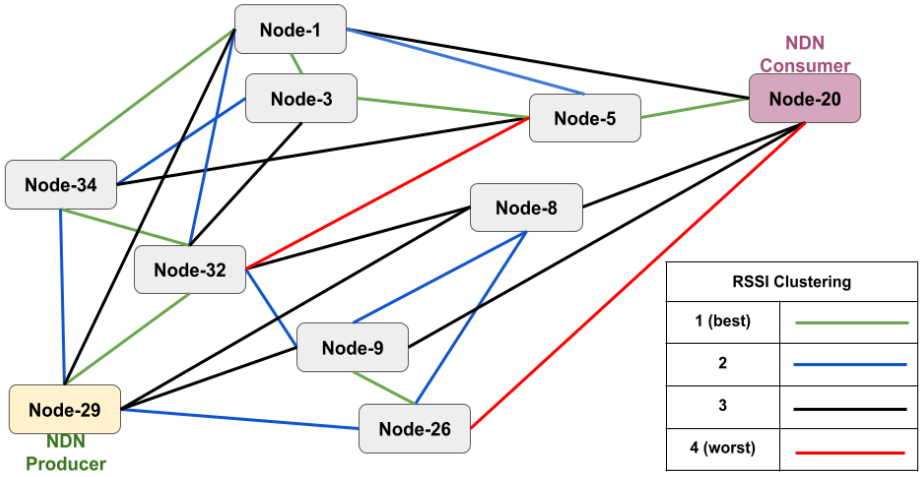

Figure 2.10: RSSI clustering results among network nodes.

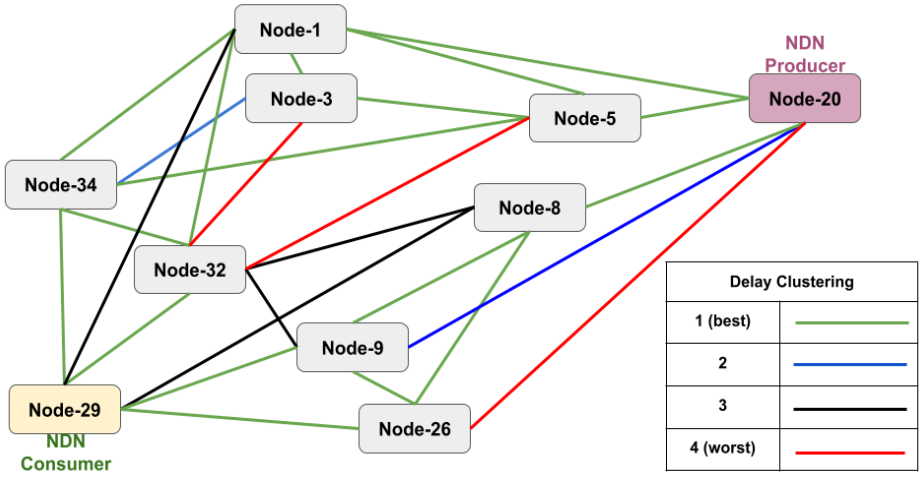

Figure 2.11: Delay clustering results among network nodes.

The second part of this scenario includes: (i) the performance evaluation of the available NDN paths over the considered WMN and (ii) the comparison of the clustering results with the choices of our dynamic SDN-based solutions. Specifically, in this phase, we performed 15,000 Interest-data exchanges (using the reactive process) over the topology described in Figure 2.5, and we recorded the BATMAN path choices (i.e., which paths have been chosen and how many times). Then, we performed 1500 Interest-data exchanges and 100 file transfers

for each selected path using the NDN chunk method, and finally, for each path, we present the clustering results.

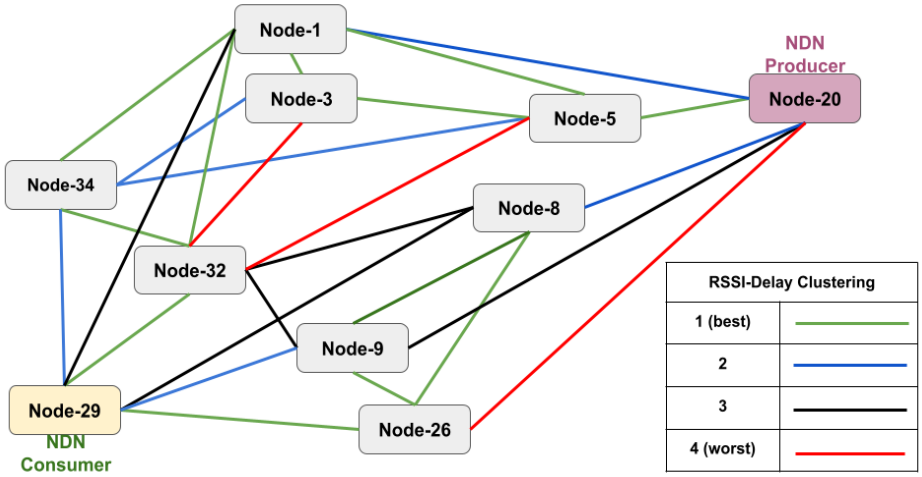

Figure 2.12: RSSI-Delay clustering results among network nodes.

Table 2.5 enlists: (i) the reactive's solution selected paths out of a total of 15,000 interest-data exchanges (Paths); (ii) the hops number of each path (Hops); (iii) the times each path was chosen during the experiment (reactive's solution path choices); (iv) the average delay for the Interest-Data exchanges (Interest-Data); (v) the percentage of Interest-Data exchange failures (Fails %), and (vi) the clustering results, for the metrics under consideration, i.e., RSSI, Delay, and RSSI-Delay. Here, notation C.i, $i = \{1, 2, 3, 4\}$ refers to paths sorted from best to worst. A path is characterized by its worst link, for example, C.1 exclusively includes the best links in terms of clustering and C.4 includes at least one link estimated as the worst link.

We discuss the results focusing on the Interest-Data exchange performance. In this context, Table 2.5 highlights that (i) the number of hops does not affect the Interest-Data performance and (ii) the reactive solution's path choices do not correspond to the best path selection of the clustering approach, for example, the path 29-8-20, No.14; (iii) reactive solution avoids unreliable paths, i.e., paths characterized by significant failures (Fails %). Regarding the clustering outputs, one may observe that the Delay based and the RSSI-Delay based clustering successfully reveal low delay paths, while, this is not the case in

the RSSI-based clustering, for example, the path with the minimum delay is characterized as C.3). Finally, all three clustering realizations provide similar results in terms of reliability.

Table 2.5: Comparison of the clustering results with the reactive process, considering the average delay and fails (%) of 1500 interest-data exchanges, over each of the available NDN paths

No.	**Paths**	**Hops**	**Reactive solution's path choices**	**Interest-Data Delay (msec)**	**Fails (%)**	**RSSI**	**Delay**	**RSSI-Delay**
1	29-32-1-20	3	42	14.82	3.2	**C. 3**	**C. 1**	**C. 2**
2	29-32-1-5-20	4	781	15.61	3.9	**C. 2**	**C. 1**	**C. 1**
3	29-34-1-20	3	52	16.79	5.7	**C. 3**	**C. 1**	**C. 2**
4	29-9-8-20	3	335	18.72	4.0	**C. 3**	**C. 1**	**C. 2**
5	29-32-34-1-5-20	5	8	19.22	2.3	**C. 2**	**C. 1**	**C. 1**
6	29-34-1-5-20	4	624	19.5	6.5	**C. 2**	**C. 1**	**C. 2**
7	29-26-8-20	3	359	21.67	5.1	**C. 3**	**C. 1**	**C. 2**
8	29-32-34-1-3-5-20	6	4	23.16	2.3	**C. 1**	**C. 1**	**C. 1**
9	29-32-1-3-5-20	5	40	26.79	3.9	**C. 2**	**C. 1**	**C. 2**
10	29-34-1-3-5-20	5	21	27.75	5.9	**C. 2**	**C. 1**	**C. 2**
11	29-1-5-20	3	1419	28.48	8.9	**C. 3**	**C. 3**	**C. 3**
12	29-1-20	2	208	33.46	10.9	**C. 3**	**C. 3**	**C. 3**
13	29-34-5-20	3	528	38.14	19.2	**C. 3**	**C. 1**	**C. 2**
14	29-8-20	2	5798	39.68	9.1	**C. 3**	**C. 3**	**C. 3**
15	29-32-34-5-20	4	5	43.27	23.5	**C. 3**	**C. 1**	**C. 2**
16	29-34-3-5-20	4	173	48.39	8.9	**C. 2**	**C. 2**	**C. 2**
17	29-32-34-3-5-20	5	20	54.68	8.9	**C. 2**	**C. 2**	**C. 2**
18	29-9-20	2	3758	54.92	6.2	**C. 3**	**C. 2**	**C. 3**
19	29-26-9-20	3	87	79.03	11.6	**C. 3**	**C. 3**	**C. 3**
20	29-32-8-20	3	398	79.95	22.4	**C. 3**	**C. 3**	**C. 3**
21	29-32-9-20	3	31	123.77	14.6	**C. 3**	**C. 3**	**C. 3**
22	29-26-20	2	28	144	21.1	**C. 4**	**C. 4**	**C. 4**
-	Reactive solution (total)	2 to 6	15000	89.19	4.7	-	-	-

Table 2.6 presents the number of hops over all the available paths in the corresponding network topology, and, the total amount and the percentage of hops included in the path selection of the reactive process, over 15,000 Interest-data exchanges. As described in Table 2.6, the reactive solution tends to select paths with the fewest hops, e.g., the paths with two hops are selected for 65% of the total requests.

Here, we evaluate the performance of NDN over the aforementioned static

Table 2.6: Number of hops included on each reactive's solution path choise, over 1500 interest-data exchanges

Number of hops	**Total amount of reactive's solution choices**	**Percentage of reactive's solution choices (%)**
2	9792	65,28
3	3251	21,67
4	1583	10,55
5	93	0,62
6	4	0,03

Table 2.7: Comparison of the clustering results, considering the average delay (secs) of 1MB file transfer using NDN-Chunks application

N.	Paths	**NDN-Chunks Delay (secs)**	**RSSI**	**Delay**	**RSSI-Delay**
1	29-34-1-20	11.48	**C. 3**	**C. 1**	**C. 2**
2	29-32-1-20	11.57	**C. 3**	**C. 1**	**C. 2**
3	29-26-8-20	11.72	**C. 3**	**C. 1**	**C. 2**
4	29-34-1-5-20	12.09	**C. 2**	**C. 1**	**C. 2**
5	29-9-8-20	12.26	**C. 3**	**C. 1**	**C. 2**
6	29-32-1-5-20	12.27	**C. 2**	**C. 1**	**C. 1**
7	29-34-5-20	12.7	**C. 3**	**C. 1**	**C. 2**
8	29-34-1-3-5-20	12.95	**C. 2**	**C. 1**	**C. 2**
9	29-32-34-1-5-20	12.98	**C. 2**	**C. 1**	**C. 1**
10	29-34-3-5-20	13.03	**C. 2**	**C. 2**	**C. 2**
11	29-32-1-3-5-20	13.18	**C. 2**	**C. 1**	**C. 2**
12	29-32-34-1-3-5-20	13.82	**C. 1**	**C. 1**	**C. 1**
13	29-32-34-3-5-20	15.24	**C. 2**	**C. 2**	**C. 2**
14	29-1-20	16.89	**C. 3**	**C. 3**	**C. 3**
15	29-1-5-20	16.9	**C. 3**	**C. 3**	**C. 3**
16	29-32-34-5-20	18.74	**C. 3**	**C. 1**	**C. 2**
17	29-8-20	20.18	**C. 3**	**C. 3**	**C. 3**
18	29-32-8-20	24.82	**C. 3**	**C. 3**	**C. 3**
19	29-9-20	25.37	**C. 3**	**C. 2**	**C. 3**
20	29-26-9-20	25.59	**C. 3**	**C. 3**	**C. 3**
21	29-32-9-20	27.11	**C. 3**	**C. 3**	**C. 3**
22	29-26-20	31.36	**C. 4**	**C. 4**	**C. 4**

NDN paths (i.e., Table 2.5) considering an NDN application that generates long flows. More precisely, Table 2.7 depicts the average delay (in secs) of transferring a file, sized 1 Mb, as NDN data segments utilizing the *ndncatchunks and ndnputchunks* tools. In this particular application, we enable NDN retrans-

missions and consequently, all requests are served successfully. The results in Table 2.7 reconfirm the conclusions of the previous use case. More specifically: (i) the RSSI-based clustering does not efficiently categorize the paths in terms of performance, since the lower delay values correspond equally to the C.2 and C.3 categories; (ii) delay-based clustering successfully identifies links with high delay performance. Interestingly, paths are categorized as C.1, such as the path 29-32-34-5-20 (No. 16), show inefficient delay, due to the influence of retransmissions on the total delay; (iii) The RSSI-Delay clustering provides comparable outputs to that of delay clustering.

Table 2.8: RSSI clusters average performance

RSSI	**No. of paths**	**Reactive solution's path choices (%)**	**Interest-Data Delay (msec)**	**Fails (%)**	**NDN-Chunks Delay (secs)**
C. 1	1	0.03	23.16	2.33	13.82
C. 2	7	11.11	30.28	5.76	13.11
C. 3	13	86.80	45.59	11.11	18.10
C. 4	1	0.19	144.00	21.13	31.36

Table 2.9: Delay clusters average performance

Delay	**No. of paths**	**Reactive solution's path choices (%)**	**Interest-Data Delay (msec)**	**Fails (%)**	**NDN-Chunks Delay (secs)**
C. 1	12	18.66	23.79	7.13	12.98
C. 2	3	26.34	52.66	8.02	17.88
C. 3	6	52.94	64.06	12.92	21.92
C. 4	1	0.19	144.00	21.1	31.36

Table 2.10: RSSI-Delay clusters average performance

RSSI-Delay	**No. of paths**	**Reactive solution's path choices (%)**	**Interest-Data Delay (msec)**	**Fails (%)**	**NDN-Chunks Delay (secs)**
C. 1	2	5.29	19.33	2.84	13.02
C. 2	11	14.66	30.05	8.62	13.18
C. 3	7	77.99	72.91	13.11	23.53
C. 4	1	0.19	144	21.13	31.36

Tables 2.8–2.10, summarize the clustering results, over Delay, RSSI, and

RSSI-Delay, by evaluating the average performance for both applications, i.e., NDN Interest-Data exchanges and 1MB file transfer using the NDN-Chunks application). More specifically, we measure for each cluster: (i) the number of paths; (ii) the percentage of reactive's solution path choices per cluster out of the 15,000 interest-data exchanges; (iii) the average delay of the Interest-Data exchange (msec) carried out on the paths belonging to each cluster; (iv) the percentage of failed Interest-Data exchanges, and (v) the average delay of the file transfer, considering the chunk application. Results indicate that the proposed clustering solutions efficiently categorize the paths, i.e., the average values of delays (for both applications) and fails increase according to the clustering characterization.

