UNIVERSIDADE D
COIMBRA

Karima Daniela Velasquez Castro

MECHANISMS FOR LATENCY REDUCTION IN FOG ENVIRONMENTS

Tese no âmbito do Programa de Doutoramento em Ciências e Tecnologias da Informação, orientada pela Professora Doutora Marilia Curado, e pelo Professor Doutor Edmundo Monteiro, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Dezembro de 2020

UNIVERSIDADE D
COIMBRA

Department of Informatics Engineering
Faculty of Sciences and Technology
University of Coimbra

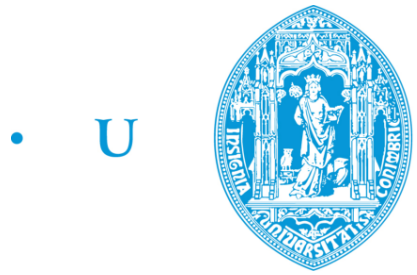
Mechanisms for Latency Reduction in Fog Environments

Karima Daniela Velasquez Castro

Doctoral Program in Information Science and Technology
PhD Thesis submitted to the University of Coimbra

Advised by Prof. Dr. Marilia Curado
and Prof. Dr. Edmundo Monteiro

December, 2020



Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Mecanismos de Redução de Latência em Ambientes de Nevoeiro

Karima Daniela Velasquez Castro

Programa de Doutoramento em Ciências e Tecnologias da Informação
Tese de Doutoramento apresentada à Universidade de Coimbra

Orientado pela Prof. Dr. Marilia Curado
e pelo Prof. Dr. Edmundo Monteiro

Dezembro, 2020

This work was partially supported with scholarships by the Portuguese Foundation for Science and Technology (FCT) under the projects grants CENTRO-07-ST24-FEDER-002003, PTDC/EEI-SCR/6453/2014; as well as under the PhD grant SFRH/BD/119392/2016. Additionally by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under the project grant CAPES-FCT/8572/14-3 and by the Portuguese Foundation for Science and Technology (FCT) under the project grant MITP TB/CS/0026/2013.

Acknowledgments

TAKING a look back at the last couple of years during the road literally traveled to perform this PhD, there are some people without whom this experience would not have been quite the same.

To mom, dad, and my two awesome sisters, Adriana and Gabriela; you are my cheerleading squad. Thank you for always believing in me and making me believe that I can accomplish anything, and for always being by my side, in any way or form.

To Cesar Alejandro, you bring the purest joy to my heart. Thank you for making me look at things with a new perspective and for always making me laugh.

To Prof. Luís Paquete, thank you for your selfless help. Your efficiency and quickness to approach problems amazes me.

To Prof. Edmundo Monteiro, you are the prove that doing high quality research does not have to be boring. Thank you for your wise input; every meeting was a learning experience.

To Prof. Marilia Curado, you are a role model. You have a gracious and highly effective approach for leading, pushing me to my limits in the best possible way. Thank you for showing me it is possible to be a badass boss-lady in a field filled with men.

To my LCT family, Mariana, Marcelo, Vitor, Proença, Rui, Duarte, Paulo, Tina, and DavidBR. Thank you for the many meals, canoeing trips, hikes, and every activity (even the laser tag, which I hated); you guys made all the difference.

To Ricardo, you are the kindest person I know, and I am proud to call you my friend. When I grow up, I want to be like you.

To David, I could not have done this without you. *Somos equipo.*

Abstract

A new industrial revolution driven by digital data, computation, and automation has arrived. Human activities, industrial processes, and research lead to data collection, generation, and processing on an unprecedented scale, spurring new products, services, and applications. Among the applications that generate more traffic are those related to augmented/virtual reality and video streaming. These applications have strict time restrictions to perform in a manner that is expected by final users, and usually rely on the use of Cloud computing to achieve elasticity, on-demand self-service, resource pooling, and timely delivery. However, new generation delay-sensitive applications and services have requirements that are only partially met by existing Cloud computing solutions.

In recent years there has been a paradigm shift to bring Cloud services towards the edge of the network. In this peripheral area, there is an abundance of heterogeneous resource-constrained devices both generating and consuming data. This represents an increment on the amount of data, that would lead to increased traffic and response time to transport to the Cloud and back. It is possible to place storage and processing devices at the rim of the network to help preprocess this data and alleviate the load sent towards the core network, while also reducing response times which particularly benefits delay-sensitive applications. This solution is known as Fog computing.

Fog computing is an important paradigm to help address the requirements that are not completely covered by the Cloud; nonetheless, the use of this technology creates new challenges. The Fog needs to support the orchestration of applications and services on demand, with adaptability, while providing flexible and time constrained performance. In practice, traditional service orchestration approaches that were applied to Cloud services are not suitable for the large scale and dynamism of Fog services. This creates the need for new mechanisms for the coordination of resources, applications, and services in the Fog. This work proposes a smart orchestration framework based on a hybrid approach that combines centralized orchestration and distributed choreography to deal with resource management, as well as a service orchestrator architecture that includes a depiction of its different modules and the interactions among them.

To cope with the time constraints for delay-sensitive applications, the orchestrator must optimize where applications and services are deployed. One of the modules of the service orchestrator architecture proposed in this work is in charge of the decision regarding the final placement of application services in a smart and context-aware manner with the objective of minimizing latency. Particularly, this work presents three different mechanisms for service placement in the Fog ecosystem, namely, one based in Integer Linear Programming, one based

in Genetic Algorithms, and one based in graph partition using the PageRank algorithm. All three mechanisms use a combination of metrics that evaluate characteristics of both the applications as well as the network infrastructure to guide the placement process. From the network perspective, the propagation delay is considered; and from the application perspective, the popularity of the applications (measured by the amount of requests) is used. This approach, unlike any previous work, leads to the prioritization of popular applications during the placement process, thus benefiting a larger number of final users.

The mechanisms are tested via simulation under different scenarios and conditions. The experiments show that it is possible to provide lower latency to popular applications while also reducing the latency of the overall system. Furthermore, for dynamic scenarios, a profiling scheme based on application popularity allowed to reduce not only the latency but also the jitter for popular applications.

Keywords: Fog; Latency; Service Orchestration; Service Placement; Popularity

Resumo

Estamos perante uma nova revolução industrial impulsionada por dados digitais, computação e automação. A atividade humana, processos industriais e investigação levam à recolha, geração e processamento de dados numa escala sem precedentes, fomentando novos produtos, serviços e aplicações. Entre as aplicações que geram mais tráfego, estão as aplicações relacionadas com realidade aumentada/virtual e transmissão de vídeo. Estas aplicações têm restrições temporais estritas para serem executadas de acordo com o esperado pelo utilizador final e, geralmente, recorrem ao uso de computação em Nuvem para alcançar elasticidade, serviços a pedido do utilizador, agrupamento de recursos e entrega atempada. No entanto, a nova geração de aplicações e serviços sensíveis a atraso tem requisitos que são apenas parcialmente cumpridos pelas soluções de computação em Nuvem existentes.

Recentemente, tem havido uma mudança de paradigma no sentido de trazer os serviços da Nuvem para a perímetro da rede. Nesta área periférica existe uma abundância de dispositivos heterogéneos com recursos limitados tanto na geração como no consumo de dados. Este cenário representa um aumento na quantidade de dados, que leva ao aumento do tráfego e do tempo de resposta nas comunicações de, e para, a Nuvem. No sentido de ajudar o pré-