Table 2.11: Comparison of reactive and proactive NDN path selection strategies

Strategy	Advantages	Disadvantages
Reactive NDN path selection based on BATMAN protocol	Reliability and fault tolerance	Control management overhead
	Adaptation to unstable conditions (topology rearrangements)	Need to implement a reliable channel for the control plane
	Rapid detection of network changes (e.g., connection failures)	High complexity, support of small-scale topologies
Proactive cluster-based NDN path selection	Routing decisions based on Delay and RSSI quality measurements	Lack of adaptation to dynamic changes
	Support of larger topologies	Ignores the current network state
	Low complexity and control overhead	Does not detect topology changes

Based on the analysis of the scenario's 2 results, we can synthesize our conclusions in the following two points:

- The usage of dynamic protocols over relatively stable wireless mesh networks is not always the best solution in terms of delay, i.e., the reactive solution, which is based on BATMAN routing protocol, does not always select the "best" NDN path.

- Clustering is an efficient method for determining the "best" NDN paths over a stable WMN, considering both performance and reliability, for example, a clustering technique incorporating both RSSI and delay metrics may successfully locate the paths with low delay and interest data exchange failures.

Concluding this Section, in Table 2.11, we present the major advantages/disadvantages of both approaches, derived from the experimental analysis. More precisely, the overall results provided a piece of strong evidence that there is not a one-fits-all approach for the establishment of the appropriate NDN path over WMNs. More importantly, efficient rooting strategies should consider the communication conditions of the network environment under consideration.

2.6 Conclusions and Future Work

Smart-city environments incorporate a huge amount of nodes deployed in large areas, highlighting the need for efficient multi-hop NDN communication based on appropriate NDN path selection. To address the latter, in this paper, we have discussed two alternative SDN-based NDN strategies, involving a reactive and a proactive solution, extensively evaluated over a real WMN smart-city testbed. We investigated the potential gains of each approach in terms of end-to-end NDN delay performance, considering several use cases and wireless communication conditions. The real experiments demonstrated that i) a reactive approach is preferable in communication environments with frequent network changes since the real-time centralized management increases the network overhead; ii) a proactive solution (e.g., clustering based) may successfully identify wireless routes with high performance and reliability, considering smart-city deployments with stable network conditions.

In future work, we plan to

- develops a hybrid-protocol SDN platform involving mechanisms to distinguish between smart-city regions with stable and unstable network communication conditions, deploying accordingly the appropriate NDN path selection strategy.

- extend the controller's decision-making capabilities: (a) involving additional NDN-related parameters, for example, caching information of the intermediate network nodes and (b) elaborating improvements on both (reactive/proactive) NDN path selection strategies, based on AI/ML algorithms.

- validate the proposed NDN path selection approaches over large-scale WMN, with multiple consumers and producers.
- compare experimentally our SDN-based solutions with non-SDN strategies.

Chapter 3

Microservices-Adaptive Software-Defined Load Balancing

3.1 Introduction

5G and beyond networks (5GB) are enabling new applications and network services with challenging, stringent requirements, including ultra-low delays and high throughput, which are radically transforming vertical sectors, including manufacture, media & entertainment, automotive industry and energy. The main goals of 5GB include (i) exploiting higher frequency bands for improved throughput, e.g., millimeter-wave (mmWave) spectrum; (ii) improving spectrum usage efficiency through its intelligent management; and (iii) radically transforming the telecommunication system with the virtualization of physical network functions, reducing the capital expenditure (CAPEX) and improving the deployment and configuration flexibility.

Since challenging application performance concerns end-to-end (E2E) communication efficiency, 5G and beyond networks participate in flexible ecosystems integrating emerging technologies that improve the adaptability of the network or cloud environment to dynamic changes in application requirements or network conditions. Indicatively, for a 50ms E2E delay documented in 5G paper [42], radio access network (RAN) contributed to the 13%, while network and cloud aspects to the 87% of delay. As a bottom line, communication and capacity improvements should be matched by particular transformations at higher layers of network stack and computation aspects, i.e., jointly implementing

awareness and *adaptability* of radio, transport and cloud resources.

The cloudification of RAN processes e.g., Open RAN [43], makes the 5G operations flexible, bringing the ability of multiple independent instances of RAN functions to run on a common hardware platform. However, this technological approach requires efficient workload assignment and resource management improvements, as the virtualization of network services (i.e., breaking the software from the hardware allowing the RAN software to run on any hardware) is inherently energy inefficient. On the other hand, the resource requirements of the highly demanding 5GB applications (e.g., virtual reality) vary over time requiring dynamic adaptation of RAN resources to achieve acceptable levels of QoS or QoE.

Elasticity is a key feature of cloud computing provided to address the above needs. For example, virtual or physical resources may scale (up or down) according to monitored user demands, implementing horizontal and vertical elasticity policies. Traditional cloud environments use sizable virtual machines (VMs) and inflexible applications, while elasticity is mainly considering medium or long time-scales. For example, it may take minutes to boot up a VM and re-configure a non-flexible application, which leads to a non-responsive system for rapid changes in the network environment. Furthermore, VM migration or clustering requires the transmission or storage of data at the range of GBs, which is resource-consuming.

The microservices architectural paradigm appeared as a solution to above issues, breaking down complex monolithic applications or network services to a collection of simple, single-purpose communicating microservices, usually in the form of lightweight containers with rapid manipulation capabilities. A microservices orchestrator does not need to scale the whole application, but only those microservices that are overloaded. The employment of microservices targets large-scale resource-efficient applications with fault-tolerance requirements, since new microservices can be quickly deployed near the users, i.e., reducing latency, in a resource-efficient manner and without time-consuming configuration changes.

The centralized control of the Software-Defined Networks (SDN) offers flexibility in cloud computing improving the network adaptability and traffic control aiming to efficient service performance and resource allocation. Through

the global view of the cloud environment, the SDN enables the dynamic adjustment of network policies making rapid decisions, taking into account monitoring information of multiple entities of an ecosystem. Here, we argue that the SDN is an appropriate solution for microservices load balancing taking into account multiple characteristics such as microservice needs, cloud resource consumption, and network state.

However, in the most complex landscape of 5G networks and beyond, applications or network services may consist of tasks with heterogeneous resource requirements (e.g., some being CPU intensive, others network-sensitive, etc), increasing the complexity of resource management. We argue that these tasks could be decomposed based on specific resource requirements in stand-alone units of software packages (e.g., containers) thus constituting atomic microservices. Based on this approach, elasticity facilities driven by the prediction or rapid detection of load as well as the efficient workload assignment maintain resource-availability and simplify the traffic patterns reaching the microservices. In this work, we assume a specific class of applications that organizes its functions in a way that each one of them is characterized by simpler resource-allocation demands, compared to those of the whole service. We split these functions into containerized microservices call them Resource-Organized Microservices (ROM) focusing on the efficient workload assignment aspect among them.

In Fig. 3.1, we illustrate an example snapshot on the operation of alternative load balancing solutions with a relevant service. On its left side, we highlight the impact of a simple load balancing solution prioritizing servers with the least network load. This leads to an efficient balancing of network resources, but other resource types may be over-utilized, such as CPU and memory consumption in first and second servers, respectively. This strategy also leads to the deployment of additional microservices to handle the increased load, i.e., to a vertical elasticity event.

On the right side of Fig. 3.1, we depict a load balancing mechanism that is aware of the particular requirements of all microservices, the current resource availability of cloud resources, as well us of network properties, e.g., network utilization and flow characteristics. This approach can achieve a better balance of all resource types, improved application performance, while avoiding

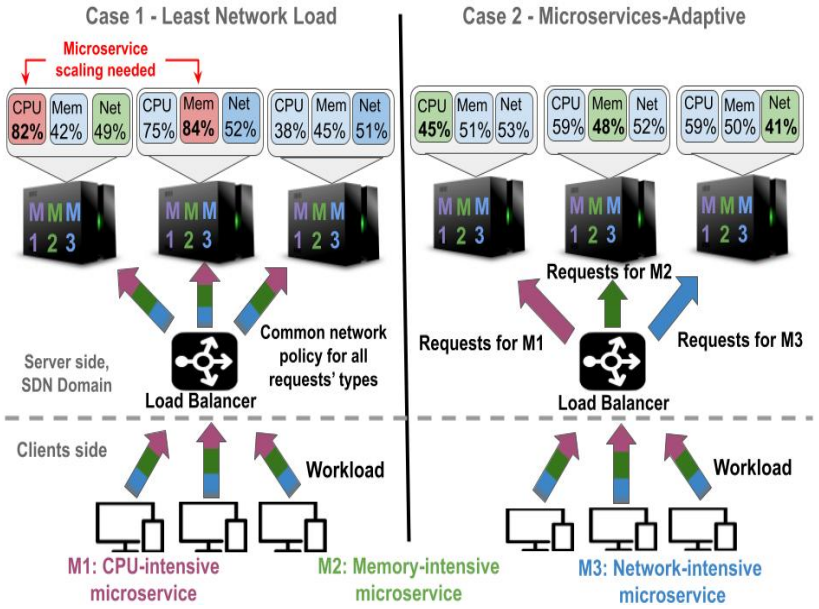

Figure 3.1: Simple vs microservices-adaptive load balancing

unnecessary scalability actions. Consequently, we argue that load-balancing should be microservices-aware, in terms of being *aware* and *adaptable* to the dynamic resource requirements of all microservices implementing the service.

In our understanding, typical load-balancing approaches do not fully address above requirements, as we show in Table 3.1, where we enlist indicative related works. For example, Kubernetes usually employs simple load balancing policies, including: (i) *round-robin*, balancing one-to-one requests among available servers in a circular fashion; (ii) *least network load*, assigning requests to the server with the lower network utilization; and (iii) *shortest expected delay*, assigning an incoming job to the server with the lower estimated expected delay.

As we see in Table 3.1, most relevant approaches balance load based on network and flow characteristics, while some of them take into account cloud resource availability [44], [45]. A few proposals consider individual microservices in the load balancing decisions, e.g., [46] proposes a chain-oriented load balancing algorithm to minimize microservice chains response time and [47] introduces a QoS-aware load balancing model considering links' capacity and delay between microservices. Due to the complexity of 5GB microservices-based applications, a sophisticated load-balancing mechanism should consider most above aspects, i.e., server resources as well as network properties, including

bandwidth availability and flow characteristics, in order to match the unique requirements of each individual microservice. In our understanding, our proposal (i.e., MALB) is the only solution in this direction.

Table 3.1: Well-known load balancing approaches

Load balancing (LB) mechanism	**Micro-services**	**Cloud**	**Net.**	**Flow char.**
Kubernetes round-robin	-	-	-	-
Kubernetes least network load	-	-	✓	-
Kubernetes shortest expected delay	-	-	✓	-
Mahout [48]	-	-	✓	✓
FDALB [49]	-	-	✓	✓
Hedera [50]	-	-	✓	✓
Server Cluster LB [44]	-	✓	✓	-
e-STAB [45]	-	✓	✓	✓
LB accross microservices [46]	✓	-	-	-
LB for Interdependent IoT Microservices [47]	✓	-	✓	✓
MALB	✓	✓	✓	✓

Here, we propose the Microservices-Adaptive Load Balancing (MALB) platform and corresponding mechanisms characterized by the following novelties:

- *microservice-awareness* through online profiling that quantifies the level of importance of each resource type, based on simple prediction policies.

- *microservice-level adaptability* of bespoke SDN-based load balancing policies, considering both cloud (i.e., CPU and memory) and network aspects (i.e., bandwidth allocation and flow sizes, in terms of duration time).

- *real experimentation* of our approach with a *challenging use-case* aligned to the class of ROS, demonstrating efficient and balanced server resource allocation as well as improved application performance in terms of response times, throughput and fairness.

In the following, a motivating use case scenario is presented, highlighting the advantages of the proposed platform.

3.2 Motivating use-case scenario

We focus on applications that consist of functions can be implemented as Resource-Organized Microservices (ROM). In this context, we assume a particular *virtual reality gaming use-case*, inspired by [9], and emulate its behavior in terms of network and compute resource utilization. It consists of service functions with diverse characteristics, in terms of resource requirements, function execution times, and flow sizes, including: (i) video streaming microservices being bandwidth-intensive and mainly producing long flows; (ii) user interactions' microservices, producing short flows and being characterized by low-latency and moderate CPU and bandwidth demands; and (iii) back-end microservices, including on user behavioral analysis, which are CPU-intensive but with a fixed execution time to maintain a given latency, i.e., return the best outcome within a deadline.

Typical dynamic load-balancing techniques may monitor a specific resource type, e.g., network utilization, and assign each new flow to the least overloaded entity (e.g., server, server cluster, edge cloud, etc.), in a reactive manner. This may work well for the video streaming microservices of the *virtual reality gaming use-case*, characterized by intense network communication, however, it may cause performance issues in other microservices that are sensitive to different types of resources (e.g., the back-end microservices) or accommodate flows with shorter sizes than the monitoring period (e.g., the user interactions' microservices).

Consequently, load balancing in the context of the considered *virtual reality gaming use-case*, should be able to: (i) detect the diverse resource-requirements of microservices using profiling techniques; (ii) accommodate alternative load balancing policies, matching the particular requirements of each microservice type; and (iii) consider both cloud and network-faced aspects. These requirements relate to the focus of this paper on transport and cloud aspects of 5G and beyond ecosystems, rather than on network and capacity improvements based on advances in radio technologies.

In the sections that follow, we present our relevant proposal and highlight experimentally its capabilities based on the considered use-case.

3.3 Proposed System

Here, we present our Microservices-Adaptive Load Balancing (MALB) platform and its corresponding mechanisms, targeting: (i) the minimum and balanced resource utilization of both networking and cloud aspects; and (ii) the efficient operation of applications constituting of multiple types of microservices with diverse resource-demands. As we show in Fig. 3.2, MALB is built on top of an SDN-based network and a cloud environment hosting containerized microservices consider as ROMs aiming to simplify the monitoring and management procedures. The three key components of our infrastructure support the following novel operations:

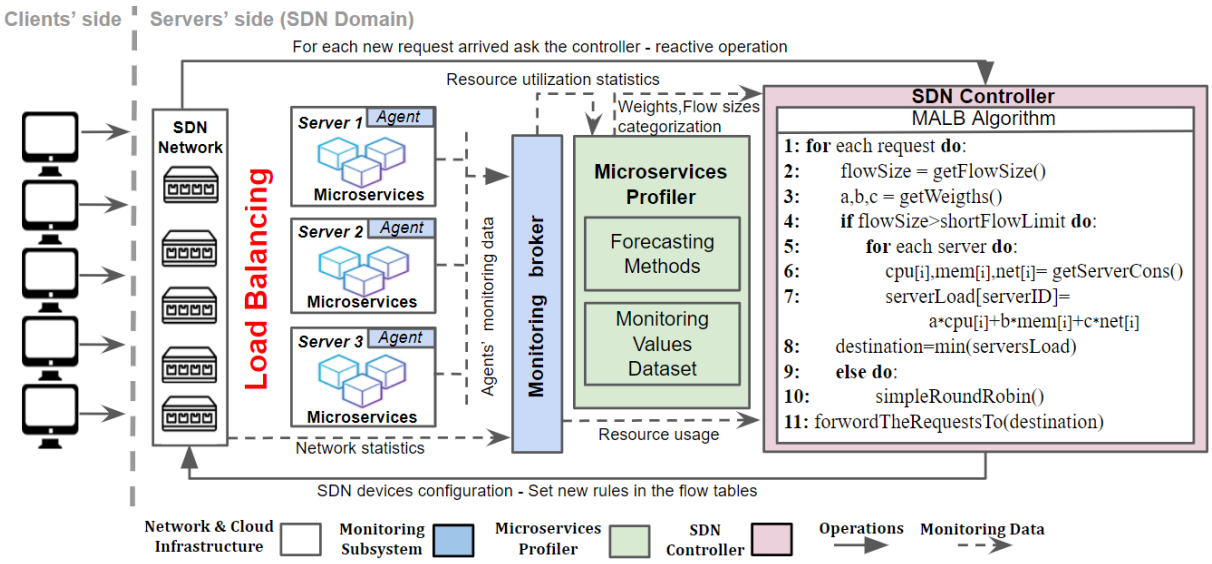

Figure 3.2: The MALB Architecture

- the *Monitoring Subsystem* monitors the SDN-based network, cloud servers and microservices, providing a holistic awareness of the application environment.
- the *Microservices Profiler* dynamically predicts the level of microservices' impact on each resource type and determines the typical flow size characterizing the latter

- the *MALB Algorithm* balances the load bespoke to each microservice type, exploiting the produced insights regarding the network and cloud environment hosting the microservice, as well as its unique resource-allocation characteristics.

A detailed description of the above components follows.

3.3.1 Monitoring Subsystem

The *Monitoring Subsystem* enables microservice-awareness through online monitoring of microservices and their network and cloud server environment. It consists of two main components: (i) the *Monitoring Broker* collecting centrally all monitoring information; and (ii) the *Agents* handling the extraction of monitoring data from particular cloud servers. For simplicity, we currently employ *Agents* in the servers only, since we emulate SDN devices through Open vSwitch, i.e., the latter reports directly to the *Monitoring Broker*.

Monitoring Broker collects link utilization and flow sizes from SDN network, as well as CPU and memory utilization for both cloud servers and each particular microservice. The network statistics are being provided to the *Monitoring Broker* through the Open vSwitch API. This approach offloads the *SDN Controller* from all monitoring information, because the data are being processed from the *Microservice Profiler*, i.e., the former receives summarized information, only. Alternatively, the *SDN Controller* could lookup link utilization and flow size data from the switches directly, based on the OpenFlow protocol. The cloud statistics are being provided from the *Agents*, which periodically lookup monitoring data through the docker stats application. *Monitoring Broker* communicates with the *Agents* through REST calls.

The monitoring aspect is critical and challenging on its own. Our next steps include the implementation of an *Agent* for real SDN switches interconnecting physical servers, utilizing a sophisticated monitoring infrastructure for large-scale service deployments, like [51], and carry out a study on the impact of monitoring period, tuning the trade-off between monitoring accuracy and involved control overhead, e.g., *SDN Controllers* are typically overloaded.

The monitoring data represent the input of *Microservice Profiler*, which description follows.

3.3.2 Microservices Profiler

The microservices-based adaptability of MALB is grounded on a profiling activity building the complete view of each microservice, in terms of particular resource demands and resource status of the surrounding network and cloud environment. This process is being handled by the *Microservices Profiler*, producing the following two-fold output for each microservice: its impact degree prediction on diverse resource types and a classification of its network flows into two categories, i.e., short and long. This output is being communicated to the *SDN Controller* via REST calls, while the historical values of monitoring data are being stored in a dataset.

The *Microservices Profiler* collects and processes the latest monitoring information communicated from the *Monitoring Subsystem* for each microservice and calculates a coefficient for each resource type (i.e., CPU, memory or network), reflecting the impact degree of the microservice on the particular resource to the total normalized resource consumption. Such values are placed on a window-based prediction mechanism that estimates the upcoming weights for each resource, after smoothing their evolution to remove outliers (e.g., CPU peaks). Although CPU peaks are important for load balancing [46], we decided to focus here on the average behavior of the system rather than on variance aspects, for simplicity.

Here, we assume that simple mechanisms exhibit a decent accuracy in predicting the resource demands of microservices, due to the single-purpose functionality of the latter. At this point of investigation, we employ window-based mechanisms based on the Rolling Mean (RM) and the AutoRegressive Moving Average (ARMA). RM is biased towards the recent measurements and it smooths the short-term fluctuations, while ARMA assumes linear dependence for the data. Although we cannot claim achieving maximum resource prediction accuracy, such shortcomings were not critical in terms of validating the key message in the paper, i.e., the importance of microservice-awareness and microservice-level adaptability. MALB could ideally employ a bespoke prediction approach to each resource type, matching the structure of the corresponding time-series. The incorporation and study of more sophisticated prediction mechanisms in MALB, considering also the variance aspect, are left

as a future work.

The flow sizes' classification determines which microservices produce short and which long flows, in terms of flow duration times, assuming that microservices in the context of ROM produce mostly either short or long flows. In practice, the *Profiler* groups the flow durations per microservice type and tags a microservice as a short flow one, when its flows last less than a specific threshold, and as long, when they last more. This strategy allows us to handle the case that a flow may be completed, before a load balancing schema is able to determine the status of microservices or servers, in terms of resource demands or availability, respectively.

Here, we argue that simpler policies with low complexity, including the Round Robin load balancing, suffice for such microservices producing short flows as due to their short duration there is no risk of their accumulation in a destination. We note that short flows benefit greatly from an efficient resource allocation of long flows sharing the same resources. In our case, MALB produces an accurate status of the system every 3 seconds, due to performance constraints of the open-source facilities we employ, i.e., Floodlight controller and docker stats tool. For example, a control plane latency that fluctuates by tens of milliseconds may create monitoring accuracy issues, in case of an 1-sec interval.

The *SDN Controller Algorithm* that follows realizes the microservices-based adaptability of MALB platform, based on the above two outputs.

3.3.3 MALB Algorithm

The microservice-level adaptability of our load balancing platform is realized through *MALB Algorithm*, an *SDN Controller* module that defines the appropriate destination (i.e., server, in our case) to forward each incoming request. In practice, MALB aims to balance the load among the servers and bring uniformity in the utilization of network and server resources, i.e., avoiding both network congestion and server overloading.

As shown in Fig. 3.2, MALB receives the outcome of *Microservices Profiler* expressing the predicted weighted resource demands of each microservice as well as a categorization of the typical flows' size of the latter. Furthermore, it

receives frequent snapshots of the resource utilization of all servers. At this point of investigation, MALB determines the type of microservice based on layer-4 ports.

MALB applies the Round Robin load balancing schema for the short flows (e.g, the user interactions' microservices), which is a simple and efficient strategy, in our experience. In the case of long flows (e.g., the streaming microservices), requests are forwarded to the server with the lower estimated load, i.e., combining the weighted resource demands of the particular microservice with the online resource availability of the servers. The load of each server is expressed as the weighted sum of its estimated CPU, memory and network utilization percentance. The respective weights are α, β, and γ, reflecting the importance of each resource type in the particular microservice.

SDN Controller calculates the load of each server and the particular microservice and forwards upcoming requests to the servers with the lowest load, i.e., through applying the corresponding flow table rules to the SDN devices.

3.4 Performance Evaluation

In this section, we provide our experimentation analysis on the i) evaluation of microservice profiling process of MALB, while backing relevant design choices; and ii) validation of MALB proposal, in terms of efficient service performance and cloud resource utilization.

Since our solution brings together SDN load balancing with containerized microservices, we conducted our experiments on Containernet [52], which supports both SDNs and Docker Containers. The studied load balancing mechanisms correspond to relevant Kubernetes policies, but are implemented in the Floodlight controller [53]. We simulated the workload, i.e, characterized by the number of requests from clients, through the Apache JMeter. We utilize a test-bed environment consisting of two physical servers and one switch. The first server hosts the client emulation facilities and the second the services.

The considered ROMs are aligned to the proposed use-case, which are: i) a video streaming service delivering videos of different sizes, built using *VLC* server; ii) a web service representing the users' interactions functionality based on *Flask* web framework; and iii) a back-end service (developed through PHP

using Apache server) which executes demanding calculations with configurable durations. Each type of microservices is characterized by particular resource requirements, i.e., the video streaming service is bandwidth-intensive, the back-end service is CPU-intensive and the user interaction service utilizes both processing and network resources, at a moderate level. The services are configurable to generate flows with different sizes. We do not vary the number of deployed microservices, since we focus on optimizing the system between elasticity events. We have assigned isolated physical resources to four different sets of microservices, i.e., representing four physical servers.

In our experiments, we consider an on-line game event with a 30-min duration, where the workload is characterized by three different 10-min phases, which descriptions follow: (i) *gradually increasing*, linearly increasing the load over a fixed time period, resembling gamers entering the event; (ii) *stable*, having a fixed workload, assuming a particular number of active players; and (iii) *gradually decreasing*, linearly reducing the workload over the same fixed period, i.e., users are leaving the system. All requests have been conducted asynchronously, i.e, using separate threads.

To realize the objectives of our experimentation analysis and highlight the novel aspects of MALB proposal, we devised the following scenarios: (i) the *microservices profiling*, evaluating the corresponding mechanisms, while motivating and supporting technically our approach to predict the requirements of microservices, in terms of CPU, memory, and network resource-demands; and (ii) the *MALB platform evaluation* scenario that validates the novel features of our proposal.

3.4.1 Scenario 1: Microservices Profiling

Here, we validate the dynamic microservice profiling process of MALB and investigate relevant design choices and configurations. Due to the dynamic nature of beyond 5G services, MALB estimates, for each forthcoming time-instance of the monitoring period, which is 3 sec in our case, the CPU, memory and network utilization of each microservice. This process defines the weight values of MALB algorithm, enabling microservice-aware load balancing. In practice, we evaluate the accuracy of RM and ARMA with different window

sizes, in terms of producing efficient weight values.

For understanding better the profiling process, we document the average calculated weights α, β, and γ for the full duration of an experiment with 1 video streaming request per 2 sec, which are 0.012, 0.452, and 0.536. This run produced average CPU, memory and network utilization values 0.51, 19.26, 22.87, respectively. Indicatively, γ value 0.536 reflects the importance of network (22.87) in the total normalized resource allocation 0.51+19.26+22.87. This means that this service requires insignificant network resources. The idea of our dynamic profiling feature is to carry out a similar process at each time-instance, while consider predicted resource allocations, instead of measured.

We conducted next experiments with separate deployments of: (i) the three considered microservice types; (ii) RM and ARMA having window sizes of 2, 5, 10 and 20, 50, 100, respectively; and (ii) the gradual increasing, stable and gradual decreasing user patterns, i.e., linearly increasing from 1 to 3 requests/sec, having 3 requests/sec, and linearly decreasing from 3 to 1 request/sec, respectively.

According to our results based on Mean Squared Error (MSE) measurements, both RM and ARMA exhibit a descent accuracy (i.e., MSE ranging most of the times between 0 and 8), while being especially accurate with under-utilized resources. This can be justified by the single-purpose functionality of microservices, i.e., they have a more expectable resource utilization in contrast to monolithic applications. Smaller window sizes seem to favor RM, while larger ones ARMA. However, it is more challenging to predict CPU and network utilization, characterized by a number of short-term peaks, especially for ARMA with its linear properties. For example, CPU peaks are more frequent with the stable number of clients, since such run utilizes more clients on average, i.e., a higher mean value produces a lower variance, causing a high MSE in the case of back-end service and CPU utilization metric. RM with window 2 either achieves the best accuracy or it is marginally outperformed. This is expected, since it follows the short-term dynamics of the resources.

For simplicity, we selected to use Rolling Mean with window size 10 in our experiments, balancing its accuracy with a smoothness level that follows the average behavior of the system. Such strategy appeared effective in our results, which could be further improved with a more accurate relevant prediction

mechanism, ideally considering both mean and variance aspects.

3.4.2 Scenario 2: MALB Platform Evaluation

In this scenario, we assess resource-allocation efficiency and impact on service performance of *MALB* in contrast to the *Simple Round Robin (SRR)* and *Least Network Load (LN)* strategies, both being widely used in SDN environments and microservice management platforms, such as Kubernetes.

We implement an SDN environment that stress tests above mechanisms with the communication of clients with four servers, each one of them hosting all considered microservices. Our goal is to evaluate the impact of *MALB* from both provider's (i.e., in terms of server and network resource utilization as well as energy efficiency) and clients' side (i.e., in terms of request response time, throughput and fairness). All provided figures and tables illustrate the average values between 10 runs. We did not have significant deviations among the runs, e.g., standard deviation ranged between 0.03 and 0.47, for the average measurements of CPU, memory and network resources.

Here, we emulated: (i) light (i.e., 1MB) and medium-sized (i.e., 10MB) file requests, for the *user-interaction* microservices; (ii) medium (i.e., 10 sec flow sizes, 15% of CPU) and heavy-sized (i.e., 30 sec flow size, 30-35% of CPU) communication, for the *back-end* microservices; and (iii) a live video broadcasting, i.e., without a specific flow duration, for the *video streaming*. The clients to all above microservices are being deployed, according to the assumed on-line game event.

We estimated the servers energy consumption based on work [54], by using the non-linear model for the CPU and the linear for memory and network. According to same work [54], the coefficients used are r = 1.4, CPU = 160W (4 processors), Memory = 36 W (4 memories), Other devices = 12W (disk) + 25W (pci slots) + 25W (motherboard) + 10W (fan) = 72 W.

Table 3.2: Load balancing level among the servers

	SRR				**LN**				**MALB**			
Metric	**CPU**	**Mem**	**Net**	**Watt**	**CPU**	**Mem**	**Net**	**Watt**	**CPU**	**Mem**	**Net**	**Watt**
STDEV	17.3	2.1	9.3	27.1	15.0	2.0	9.45	24.7	11.2	1.9	10.1	18.3
Range	44.2	5.2	23.6	70.1	38.4	5.1	24.3	62.8	29.2	5.1	25.1	47.7

Table 3.2 enlists the average of all standard deviations (STDEV) and ranges (i.e., max - min) for each timestamp among the four servers, quantifying their load balancing level. We observe that *MALB* demonstrates a significant higher load balancing level, compared to the other two approaches, of the CPU resources (e.g., 34%, and 24% lower Range than *LN* and *SRR*, respectively) and almost equally to the lower Range of *LN* for memory and network. Consequently, as the CPU is the most energy-consuming of the resources, the results show that MALB is significantly energy efficient (e.g., 32%, and 24% lower Range than *LN* and *SRR*, respectively). This result highlights the performance advantages of *MALB* in terms of load balancing, due to its wider view of resources and knowledge of the microservices' resource-demands.

In Fig. 3.3, we illustrate the mechanisms' performance in terms of average and worst-case resource consumption of CPU and network resources, over all servers at the full duration of the experiments. We quantify the worst-case resource consumption as the average of the 30 higher values of the particular metric, i.e., corresponding to the 3% of total measurements. The green bar represents the average of the worst cases among all servers and purple one the metric of the overloaded server. We also denote an elasticity threshold, when one of the metrics exceeds the 70%.

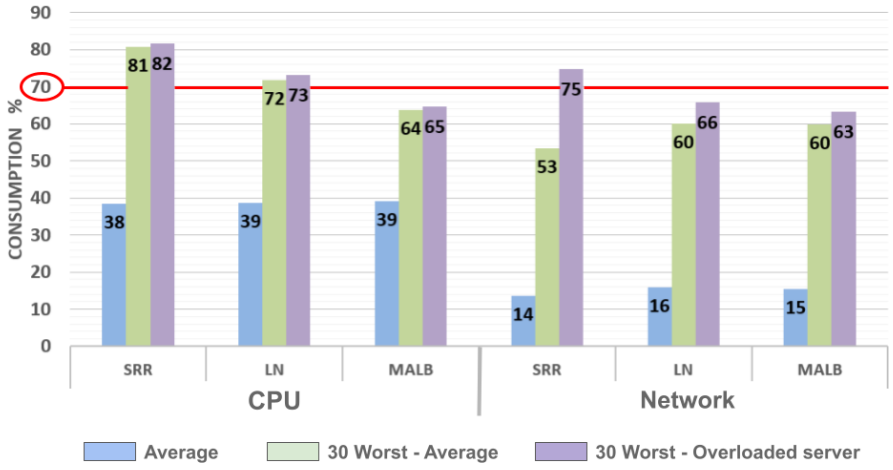

Figure 3.3: MALB Provider-Side Evaluation

Although all mechanisms exhibit a similar performance in terms of average consumption of CPU and network, i.e., attributed to the under-stressed cloud

resources, there are cases of high CPU utilization for both *SRR* and *LN*, triggering elasticity events that could be avoided with a better balancing of resources. For example, the worst CPU allocation of *MALB* is 16.9% lower than those of *SRR*. Also, in the case of MALB the overloaded server metrics are close to the average worst cases highlighting the fairness achieved in load balancing.

Table 3.3 illustrates our evaluation results from the clients' point of view. It enlists the type of microservices (*Microservice Type*), the total number of requests (*Requests*) for each particular run, as well as the considered metric (*Metric*) and the corresponding values. We document *Requests' Completion Time (RT, ms)* for all microservice types, besides video that does not have a a fixed execution time. The performance of the latter is measured in terms of application *Throughput (TP, kB/sec)*. We enlist the average and standard deviation of each corresponding metric over the indicated number of requests. Consequently, the latter metric is an indication of the fairness level among the microservices of the same type.

Table 3.3: MALB Client-Side Evaluation

Microservice Type	Requests	Metric	SRR AVG	LN AVG	MALB AVG	SRR STDEV	LN STDEV	MALB STDEV
User Int. Light	1600	RT	448	460	429	313	256	201
User Int. Medium	1500	RT	1175	1177	1068	552	493	336
Back-end Medium	900	RT	10226	10224	10224	198	125	89
Back-end Heavy	550	RT	30232	30230	30223	142	118	86
Video Stream	650	TP	1560	1818	1767	36	40	43

In general, Table 3.3 results verify the observations of Fig. 3.3 and Table 3.2, showing that efficient load balancing affects the performance of services. On the one hand, we observe from the average measurements that, *MALB* achieves the best *RT* performance of both short-flow and long-flow requests e.g., in the case of *user-interaction medium requests* workload, *MALB* completes requests 107 ms and 109 ms sooner than *SRR* and *LN*, respectively. This also underlines that microservice-level adaptability of *MALB* is an effective strategy. Regarding *video streaming* microservices, *LN* slightly outperforms *MALB* in terms of throughput (e.g., 51 kB/sec), since it is a bandwidth-intensive application. This outcome could be improved with a more accurate prediction mechanism for network consumption.

On the other hand, the STDEV measurements reveal significant advantages

of *MALB* in terms of microservice performance fluctuation and fair operation among the microservices of the same type, aspects being crucial for 5G and beyond services. *MALB* achieves up to 55% and 31% STDEV reduction contrasted to *SRR* and *LN*, respectively.

Our results can be summarized as follows. *MALB* improves resource utilization and application performance for the considered resource-organized microservices. We also confirm main *MALB* design directions: (i) to incorporate a simple load-balancing policy, such as *SRR*, for services generating short flows; and (ii) to handle long flows with dynamic load-balancing equipped with online resource monitoring of both network and compute resources, as well as adaptability to the particular resource-demands of each microservice type, driven by dynamic microservice profiling.

3.5 Conclusions

This chapter presented MALB, a novel SDN-based load-balancing facility that focuses on a special case of 5G and beyond services, i.e., services consisting of microservices with heterogeneous resource requirements. MALB employs load-balancing that adapts to the particular requirements of microservices, supported by dynamic microservice profiling. We provided an experimental analysis based on a virtual reality gaming use-case and a client requests' pattern resembling an on-line gaming event. Our results revealed the significant performance advantages of MALB, in terms of resource utilization of cloud environment as well as the response times, application-layer throughput and fairness of the microservices implementing the considered use-case, further supporting the significant radio performance and capacity advantages of 5G and beyond ecosystems.

Chapter 4

Evaluation of Prediction Models for Microservices' Resource Consumption

4.1 Introduction

Microservices have been widely adopted in cloud environments by both application and telecommunication providers. For instance, the 5G core is designed to be cloud-native, implying that its components are structured as sets of interconnected containerized network functions. These containers are orchestrated within cloud infrastructure by microservices orchestration platforms (e.g., Kubernetes) bringing several benefits such as: (i) enhanced scalability, allowing services to efficiently adjust to varying resource demand levels; (ii) fault tolerance, ensuring that a failure in one does not impact the others services; (iii) flexibility by enabling individual services to be updated or replaced without affecting the overall system; and (iv) efficient resource allocation due to the modular nature of microservices, enabling precise allocation of resources per service based on its unique requirements. This architecture is particularly beneficial in dynamic environments, such as 5G networks, where the demand and network conditions can change rapidly.

Efficient resource allocation poses challenges in microservice management, especially regarding scaling up (or down) in response to varying workloads.

Unlike the monolithic architectural paradigm, where resource allocation is determined based on the application's overall demands, the microservices approach presents a complex landscape of diverse resource needs and usage patterns for each distinct service. The critical task of microservice management necessitates efficient microservices profiling, which relies on the accurate prediction of resource consumption patterns, considering both immediate and long-term behaviors. Additionally, microservices are typically associated with heterogeneous resource demands (e.g., CPU and memory), where different profiling techniques need to be considered in order to adapt to a variety of demands, e.g., resource or data.

In our research presented in Chapter 3, we demonstrated performance benefits that could be derived from incorporating load balancing with microservice-awareness through on-line application profiling. This method quantifies the level of importance of each resource type, based on simple prediction techniques considering both cloud (i.e., CPU and memory) and network aspects (i.e., bandwidth allocation and flow sizes, in terms of duration time). Our findings highlighted significant performance advantages, in terms of resource utilization of cloud environment as well as better response times, application-layer throughput, energy consumption, and fairness.

In this chapter, we aim to investigate the effectiveness of various statistical and machine-leaernig models in predicting resource consumption for diverse types of microservices. Our work concentrates on short-term behavior, acknowledging the significance of accurate resource consumption predictions within limited number of time intervals. To achieve this, we leverage historical monitoring data obtained from CPU, memory, and network measurements. Our primary objective is to predict future resource values for containers' resource usage, considering the data generated during short time periods. Additionally, we incorporate rolling approaches, wherein the models are retrained for each new prediction, enabling us to recognize and adapt to the current resource state. Furthermore, we investigate the performance of both multi-step and single-step prediction strategies employed by the respective models under different traffic patterns. By examining various use cases, we can evaluate the models' adaptability and suitability for different microservices with distinct resource usage patterns.

4.2 Contributions and Chapter Organization

The main contribution of this Chapter are listed below:

1. We evaluate the performance of classical machine learning and statistical-based techniques over a diverse set of measured parameters and microservice requirements, spanning from typical time-series approaches, e.g., ARMA, ARIMA, and Kalman Filter to machine learning, e.g., LSTM and Random Forest.

2. We investigate both single-step and multi-step prediction processes

3. We discuss the performance advantages and trade-offs of each choice, according to the accuracy and the computational complexity.

4. We investigate the potential advantages of rolling time-series procedures in order to adapt to dynamic changes. Which is, in practice, infeasible for ML approaches.

5. Our methods are assessed on real measurements of containerized microservices with different resource requirements (e.g. CPU or Network intensive) considering five different types of traffic patterns: gradually decrease, gradually increase, sharp increase, stable and random.

4.2.1 Chapter Organization

The remaining chapter is organized as follows: In the next section, we contrast our approach with related work. Following that, in Section 4.4, we provide a detailed description of our experimental methodology, including the technical characteristics of the microservices, the considered traffic patterns, and the models employed. Section 4.5, we provide and discuss our research results, while our conclusions and directions for future work are presented as the final section.

4.3 Background and Related Works

Numerous related works in the literature have addressed resource consumption prediction, employing statistical and machine learning models. The majority of

these works focus primarily on workload prediction with the goal of anticipating resource consumption. For example, the works [55–57], have leveraged ARMA and ARIMA models to predict workloads, defined as the inflow of clients or requests to the network, under different traffic patterns. More precisely, work [56], uses the ARIMA model to predict future workloads evaluating the model's accuracy using real traces of requests to Web servers.

There has been a growing trend toward adopting machine learning models, specifically Long Short-Term Memory (LSTM), for forecasting purposes based on time-series data analysis. For instance, the study [58] applied the LSTM model in contrast to the ARIMA model to balance workload using Alibaba and Dinda workload traces datasets. Such a comparative approach provides valuable insights into the performance differentiation between a machine learning model (LSTM) and a statistical model (ARIMA) for an identical task.

A separate group of studies, represented by [59, 60], has targeted the prediction of power consumption. These studies draw on historical monitoring data from various resources, including CPU, memory, and network, and implement ML models for resource utilization prediction. They mainly focus on CPU utilization due to its significant contribution to overall energy consumption. Specifically, the study in [60] used a combined approach of Random Forest and LSTM to predict VM's CPU utilization by harnessing historical CPU utilization data.

Furthermore, certain works, like [61], aim to predict the power consumption of cloud servers to improve energy efficiensy in cloud data centers. The authors propose an Artificial Neural Network (ANN) method to model server power consumption, considering CPU-intensive and memory-intensive workloads. In addition, there are studies like [62] that apply statistical and machine learning prediction models based on historical resource usage data, but they focus on long-term predictions. As an example, the authors in [62] used LSTM and ARIMA models to predict CPU consumption in Google clusters, with a forecast scope spanning 2-hour, half-day, and 24-hour periods.

In contrast to these works, our study investigates the resource usage of containerized microservices, encompassing all three key aspects: CPU, memory, and network.

Other related works, such as [63], concentrate on container, application, or

microservice resource consumption, but their methodologies diverge. Instead of prediction models, these studies employ classification models and do not aim at online prediction. A notable example is the work [63], which adopts a machine learning-based classification approach to profile containerized applications. The main goal is to find the most suitable hardware infrastructure for deploying these applications, taking into account operational costs and the required Quality of Service (QoS).

Among the related works, [64] bears considerable similarities to our study. In this research, a hybrid model comprising ARIMA and triple exponential smoothing was designed and implemented for Docker container resource consumption prediction data. They also considered CPU and memory resource usage based on predicted values average in comparison to ARIMA, accommodating different types of workloads in their study.

In contrast to the aforementioned studies, our work is based on historical monitoring data from CPU, memory, and network to predict future values, focusing on the resource consumption of Docker containers considering online streaming data in short time periods (10 minutes experiments). We perform an in-depth comparative analysis of the performance of both statistical and ML models incorporating also rolling approaches (i.e., retraining the model for each new prediction) enabling us to recognize the existing state of resources. On top of that, our study includes the investigation of the performance of both multi-step and single-step prediction strategies of the respective models in use cases of different traffic patterns.

4.4 Experimental Methodology

Our experimental evaluation aims to investigate the effectiveness of various statistical models in predicting resource consumption for diverse types of microservices, focusing on small time periods. More precisely, our study seeks to answer the following research question: Which model performs better in forecasting resources: (i) generally, (ii) based on microservice type, (iii) by resource type, and (iv) by both microservice and resource type?

Internet applications or network services are inherently diverse, encompassing a wide range of functions that often exhibit varying resource demands.

These demands can include CPU-intensive tasks, as well as functions that are sensitive to network conditions. To capture the intricacies of such applications, our study involves the utilization of microservices with different resource requirements, taking into account the multifaceted nature of traffic patterns commonly encountered in cloud environments.

The evaluation metrics are related to the resource consumption percentage of CPU, memory, and network (bandwidth). The time series of the measurements have approximately the size of 300-400 values which corresponds to 10-minute experiments (cases with resource constraints, and low storage availability). A detailed description of our experimental methodology follows.

4.4.1 Considered Microservices

The microservices utilized in our study are implemented as Docker containers and possess the following characteristics:

- A video Streaming microservice, constructed using VLC server, which delivers 30-second videos. This microservice primarily demands bandwidth but has minimal memory and CPU requirements.
- A content delivery microservice that employs the Python Flask server to deliver Web pages of various sizes. While this microservice also relies on bandwidth, it exerts a more substantial influence on CPU and Memory consumption compared to Video Streaming.
- A back-end service developed with PHP using the Apache server, designed to perform on-demand calculations with adjustable durations. This microservice is CPU intensive and has negligible bandwidth impact, as its sole output after execution is an "OK" message.

4.4.2 Traffic patterns

We conducted a sequence of experiments, taking into account the previously mentioned microservices. All the requests were managed asynchronously via individual threads. The workload comprised of five distinct 10-minute traffic patterns, the specifics of which are outlined below:

- **Gradual Increase Scenario**: This setup involves a linear surge in workload over a predetermined time. Here, the test started with a rate of 1 request per second, which was systematically doubled every 2.5 minutes.

- **Gradual Decrease Scenario**: Contrary to the first scenario, this experiment involved a linear decline in workload over the same fixed duration. It commenced with a rate of 4 requests per second, which was halved at 2.5-minute intervals.

- **Stable Scenario**: This experiment was characterized by a constant workload. Throughout the test, the rate was maintained at 3 requests per second.

- **Random Scenario**: In this case, the request rate varied within certain time frames. The test included requests at different random intervals (i) ranging from 1 to 5 seconds, (ii) ranging from 1 to 10 seconds, and (iii) from 1 to 15 seconds.

- **Sharp Increase Scenario**: This scenario began with a rate of 1 request per minute. However, there was a drastic tripling of the request rate from the 3.5-minute to the seventh minute of the test.

4.4.3 Single-step Prediction Models

Here, we provide details about the machine learning and statistical-based techniques employed on the measured parameters outlined earlier. Initially, we apply conventional time-series methods, namely: (i) Auto-regressive Moving Average (ARMA); (ii) Auto-regressive Integrated Moving Average (ARIMA); (iii) Exponential Moving Average (EMA); and (iv) Kalman Filter (KF). Additionally, we incorporate advanced machine learning approaches such as Long Short-Term Memory (LSTM) and Random Forests (RF).

Given our focus on analyzing short time periods with limited data, considering the presence of resource constraints and limited storage capacity, we investigate the potential benefits of employing rolling time-series methods which effectively are adapted to dynamic fluctuations. This is particularly crucial as the available data may introduce cases where resource trends might not

have been previously observed by the models. Consequently, for every new prediction, the models undergo retraining utilizing the preceding values.

For all models employed, the data were divided into training (60% of the total data) and testing data (40% of the total data), with predictions commencing from the first value of the testing data. In terms of the rolling approach, we employed two strategies: (i) the models were trained to all previous values, including the testing portion, for each new forecast; and (ii) we implemented a fixed window size of 50 in order to expedite the training process while adapting the models to the current state. For example, when using a window size of 50, the model predicts the first value of the testing data utilizing the most recent 50 values from the training data. Similarly, for the second value prediction, the model retrained utilizing the last 49 values from the training data and the first value from the testing data.

A more detailed description of the models follows:

- **ARMA (2,1) & ARIMA (2,1,1)**: These are statistical models for time series data that describe the autocorrelation in the data. The ARMA model is characterized by two parameters: the order of the autoregressive part (2) and the order of the moving average part (1). The ARIMA model has an additional parameter that represents the degree of differencing (1) involved to make the time series stationary. In our implementation, both ARMA and ARIMA models are trained on the training data, and for each new value from the testing data, the model is updated to predict the new value one by one.

- **Rolling ARMA (window 50) and Rolling ARIMA (window 50)**: These models follow the same concept as the ARMA and ARIMA models, but instead of training on all previous data, these models only consider the last 50 values (referred to as the "window"). For example, to predict the second value in the testing set, the models are trained using the final 49 values from the training data and the first value from the testing data.

- **Rolling EMA (window 50)**: The Exponential Moving Average (EMA) model applies more weight to recent observations. The weight or "alpha" is calculated as 2 / (50 + 1) in this setup. The EMA model follows the same rolling window tactic as the rolling ARMA/ARIMA models.

- **Kalman Filter**: The Kalman Filter is a recursive algorithm used to estimate the current state of a dynamic system based on the previous observation and a model of how the system evolves over time. The algorithm has two main steps: the prediction step, where the future state of the system is estimated, and the update step, where the predicted state is updated based on new observations. In this setup, both the state and observation dimensions are set to 1.

- **Rolling Kalman (window 50)**: This model applies the same rolling window schema as the rolling ARMA/ARIMA models using the Kalman Filter algorithm.

- **LSTM (windows of 5, 10, 20)**: LSTM is a type of Recurrent Neural Network (RNN) that can learn and remember patterns over long sequences. The LSTM model in this setup is built using the Sequential class, allowing for multiple layers in the model. The first layer is an LSTM layer with 64 units. The output of the LSTM layer is passed to a dense layer with a single output unit, representing the predicted value. The model is compiled using the Adam optimizer and Mean Squared Error (MSE) loss function. The model is trained on the training data with 100 epochs and a batch size of 8.

- **Random Forest (windows of 5, 10)**: A Random Forest model is an ensemble learning method that operates by constructing multiple decision trees. In this setup, the Random Forest model consists of 100 decision trees (n_estimators=100). The random number generator used for initializing the trees is seeded with the value 42 (random_state=42).

- **Rolling LSTM and Random Forest**: These models apply the same rolling window schema as above. For each new value, the models are retrained using all the training data.

4.4.4 Multi-step Prediction Models

In this section, we describe the multi-step approaches. The 'steps' parameter is set to 5, which signifies that the models are designed to forecast the next 5 time

periods. Also in these experiments, the data are divided into training (60% of the total data) and testing (40% of the total data) sets. The models are then used to predict the first 5 values of the testing data based on the training data.

- **ARMA (2,1) and ARIMA (2,1,1)**: These models are trained using all the data from the training set. After training, they are used to forecast the first 5 values of the testing data.
- **Rolling ARMA and Rolling ARIMA (window 50)**: Unlike the traditional ARMA and ARIMA models, these models incorporate a rolling window approach. They are trained using the most recent 50 data points from the training set (the "window") and are then used to forecast the first 5 values of the testing data.
- **LSTM (window 5, 10, 20)**: The training process of the model is conducted using the available training data, where it learns patterns and relationships from historical information. In the case of the specific model being discussed, it employs a window approach for predictions. This means that the model utilizes the final 5 values of the training data as a window to make predictions for the first 5 values of the testing data. Similarly, for windows of size 10 and 20, the model considers the last 10 values of the training data to forecast the first 5 values of the testing data.
- **Random Forest with window sizes of 5, 10**: This method uses the same tactic with LSTM, however, due to the absence of a library that supports multi-step prediction for Random Forest models, an iterative prediction method with a rolling approach is employed. This involves making a single-step prediction for the next value, then including the predicted value as part of the input data for predicting the subsequent value. This process is repeated until all 5 values have been forecasted.

4.5 Experimental Results

In this section, we present and analyze the results obtained from our experiments. Our objective is to evaluate the performance of various forecasting models in

both single-step and multi-step prediction scenarios.

We begin with an evaluation of single-step prediction in Subsection 4.5.1. For each model, we assess the average computation time required for training and prediction, along with the accuracy of the predictions, which is quantified using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). This evaluation is conducted considering all scenarios, including different microservices and traffic patterns. We then delve deeper into the results, providing analysis per application and per resource type.

In Subsection 4.5.2, we extend our evaluation to multi-step prediction scenarios. Similar to the single-step prediction evaluation, we begin with an overall analysis considering all scenarios based on average computation time, RMSE, and MAE. This is followed by a more detailed investigation of results per application and per resource type.

Through this analysis, we aim to identify the strengths and weaknesses of each model in different scenarios and to provide insights that could guide the selection of suitable models for specific microservice and resource types.

4.5.1 Evaluation Results of Single-step prediction

Firstly, we quantify the computational complexity of each model, measured as the execution time required to predict all values in the testing data using the single-step approach. Notably, rolling approaches do not have a distinct training phase, as they are retrained at each step of the prediction process. Conversely, non-rolling LSTM (Long Short-Term Memory) and RF (Random Forest) models are trained once on the training data before making predictions on the testing data.

For these non-rolling models, we further distinguish between two specific measurements for better insight: (i) the total time taken by the LSTM and RF models to be trained based on the training data and subsequently predict the testing data; and (ii) the time taken by the LSTM and RF models to make predictions on the testing data, excluding the time spent on training.

According to Table 4.1, it is clear that the non-Rolling ML prediction models such as RF (Random Forest) and LSTM (Long Short-Term Memory) appear to have fastest prediction times. The prediction times for the entire testing

Table 4.1: Execution Time of the Forecasting Models (Single-Step Prediction)

Execution Time (sorted)	
Model	**Time (sec)**
RFw10	0.003
RFw5	0.003
REMA	0.010
LSTMw10	0.031
LSTMw5	0.031
LSTMw20	0.033
RFw5 (With Training)	0.074
RFw10 (With Training)	0.086
RKF	0.129
KF	0.131
ARMA	2.841
LSTMw5 (With Training)	3.234
ARIMA	3.398
LSTMw10 (With Training)	3.859
RARIMA	5.460
LSTM20 (With Training)	5.572
RARMA	7.656
Rolling RFw5	9.854
Rolling RFw10	11.927
Rolling LSTMw5	132.599
Rolling LSTMw10	163.654
Rolling LSTMw20	216.148

set range from 0.003 seconds for RF with a window size of 10 (RFw10) and 5 (RFw5), to 0.033 seconds for LSTM with a window size of 20 (LSTMw20).

Even when the training phase is included, RF models still demonstrate quick execution times, ranging from 0.074 seconds for RFw5 to 0.086 seconds for RFw10. This underscores the efficiency of RF models both in training and prediction phases.

On the other hand, traditional statistical models such as the Kalman Filter (KF), ARMA, and ARIMA exhibit more substantial delays. This is largely because they need to be retrained at each new prediction step, which significantly increases their computational complexity and hence the execution time.

The slowest models in terms of prediction time are the rolling variants of ML models, specifically RF and LSTM. Due to the retraining required at

every prediction step, these models exhibit much longer delays. For instance, the execution time for Rolling LSTMw20 model reaches a high of 216.148 seconds. This underscores the fact that the training phase for ML models is more time-consuming compared to traditional statistical models.

In conclusion, while ML models like RF and LSTM offer rapid prediction times, their rolling counterparts require significantly more time due to the constant retraining involved. As such, the choice of model should take into account the trade-off between prediction speed and the ability to adapt to new data.

Table 4.2: Comparative Performance of the Forecasting Models Based on RMSE and MAE Metrics

RMSE			**MAE**		
Model	**RMSE Average**	**RMSE ST.DEV**	**Model**	**RMSE Average**	**RMSE ST.DEV**
ARIMA	3.31	5.80	**ARIMA**	1.67	4.59
RARIMA	3.44	6.00	**RARIMA**	1.75	4.74
ARMA	3.57	5.93	**Rolling LSTMw5**	1.97	(winner) 2.95
Rolling LSTMw5	3.88	(winner) 5.36	**Rolling RFw5**	2.18	3.78
Rolling RFw5	3.90	6.12	**ARMA**	2.23	4.94
Rolling RFw10	4.03	6.67	**Rolling LSTMw10**	2.28	3.64
RARMA	4.24	7.40	**Rolling RFw10**	2.44	4.37
Rolling LSTMw10	5.52	9.58	**RARMA**	2.60	5.81
LSTMw5	5.88	9.44	**RKF**	3.85	11.35
RKF	5.90	12.89	**LSTMw5**	3.90	7.72
KM	5.97	12.98	**KM**	3.93	11.56
RFw5	6.65	9.71	**LSTMw10**	3.95	6.83
RFw10	6.79	10.24	**RFw5**	4.47	7.61
LSTM20	7.43	13.43	**RFw10**	4.86	8.56
REMA	7.65	11.51	**REMA**	5.88	10.35
Rolling LSTM20	8.76	16.86	**Rolling LSTM20**	6.39	17.41
LSTMw10	12.38	30.11	**LSTM20**	24.13	80.19

Table 4.2 presents a comparative performance analysis of the considered

models for the three distinct microservices: Video Streaming, PHP Back-end, and Web Content Delivery. The table provides insights into the models' Root Mean Squared Error (RMSE) average and RMSE standard deviation for each microservice. The term 'Winner' is used to highlight the best-performing model in each category. According to the result, we make the following key observations:

The ARIMA model outperforms other models in terms of both the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) when considering average values. With an average RMSE of 3.31 and MAE of 1.67, it indicates that the ARIMA model has demonstrated the most accurate predictions on average across all datasets. This suggests that the ARIMA model is capable of capturing the underlying patterns in the time series data effectively, leading to lower prediction errors.

On the other hand, the Rolling LSTM model with a window size of 5 (Rolling LSTMw5) demonstrates the most robust performance in terms of standard deviation in both RMSE and MAE. This implies that the Rolling LSTMw5 model provides the most consistent prediction results across different datasets. While its average error metrics may not be the lowest, its performance is less likely to vary greatly when applied to different datasets, making it a reliable choice for forecasting.

It is also interesting to note that the non-rolling versions of machine learning models tend to have larger standard deviations compared to their rolling counterparts, indicating that their performance might be more sensitive to the specific characteristics of the dataset.

In summary, the choice of model could depend on the specific requirements of the forecasting task. If the main objective is to minimize prediction errors on average, the ARIMA model would be the preferred choice. However, if the aim is to achieve reliable and consistent performance across various datasets, the Rolling LSTM20 model might be more suitable.

Results per application considering all types of resources

Here, we provide an evaluation of the single-step prediction results obtained from different models across different microservice types. The focus of this

analysis is to assess the performance of statistical and machine learning models while considering all types of resources involved.

The evaluation results, as depicted in Table 4.3, showcase the performance of each model for the Video, PHP, and Web microservices. The evaluation metrics include the RMSE Average and RMSE Standard Deviation of all traffic pattern scenarios.

Table 4.3: Comparative Performance Analysis of the Models Across Different Microservices

Model	Video Streaming microservice		PHP Back-end microservice		Web microservice	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	4.18	4.09	3.25	4.70	1.15	1.44
ARIMA	4.17	4.07	1.79	1.78	1.05	1.44
RARMA	4.25	4.01	7.30	13.44	1.21	1.67
RARIMA	4.29	4.06	1.89	1.86	1.13	1.84
REMA	9.37	11.17	10.53	18.44	2.82	4.91
KF	4.55	4.39	12.75	25.98	1.35	1.63
RKF	4.53	4.37	12.45	25.97	1.34	1.62
LSTMw5	9.24	10.75	2.17	1.97	3.19	8.58
LSTMw10	10.15	11.20	2.29	2.07	4.26	8.89
LSTMw20	19.75	31.91	3.20	2.77	5.76	16.15
RFw5	10.85	13.14	3.26	3.06	3.40	7.25
RFw10	11.56	14.22	2.90	3.21	3.61	7.43
Rolling LSTMw5	6.42	7.15	2.61	2.29	2.30	4.03
Rolling LSTMw10	6.77	8.35	6.60	9.83	4.34	10.87
Rolling LSTMw20	10.46	16.78	5.04	6.16	7.16	16.22
Rolling RFw5	7.31	8.56	2.09	1.90	1.60	2.94
Rolling RFw10	7.69	9.67	2.02	1.94	1.94	3.77

According to the results of Table 4.3, ARIMA performs best, in terms of both RMSE Average and Standard deviation, across all three microservices. On the other hand, the ML models (LSTM and RF) with varying window sizes demonstrate a higher degree of variability. For instance, in the Video Streaming

microservice, the LSTM model with a window size of 20 (LSTMw20) has the highest RMSE average and standard deviation among all the models. Similarly, in the PHP Back-end and Web microservices, the LSTM models with larger window sizes tend to have higher RMSE averages.

As for the rolling ML models, their performance seems to be mixed, depending on the specific microservice. For the PHP Back-end microservice, the Rolling RF model with a window size of 5 (Rolling RFw5) performs exceptionally well, with the second-lowest RMSE average and a relatively low standard deviation. Similarly, for the Web microservice, the Rolling RFw5 model delivers a very competitive performance with the second-lowest RMSE average and standard deviation. However, for the Video Streaming microservice, the rolling models do not perform as well as the ARIMA model.

In conclusion, the ARIMA model generally outperforms both the ML and rolling ML models across the three microservices in terms of RMSE average. The ML models, especially LSTM with larger window sizes, tend to have higher variability, as indicated by higher RMSE averages and standard deviations.

It should be highlighted that the results obtained for each application are influenced by the type of resources used and the sensitivity of each application to these resources. As such, variations in performance may stem from the specific characteristics and requirements of each application. With this understanding, we will continue our analysis, further investigating the impact of different types of resources on the performance of these models.

Results per resource type considering all applications

Here, we provide an analysis of the performance results obtained from the models, considering CPU, Memory, and Network (Bandwidth) metrics across different microservices. The key observations regarding the results of Table 4.4 are:

- The Rolling Random Forest with window 5 (Rolling RFw5) model shows remarkable performance on the CPU metric, with the lowest RMSE average, however, it is not significantly different from Rolling ARIMA which has RMSE 1.12 and the lowest standard deviation (1.45). Regarding to the Memory metrics, the Rolling LSTM with window 5 has superior

Table 4.4: Performance Analysis of Models for CPU, Memory, and Network Metrics

Model	CPU		Memory		Network	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	1.64	3.10	1.18	Winner 1.29	6.17	4.96
ARIMA	1.49	1.57	1.17	1.79	Winner 5.13	Winner 3.94
RARMA	3.22	8.49	1.25	1.66	5.85	4.83
RARIMA	1.12	Winner 1.45	1.14	1.69	5.30	4.01
REMA	4.60	11.74	3.07	6.99	11.92	9.95
KF	5.45	16.18	1.38	1.80	5.52	4.23
RKF	5.33	16.12	1.38	1.80	5.49	4.21
LSTMw5	1.18	1.58	1.99	4.49	14.00	12.88
LSTMw10	1.51	2.13	2.74	6.78	15.75	11.41
LSTMw20	1.79	2.36	8.74	26.90	22.84	27.46
RFw5	1.63	2.35	3.28	9.56	14.70	11.10
RFw10	1.52	2.30	3.76	11.20	15.31	11.03
Rolling LSTMw5	1.40	1.86	Winner 1.05	2.13	9.85	7.15
Rolling LSTMw10	2.91	6.48	2.55	6.64	12.39	8.35
Rolling LSTMw20	3.65	4.96	2.88	7.59	19.75	16.78
Rolling RFw5	Winner 1.11	1.51	1.88	4.87	8.99	8.56
Rolling RFw10	1.16	1.57	2.68	7.56	9.01	9.67

performance, demonstrating the smallest RMSE average. These models successfully capture and utilize the inherent patterns and relationships present within the CPU and Memory data.

- Regarding to the network metrics statistical models like ARIMA and the Rolling ARIMA showcase superior performance. This distinction arises due to the influence of network bandwidth, which is not only impacted by the type of application but is also susceptible to the complexities of traffic patterns. These patterns do not exhibit consistent trends that can be easily detected by machine learning models. Hence, traditional

statistical models like ARIMA, along with their rolling counterparts excel in capturing the different behavior of network bandwidth.

These findings underscore the significance of selecting appropriate models and considering the specific characteristics of the metrics being analyzed. Rolling mechanisms, with their adaptability to emerging trends, prove advantageous in capturing dynamic patterns. However, it is crucial to recognize that the performance of models can vary based on the unique nature of each microservice and resource type and the inherent challenges they present.

Result by application and resource type

Here we provide an in-depth analysis of the performance results obtained from various models for different microservices, focusing on the metrics related to CPU, Memory, and Network. The analysis encompasses the applications of Video Streaming, PHP back-end, and Web, highlighting the performance of each model for specific resource types.

Table 4.5 presents the performance analysis of models for Video Streaming metrics resource consumption, including CPU, Memory, and Network, as well as the performance analysis of models for PHP microservice encompassing the CPU and Memory. In the PHP application, we did not consider the network metrics as all metrics were 0. This is because the PHP application was printing an "ok" via HTTP after the back-end execution without transferring additional data.

Also, according to the results of Table 4.5, we observe that the RMSE values of the CPU in case of the case of Video Streaming and the memory of PHP application are both remarkably low, not surpassing 3 and 1, respectively. This is indicative of the nature of these applications: Video Streaming is not CPU-intensive, while PHP is not demanding in terms of memory usage. In contrast, the RMSE in the network for the Video Streaming application and the CPU for the PHP application exhibit a wide range of values, this is because the resource usage of them in experiments spanning from 0 to 100%.

Upon analyzing the results obtained from Tables 4.5 and 4.6, our previous observations are further verified, providing a deeper understanding of the performance of different models across various applications and resource types.

Table 4.5: Performance Analysis of Models for Video Streaming and PHP back-end microservices with respect to CPU, Memory and Network metrics

Model	Video CPU		Video Memory		Video Network		PhP CPU		PhP Memory	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	0.93	1.27	2.30	Winner 1.34	9.32	2.98	5.24	5.42	0.27	0.03
ARIMA	0.89	1.20	Winner 2.44	1.61	9.18	2.21	Winner 2.85	1.49	0.22	0.06
RARMA	0.99	1.41	2.75	2.14	9.91	2.22	11.99	16.69	0.26	Winner 0.01
RARIMA	0.94	1.30	2.80	2.17	Winner 9.12	Winner 2.16	3.02	1.49	0.21	0.08
REMA	1.10	1.59	8.13	11.27	18.89	10.65	17.40	22.42	0.22	0.07
KF	0.91	1.19	2.91	2.29	9.82	2.40	21.09	33.00	0.25	0.05
RKF	0.91	1.19	2.91	2.29	9.77	2.40	20.59	33.17	0.25	0.05
LSTMw5	0.94	1.33	5.27	7.22	21.51	7.48	3.42	1.39	0.30	0.16
LSTMw10	1.86	3.00	7.33	11.23	21.28	7.56	3.64	1.30	0.26	0.07
LSTMw20	1.47	2.40	24.77	46.23	33.00	29.62	5.13	1.15	0.31	0.12
RFw5	0.95	1.43	9.24	16.25	22.35	7.54	5.26	1.94	0.26	0.15
RFw10	0.96	1.39	10.67	19.10	23.04	6.96	4.69	2.94	0.23	0.11
Rolling LSTMw5	1.04	1.56	3.17	3.02	15.07	4.83	4.20	1.04	0.23	0.07
Rolling LSTMw10	0.97	1.31	7.07	10.98	12.26	6.96	10.86	11.19	0.21	0.08
Rolling LSTMw20	2.94	5.35	8.02	12.57	20.40	25.11	8.25	6.11	0.23	0.11
Rolling RFw5	Winner 0.83	Winner 1.18	5.10	8.14	16.00	6.14	3.33	Winner 1.19	0.23	0.11
Rolling RFw10	1.02	1.55	7.45	12.80	14.59	7.35	3.23	1.43	Winner 0.20	0.08

Table 4.6: Performance of models for Web microservices with respect to CPU, Memory, and Network metrics

Model	Web CPU		Web Memory		Web Network	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	0.31	0.34	0.74	1.02	2.80	2.71
ARIMA	0.29	0.35	0.44	0.56	Winner 2.43	Winner 1.78
RARMA	0.32	0.34	0.58	0.77	2.74	2.13
RARIMA	0.30	0.34	0.35	0.44	2.75	2.54
REMA	0.54	0.64	0.65	0.81	7.28	6.73
KF	0.65	0.90	0.74	1.06	2.65	2.00
RKF	0.65	0.90	0.74	1.05	2.64	2.00
LSTMw5	0.23	0.14	0.36	0.32	8.98	13.77
LSTMw10	0.22	0.13	0.52	0.56	12.06	12.61
LSTMw20	0.34	0.28	0.87	1.18	16.07	26.33
RFw5	0.28	Winner 0.11	0.33	0.30	9.60	10.47
RFw10	0.32	0.15	0.34	0.32	10.16	10.50
Rolling LSTMw5	0.23	0.12	0.31	0.32	6.36	5.04
Rolling LSTMw10	0.23	0.13	0.31	0.30	12.48	16.81
Rolling LSTMw20	1.83	3.15	0.33	0.31	19.31	24.84
Rolling RFw5	Winner 0.19	0.12	Winner 0.29	Winner 0.28	4.32	3.98
Rolling RFw10	0.21	0.12	0.32	0.32	5.29	5.28

We observe that both Rolling LSTM and Rolling RF models outperform their non-rolling counterparts in terms of their overall performance. It's noteworthy to mention that the window size also plays a significant role in determining the performance of these models. Evidently, a smaller window size seems to enhance the performance for both the LSTM and Random Forest models.

When focusing on the Video streaming service (Table 4.5), the simpler statistical models like ARIMA and ARMA, seem to exhibit superior performance, especially respect to the network. The Rolling ARIMA (RARIMA) model, in particular, presents the best performance with the lowest RMSE average of 2.44 in the network. This could be attributed to the Video streaming service's

high sensitivity to changes in bandwidth. This microservice is also influenced by various traffic patterns, introducing new trends that are often challenging for prediction models to promptly detect.

However, when we examine CPU performance for the Video microservice, the scenario changes slightly. The Rolling LSTMw5 model comes out on top, but the margin between this model and other models, including non-rolling ones, is not drastic. This pattern is similarly observed in the PHP memory performance category.

In the case of PHP Back-end application's CPU performance, the ARIMA model is better once again. However, it's worth mentioning that even though PHP is highly CPU sensitive, the performance gap between ARIMA and non-Rolling ML models was not significant. This suggests that non-rolling approaches could be effectively utilized in this context, offering the advantage of lower computational costs.

Regarding to the results of the Web application (Table 4.6), the conclusions are common with the two preceding microservices. Non-rolling mechanisms perform exceptionally well for both CPU and memory, thus allowing the use of such a mechanism for these resources. Nevertheless, in the case of Network, ARIMA had by far the best performance.

Concluding to the results, we observe that the ideal model selection for prediction is highly contingent on the specific resource needs of each microservice. The one-size-fits-all approach, such as rolling models with high computational costs, may not be necessary or optimal in all situations. The choice should be determined based on the specific microservice's requirements and the nature of the resources in question.

In figures 4.1, 4.2, and 4.3 we illustrate the performance of three distinct predictive models LSTM, Rolling LSTM, and ARIMA across all the resource categories (CPU, Memory, and Network) for the Video Streaming application subjected to the Gradually Decreasing load scenario.

In Figure 4.1 (a), we observe that LSTM model struggles to predict resources accurately during decreasing load conditions, whereas the Rolling LSTM model significantly improves upon this, as indicated in Figure 4.1 (b). The ARIMA model, given its retraining regimen that includes all past data (inclusive of the training phase), has the best performance.

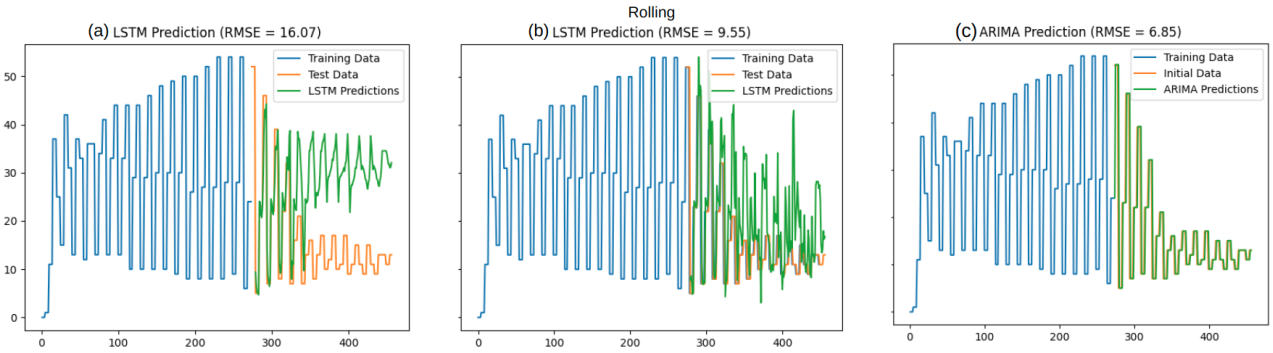

Figure 4.1: Prediction of Network Metrics for Video Streaming Application using LSTM, Rolling LSTM and ARIMA Models

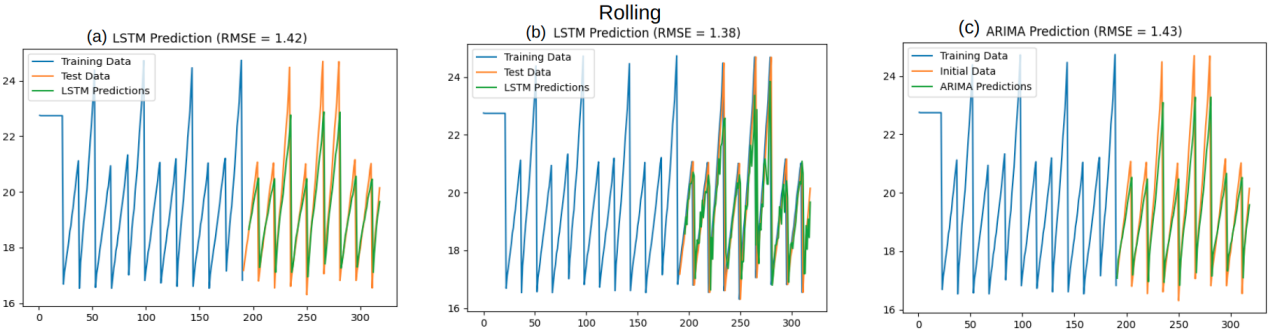

Figure 4.2: Prediction of CPU Metrics for Video Streaming Application using LSTM, Rolling LSTM and ARIMA Models

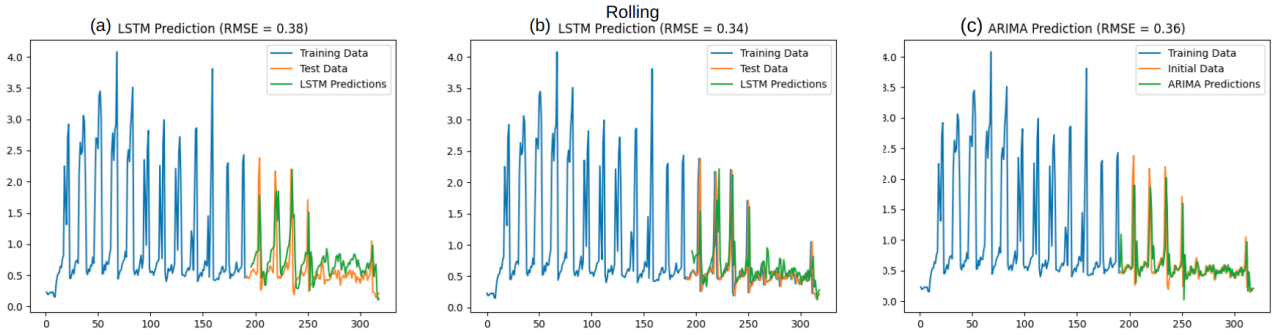

Figure 4.3: Prediction of Memory Metrics for Video Streaming Application using LSTM, Rolling LSTM and ARIMA Models

However, this performance stratification does not hold consistent across all resource types in the same scenario. As shown in Figures 4.3 and 4.2, for CPU and Memory forecasting, the non-rolling LSTM model - despite its lower computational cost and minor variation in forecast duration - matches the performance of its counterparts.

In summary, our detailed analysis reaffirms that the performance of models varies across different applications and resource types. ML models excel in capturing specific trends and patterns observed in CPU and Memory metrics, while statistical models such as ARIMA and ARMA outperform ML models when it comes to Network metrics influenced by complex traffic patterns.

4.5.2 Evaluation Results of Multi-step prediction

In this subsection, we delve into the results of the multi-step prediction processes. Our focus shifts to using the previously discussed models to forecast the first 5 values of the testing data. Notably, the analysis excludes the Kalman filter as it is not suitable for multi-step prediction. Furthermore, we exclude the rolling approaches for ML models (LSTM and Random Forest) since we only predict the initial 5 values once, so there is no need for retraining.

Regarding to the difference between the Rolling ARMA/ARIMA models and the simple ARMA/ARIMA models, the rolling models utilize the most recent 50 values from the testing data to generate predictions, whereas the simple ARMA/ARIMA models incorporate the entire training data without a rolling window approach.

By focusing on the multi-step predictions, we gain insights into the models' ability to project future values beyond a single time step. This analysis provides a broader perspective on their forecasting capabilities and informs us about their performance in longer-term predictions.

Average Time, RMSE and MAE results considering all scenarios

Table 4.7 presents the computational time in seconds required by each model for the prediction processes. The RFw5 and RFw10 models exhibit the lowest computational times, with 0.014 seconds and 0.015 seconds, respectively. These models are followed by RARMA and RARIMA, which require slightly more

computational time at 0.022 seconds and 0.028 seconds, respectively.

In terms of the statistical models, ARIMA and ARMA both take longer computational times, with 0.040 seconds and 0.041 seconds, respectively. The LSTM models, namely LSTMw10, LSTMw5, and LSTMw20, require the most substantial computational time among the models listed, with 0.090 seconds, 0.106 seconds, and 0.110 seconds, respectively.

Table 4.7: Computational Time Comparison of Multi-step Prediction Models

MODEL	**TIME Seconds**
RFw5	0.014
RFw10	0.015
RARMA	0.022
RARIMA	0.028
ARIMA	0.040
ARMA	0.041
LSTMw10	0.090
LSTMw5	0.106
LSTM20	0.110

Table 4.8: Analysis of RMSE and MAE Performance Across the Multi-step Prediction Models

	RMSE		**MAE**	
Model	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
LSTM20	1.89	3.52	1.63	3.16
LSTMw10	2.08	4.19	1.76	3.55
RFw10	2.18	5.80	1.92	5.55
LSTMw5	2.36	4.88	2.14	4.60
RARMA	2.80	6.48	2.45	5.50
ARIMA	2.89	7.80	2.60	7.11
ARMA	2.90	7.13	2.65	6.67
RARIMA	2.90	7.74	2.68	7.42
RFw5	3.27	8.28	2.91	8.03

Regarding to the results of the RMSE and MAE (Table 4.8) several observations can be made:

- ML Models Outperform Statistical Models: The ML models, such as LSTM, demonstrate superior performance in terms of RMSE and MAE

compared to the statistical models. This indicates that the ML models are better able to capture the underlying patterns and dynamics of the data, resulting in more accurate predictions.

- Impact of Window Size: Larger window sizes tend to yield better results in both LSTM and RF models. This suggests that considering a larger context of historical data enables the models to capture long-term dependencies and trends, leading to improved prediction accuracy.

It is important to note that the specific performance of each model can vary depending on the dataset and application context. However, based on these results, it is evident that ML models with larger window sizes exhibit better performance in terms of RMSE and MAE compared to the statistical models and smaller window sizes.

These findings highlight the potential benefits of leveraging ML models, particularly those with larger window sizes, for accurate and reliable multi-step predictions. However, further analysis and experimentation may be required to optimize model performance based on specific application requirements and datasets.

Results per application considering all types of resources

In this section, we detail the results of the models' assessments across different microservices. In Table 4.9, we observe that different models excel in different applications. The model performance does not follow a one-size-fits-all pattern but rather depends on the type of microservice and the type of resources being considered.

Specifically, in the Video application, the Rolling ARMA (RARMA) model appeared to be the most efficient, demonstrating the lowest average RMSE of 3.45 and the lowest standard deviation of RMSE at 3.72. This highlights the suitability of the Rolling ARMA model for capturing the complex dynamics of the video microservice.

In the PHP-backend application, the LSTM model with a window size of 5 (LSTMw5) performed the best, exhibiting the lowest average RMSE of 1.81 and the lowest standard deviation of RMSE at 2.28. The LSTM model's

Table 4.9: Multi-step Model Performance by microservice type: Results for All Resource Types

Model	Video		PHP Back-end		Web	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	4.05	6.32	7.36	14.82	0.64	1.03
ARIMA	4.04	8.16	7.29	15.13	0.66	1.33
RARMA	Winner 3.45	Winner 3.72	7.75	14.38	0.72	1.80
RARIMA	4.15	8.12	7.24	14.93	0.61	1.33
LSTMw5	5.37	7.43	Winner 1.81	Winner 2.28	0.54	1.28
LSTMw10	4.18	6.51	2.23	2.38	0.63	1.29
LSTM20	3.68	4.88	2.41	4.02	Winner 0.52	Winner 1.00
RFw5	7.10	12.92	3.70	5.89	0.57	2.18
RFw10	4.25	9.05	3.01	4.59	0.52	1.94

ability to capture long-term dependencies and patterns within the PHP-backend microservice contributed to its superior performance.

For the Web application, the LSTM model with a window size of 20 (LSTMw20) provided the most accurate predictions, demonstrating the lowest average RMSE of 0.52 and the lowest standard deviation of RMSE at 1.00. This highlights the LSTM model's effectiveness in capturing the intricate relationships and patterns within the Web microservice.

However, it is noteworthy that the differences between the top-performing model and the other models were not markedly significant, indicating a competitive performance across all models in each application. This suggests that various models can achieve reliable predictions for different microservices, showcasing their versatility and adaptability.

In conclusion, the decision for the best model for multi-step prediction should be driven by the specific application and the type of resources in focus. Each microservice and resource type impacts the resource requirements differently, necessitating a model that best fits their unique characteristics. Therefore, it is important to consider these factors when selecting the appropriate model for the task.

Below, we continue the analysis of model performance by resource type to

gain further insights into their capabilities and limitations.

Results per resource type considering all resources

Table 4.10 presents a performance analysis of various models based on different resource types. According to the results of ML models vs. statistical models across all resource types, ML learning models consistently outperform simple statistical models in terms of prediction accuracy. This highlights the effectiveness of ML models in capturing the complex patterns and dynamics of resource data for multi-step prediction.

Also, the optimal window size varies based on the resource type. For memory and bandwidth, larger window sizes yield better performance. This is because memory and bandwidth exhibit longer-term dependencies and trends that can be captured by models with larger windows. On the other hand, for CPU, smaller window sizes are more effective. CPU behavior often involves rapid changes and short-term fluctuations, such as CPU bursts, which are better captured by models with smaller windows.

Table 4.10: Performance Analysis of Multi-step Models by Resource Type

Model	**CPU**		**Memory**		**Network**	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	3.58	9.87	0.83	1.13	4.41	6.83
ARIMA	3.55	10.04	0.45	0.42	4.90	8.80
RARMA	3.79	9.68	0.85	1.29	3.76	4.28
RARIMA	3.55	9.92	0.47	0.46	4.90	8.80
LSTMw5	Winner 1.30	Winner 2.26	0.76	1.05	5.76	8.11
LSTMw10	1.41	2.05	0.64	1.09	4.75	7.02
LSTM20	1.77	3.68	0.85	1.58	Winner 3.30	Winner 4.68
RFw5	2.11	4.18	0.60	1.03	8.09	14.23
RFw10	1.72	3.34	Winner 0.14	Winner 0.15	5.27	9.90

Result by microservice and resource type

Here, we provide the performance analysis of the models per microservice and resource type.

Table 4.11: Performance Analysis of Multi-step Forecasting Models for Video Microservice Resources

Model	Video CPU		Video Memory		Video Network	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	Winner 1.34	Winner 2.27	1.43	1.68	9.37	9.03
ARIMA	1.41	2.29	0.54	0.35	10.18	12.76
RARMA	1.46	2.32	2.05	1.71	6.84	Winner 4.37
RARIMA	1.45	2.39	0.82	0.30	10.18	12.76
LSTMw5	1.91	3.17	1.72	1.41	12.47	9.47
LSTMw10	1.57	2.18	1.41	1.72	9.56	9.46
LSTM20	2.61	5.07	1.95	2.58	Winner 6.48	6.25
RFw5	1.82	2.65	1.58	1.39	17.91	19.22
RFw10	1.49	2.43	Winner 0.16	Winner 0.09	11.10	14.13

Table 4.12: Performance Analysis of Multi-step Forecasting Models for PHP back-end Microservices Resources

Model	PHP CPU		PHP Memory	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	10.81	17.84	0.47	0.33
ARIMA	10.70	18.30	0.45	0.11
RARMA	11.44	17.03	0.37	0.01
RARIMA	10.63	18.04	0.46	0.23
LSTMw5	Winner 2.48	2.61	0.47	0.30
LSTMw10	3.18	Winner 2.41	0.32	0.20
LSTM20	3.45	4.75	0.34	0.18
RFw5	5.41	6.79	Winner 0.29	Winner 0.01
RFw10	4.33	5.31	0.39	0.16

- In the Video microservice (Table 4.11), the ARMA model performs better

Table 4.13: Performance Analysis of Multi-step Forecasting Models for Web Microservices Resources

Model	Web CPU		Web Memory		Web Network	
	RMSE Average	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**	**RMSE Average**	**RMSE ST.DEV**
ARMA	0.27	0.43	0.55	0.81	Winner 1.11	Winner 1.52
ARIMA	0.22	0.37	0.39	0.55	1.38	2.14
RARMA	0.25	0.42	0.22	0.45	1.70	2.98
RARIMA	0.22	0.38	0.25	0.49	1.38	2.14
LSTMw5	0.11	0.07	0.22	0.29	1.29	2.12
LSTMw10	0.12	0.09	0.23	0.32	1.54	2.00
LSTM20	Winner 0.08	Winner 0.03	0.30	0.24	1.17	1.59
RFw5	0.12	0.07	0.05	0.02	1.55	3.79
RFw10	0.13	0.10	Winner 0.04	Winner 0.02	1.38	3.38

for CPU resource prediction, with the lowest average RMSE and ST.DEV. In terms of memory resource prediction, RFw10 outperforms other models, while RARMA takes the lead in network resource prediction.

- In the PHP microservice (Table 4.12), which is CPU-intensive, the LSTM model with a window size of 5 (LSTMw5) demonstrates the best performance in predicting CPU resource usage. For memory resource prediction, the Random Forest model with window size 5 (RFw5) exhibits the lowest average RMSE and ST.DEV.

- For the Web microservice (Table 4.13), ARMA achieves the best performance in predicting network resource usage with a small difference from machine learning models. LSTM20 excels in CPU resource prediction, and RFw5 performs best in predicting memory resource usage.

In summary, the analysis of model performance across microservices and resource types generally confirms our previous observations. Machine learning models tend to outperform simple statistical models, with the exception of the Video CPU and Web network resource predictions, where ARMA has the best performance but with a small margin from the ML models. Additionally, in the CPU-intensive PHP back-end microservice, the LSTM model with a small window (window size 5) exhibits the best performance.

4.6 Conclusions

In conclusion, our study presents the performance dynamics of several statistical and ML models across different microservices and resource types. Our analysis points towards significant variability, wherein given models excel under particular conditions and applications, but may not perform as well under others.

In particular, the ML models under investigation, i.e., LSTM and Random Forest, perform robustly in predicting CPU and Memory metrics, while the involved statistical models, i.e., ARIMA and KF, demonstrate superior performance in Network metrics affected by the clients' distribution. We further note that the performance of these models is influenced by factors such as window size and the rolling or non-rolling nature of the model, implying that the choice of model and its configuration should be informed by the specific application and resource types in focus.

For instance, the ARIMA model, when retrained with all past data, shows robust performance for Video Streaming applications, especially under decreasing load conditions. However, in the same scenario, the non-rolling LSTM model, despite having lower computational cost, demonstrates comparable performance for CPU and Memory forecasting.

Moreover, our study suggests that the 'one-size-fits-all' approach, such as defaulting to models with high computational costs, may not be always optimal. The decision for the best-suited model for prediction should be driven by the specific microservice's requirements and the nature of the resources in focus, offering potential avenues for balancing computational cost and predictive accuracy.

Regarding to the multistep prediction, ML models generally outshine simple statistical models across microservices and resource types, barring a few exceptions, making them a safer choice for multi-step prediction.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

This final chapter concludes the thesis, presenting and summarizing its scientific contributions, and describing relevant future research directions. The primary focus of the thesis is to enhance the adaptability of applications to the conditions of wireless mesh networks (WMNs), enable microservices adaptive load balancing, and predict the impact of various microservices on network and processing resources.

Regarding to the application adaptability in wireless mesh networks, we developed and evaluated experimentally two technological solutions based on SDN: (i) a reactive solution that is updated on the current state of the wireless network and appropriately adjusts the paths of network nodes; and (ii) a proactive solution where the paths are pre-selected based on classification wireless links considering real measurements in terms of delay and signal strength (RSSI).

Our evaluation demonstrated that the reactive strategy offers advantages such as reliability, fault tolerance, and adaptation to unstable conditions like topology rearrangements, through timely detection of network changes such as connection failures. However, it also poses some challenges, including the control management overhead, the need to implement a reliable channel for the control plane, and high complexity, which makes it more suitable for small-scale topologies. On the other hand, the proactive strategy bases its routing decisions on delay and RSSI quality measurements, making it suitable for larger

topologies with lower complexity and control overhead. However, this strategy faces difficulties in adapting to dynamic changes, since it largely ignores the current network state and does not detect topology changes promptly.

In the context of efficient traffic steering among microservices, the research introduces and investigates Microservice-Aware Load Balancing (MALB), aiming at addressing the stringent resource-efficiency and service performance requirements of beyond 5G networks. The main feature of MALB is the ability to adapt the workload/traffic to specific resource demands of each microservice type, thus achieving the best resource allocation and application performance, applying different load-balancing policies depending on the characteristics of each microservice. Through our experimental analyses, our research demonstrates that the performance advantages of MALB in terms of CPU, memory, and network utilization, and also in terms of response times and application-layer throughput of microservices.

Also, the research analyzed the performance dynamics of both statistical and Machine Learning (ML) models across diverse microservices and resource types. While ML models, particularly LSTM and Random Forest, performed robustly in predicting CPU and Memory metrics, statistical models such as ARIMA and ARMA were found superior in Network metrics predictions, strongly affected by traffic patterns. It was noted that the performance of these models is influenced by factors such as window size and the rolling or non-rolling nature of the model. This insight indicates the necessity of tailoring model choice and configuration to the specific application and resource types in focus.

Moreover, the research suggests moving away from the 'one-size-fits-all' approach, emphasizing that the decision for the best-suited model for prediction should consider the specific microservice's requirements and the nature of the resources.

In summary, this thesis has made significant contributions to understanding the dynamics of wireless mesh networks, improving resource utilization in microservices, and developing efficient predictive models for resource allocation. Future research should continue to advance these fields, taking into consideration the increasingly dynamic nature of network environments and the growing complexity of microservice architectures. This will help in meeting the performance and efficiency demands of next-generation Internet applications.

In the next section, we describe future plans and directions arising from this thesis.

5.2 Future works

Here, we also discuss potential future work inspired by the findings and limitations of the present study.

- **Adaptive MALB for WMNs**: Future work could consider developing an adaptive MALB specifically designed for WMNs. This approach could consider the unique characteristics and constraints of WMNs in balancing the load among microservices. The ability of such a system to adapt to the dynamic network conditions of WMNs could be a significant advancement in ensuring efficient resource allocation and usage.
- **Customizable Prediction Models in MALB**: Given the variability in the performance of statistical and machine learning models across different microservices and resource types, future work should focus on integrating customizable prediction models into MALB. This could involve a dynamic model selection process based on the specific microservice's requirements and the nature of the resources, thereby achieving an optimal balance between computational cost and predictive accuracy.
- **Advanced Clustering Techniques**: The utility of clustering in identifying optimal NDN paths over a stable WMN provides a promising research area. More advanced clustering techniques, considering also AI models, incorporating additional metrics could be explored to further improve path selection and overall network performance.
- **Hybrid Reactive-Proactive Strategies**: This thesis has shown the merits of both reactive and proactive strategies under different network conditions. Future research could investigate hybrid strategies that leverage the advantages of both approaches, dynamically switching between reactive and proactive modes based on real-time network conditions.
- **Predictive Load Balancing in WMNs**: An exciting area of future research could be the integration of multi-step prediction models into the

load balancing process of applications in WMNs. This approach could help predict future demands and optimally distribute the load in advance, thereby increasing the overall efficiency and performance of the network.

- **Integrated Framework for Beyond 5G Networks**: Finally, the development of an integrated framework that combines the adaptability of applications in WMNs, customizable prediction models, and adaptive load balancing could be a significant leap forward for beyond 5G networks. Such a framework could be capable of dynamically adjusting to network conditions, predicting resource requirements, and efficiently balancing the load among microservices.

References

[1] Jakob Struye, Bart Braem, Steven Latré, and Johann Marquez-Barja. The CityLab testbed — Large-scale multi-technology wireless experimentation in a city environment: Neural network-based interference prediction in a smart city. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 529–534, 2018.

[2] w-iLab.1 (OfficeLab) inventory. https://boss.wilab1.ilabt.iminds.be/inventory/, note = Accessed: 2022-11-10.

[3] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine*, 11(1):17–27, 2017.

[4] Network ITU. 2030-gap analysis of network 2030 new services, capabilities and use cases, 2020.

[5] Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Yasir Mehmood, Abdullah Gani, Salimah Mokhtar, and Sghaier Guizani. Enabling Communication Technologies for Smart Cities. *IEEE Communications Magazine*, 55(1):112–120, 2017.

[6] Cleitianne Silva, Yuri Oliveira, Clayson Celes, Reinaldo Braga, and Carina Oliveira. Performance evaluation of wireless mesh networks in smart cities scenarios. In *Proceedings of the Euro American Conference on Telematics and Information Systems*, pages 1–7, 2018.

[7] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco,

Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014.

[8] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[9] Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE Network*, 32(2):78–84, 2018.

[10] Taher M Ghazal, Mohammad Kamrul Hasan, Muhammad Turki Alshurideh, Haitham M Alzoubi, Munir Ahmad, Syed Shehryar Akbar, Barween Al Kurdi, and Iman A Akour. IoT for smart cities: Machine learning approaches in smart healthcare—A review. *Future Internet*, 13(8):218, 2021.

[11] Mennan Selimi, Leandro Navarro, Bart Braem, Felix Freitag, and Adisorn Lertsinsrubtavee. Towards Information-Centric Edge Platform for Mesh Networks: The Case of CityLab Testbed. In *2020 IEEE International Conference on Fog Computing (ICFC)*, pages 50–55, 2020.

[12] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys Tutorials*, 16(2):1024–1049, 2014.

[13] Ahed Aboodi, Tat-Chee Wan, and Gian-Chand Sodhy. Survey on the Incorporation of NDN/CCN in IoT. *IEEE Access*, 7:71827–71858, 2019.

[14] Boubakr Nour, Kashif Sharif, Fan Li, Sujit Biswas, Hassine Moungla, Mohsen Guizani, and Yu Wang. A survey of Internet of Things communication using ICN: A use case perspective. *Computer Communications*, 142:95–123, 2019.

[15] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Computer Communications*, 36(7):779–791, 2013.

[16] Asadullah Tariq, Rana Asif Rehman, and Byung-Seo Kim. Forwarding strategies in NDN-based wireless networks: A survey. *IEEE Communications Surveys & Tutorials*, 22(1):68–95, 2019.

[17] Wanqing Tu. Data-driven QoS and QoE management in smart cities: A tutorial study. *IEEE Commun. Mag.*, 56(12):126–133, 2018.

[18] Qingxia Chen, Renchao Xie, F Richard Yu, Jiang Liu, Tao Huang, and Yunjie Liu. Transport control strategies in named data networking: A survey. *IEEE Communications Surveys & Tutorials*, 18(3):2052–2083, 2016.

[19] Michael Baddeley, Reza Nejabati, George Oikonomou, Mahesh Sooriyabandara, and Dimitra Simeonidou. Evolving SDN for low-power IoT networks. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 71–79. IEEE, 2018.

[20] Sarantis Kalafatidis, Vassilis Demiroglou, Lefteris Mamatas, and Vassilis Tsaoussidis. Experimenting with an SDN-Based NDN Deployment over Wireless Mesh Networks. *IEEE INFOCOM WKSHPS CNERT*, 2022, 2022.

[21] Lefteris Mamatas, Vassilis Demiroglou, Sarantis Kalafatidis, Sotiris Skaperas, and Vassilis Tsaoussidis. Protocol-Adaptive Strategies for Wireless Mesh Smart City Networks. *IEEE Network*, 2022.

[22] Wireless Testlab and OfficeLab — imec iLab.t documentation. https://doc.ilabt.imec.be/ilabt/wilab/. Accessed: 2022-11-10.

[23] Wireless mesh performance measurement. https://github.com/SWNRG/wireless-mesh-performance-measurements. Accessed: 2022-11-15.

[24] Elian Aubry, Thomas Silverston, and Isabelle Chrismen. Implementation and Evaluation of a Controller-Based Forwarding Scheme for NDN. In *2017*

IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), pages 144–151, 2017.

[25] Mohammad Alhowaidi, Deepak Nadig, Byrav Ramamurthy, Brian Bockelman, and David Swanson. Multipath Forwarding Strategies and SDN Control for Named Data Networking. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6, 2018.

[26] Marica Amadeo, Claudia Campolo, Giuseppe Ruggeri, Antonella Molinaro, and Antonio Iera. Understanding Name-based Forwarding Rules in Software-Defined Named Data Networking. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[27] Won-Suk Kim, Chung Sang-Hwa, and Moon Jae-Won. Improved content management for information-centric networking in SDN-based wireless mesh network. *Computer Networks*, 92:316–329, 2015.

[28] Gaurav Verma, Arun Nandewal, and K Chandrasekaran. Cluster Based Routing in NDN. In *2015 12th International Conference on Information Technology-New Generations*, pages 296–301. IEEE, 2015.

[29] Zeinab Shariat, Ali Movaghar, and Mehdi Hoseinzadeh. A learning automata and clustering-based routing protocol for named data networking. *Telecommunication Systems*, 65(1):9–29, 2017.

[30] Xiaonan Wang, Xingwei Wang, and Yanli Li. NDN-based IoT with Edge computing. *Future Generation Computer Systems*, 115:397–405, 2021.

[31] Qing-Yi Zhang, Xing-Wei Wang, Min Huang, Ke-Qin Li, and Sajal K. Das. Software Defined Networking Meets Information Centric Networking: A Survey. *IEEE Access*, 6:39547–39563, 2018.

[32] BATMAN Concept - open-mesh - open mesh. https://www.open-mesh.org/projects/open-mesh/wiki/BATMANConcept. Accessed: 2022-11-10.

[33] Tryfon Theodorou, George Violettas, Polychronis Valsamas, Sophia Petridou, and Lefteris Mamatas. A Multi-Protocol Software-Defined Networking

Solution for the Internet of Things. *IEEE Communications Magazine*, 57(10):42–48, 2019.

[34] Tryfon Theodorou and Lefteris Mamatas. SD-MIoT: A software-defined networking solution for mobile Internet of Things. *IEEE Internet of Things Journal*, 8(6):4604–4617, 2020.

[35] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[36] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[37] Preeti Arora, Shipra Varshney, et al. Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, 78:507–512, 2016.

[38] NFD Overview — Named Data Networking Forwarding Daemon (NFD) 0.7.1 documentation. https://named-data.net/doc/NFD/current/overview.html. Accessed: 2022-11-10.

[39] Empowering App Development for Developers — Docker. https://www.docker.com/. Accessed: 2022-11-10.

[40] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, K.C. Ng, Vyas Sekar, and Scott Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 147–158, New York, NY, USA, 2013. Association for Computing Machinery.

[41] Wiki - alfred - Open Mesh. https://www.open-mesh.org/projects/alfred/wiki. Accessed: 2022-11-10.

[42] Joao F Santos, Wei Liu, Xianjun Jiao, Natal V Neto, Sofie Pollin, Johann M Marquez-Barja, Ingrid Moerman, and Luiz A DaSilva. Breaking down network slicing: Hierarchical orchestration of end-to-end networks. *IEEE Communications Magazine*, 58(10):16–22, 2020.

[43] Andres Garcia-Saavedra and Xavier Costa-Perez. O-RAN: Disrupting the virtualized RAN ecosystem. *IEEE Communications Standards Magazine*, 5(4):96–103, 2021.

[44] Qingwei Du and Huaidong Zhuang. OpenFlow-Based Dynamic Server Cluster Load Balancing with Measurement Support. *J. Commun.*, 10(8):572–578, 2015.

[45] Dzmitry Kliazovich, Sisay T Arzo, Fabrizio Granelli, Pascal Bouvry, and Samee Ullah Khan. e-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 7–13. IEEE, 2013.

[46] Yipei Niu, Fangming Liu, and Zongpeng Li. Load balancing across microservices. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 198–206. IEEE, 2018.

[47] Ruozhou Yu, Vishnu Teja Kilari, Guoliang Xue, and Dejun Yang. Load balancing for interdependent IoT microservices. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 298–306. IEEE, 2019.

[48] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637. IEEE, 2011.

[49] Shuo Wang, Jiao Zhang, Tao Huang, Tian Pan, Jiang Liu, and Yunjie Liu. FDALB: Flow distribution aware load balancing for datacenter networks. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE, 2016.

[50] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[51] Stuart Clayman, Alex Galis, and Lefteris Mamatas. Monitoring virtual networks with lattice. In *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pages 239–246. IEEE, 2010.

[52] Containernet. https://containernet.github.io/.

[53] Floodlight controller - project floodlight. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[54] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, 35(2):13–23, 2007.

[55] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. RPPS: A novel resource prediction and provisioning scheme in cloud data center. In *2012 IEEE Ninth International Conference on Services Computing*, pages 609–616. IEEE, 2012.

[56] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE transactions on cloud computing*, 3(4):449–458, 2014.

[57] Anoop S Kumar and Somnath Mazumdar. Forecasting HPC workload using ARMA models and SSA. In *2016 International Conference on Information Technology (ICIT)*, pages 294–297. IEEE, 2016.

[58] Yonghua Zhu, Weilin Zhang, Yihai Chen, and Honghao Gao. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. *EURASIP Journal on Wireless Communications and Networking*, 2019:1–18, 2019.

[59] K Valarmathi and S Kanaga Suba Raja. Resource utilization prediction technique in cloud using knowledge based ensemble random forest with LSTM model. *Concurrent Engineering*, 29(4):396–404, 2021.

[60] Jiechao Gao, Haoyu Wang, and Haiying Shen. Machine learning based workload prediction in cloud computing. In *2020 29th international con-*

ference on computer communications and networks (ICCCN), pages 1–9. IEEE, 2020.

[61] Weiwei Lin, Guangxin Wu, Xinyang Wang, and Keqin Li. An artificial neural network approach to power consumption model construction for servers in cloud data centers. *IEEE Transactions on Sustainable Computing*, 5(3):329–340, 2019.

[62] Deepak Janardhanan and Enda Barrett. CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models. In *2017 12th international conference for internet technology and secured transactions (ICITST)*, pages 55–60. IEEE, 2017.

[63] Alexandros Psychas, Phivos Dadamis, Nikolaos Kapsoulis, Antonios Litke, and Theodora Varvarigou. Containerised Application Profiling and Classification Using Benchmarks. *Applied Sciences*, 12(23):12374, 2022.

[64] Yulai Xie, Minpeng Jin, Zhuping Zou, Gongming Xu, Dan Feng, Wenmao Liu, and Darrell Long. Real-time prediction of docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing. *IEEE Transactions on Cloud Computing*, 10(2):1386–1401, 2020.

Appendices

Funding

This work received funding from EU's H2020 research and innovation programme through the 9th open call scheme of the Fed4FIRE+ project (grant agr. num. 732638). Also, it is supported by the "GSRI FUNDING FOR THE YEAR 2019 (Award for the participation in competitive E.U. projects)", Novel Enablers for Cloud Slicing - NECOS, GA No 777067, HORIZON 2020 - JOINT ACTION EU - BRAZIL, H2020-EUB-2017, Ministry of Development and Investments – General Secretariat for Research and Innovation. It is also co-funded by Greece and the European Union (European Social Fund-ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of action "Enhancing Human Resources Research Potential by undertaking a Doctoral Research," sub-action 2: "IKY Scholarship Programme for Ph.D. candidates in Greek Universities".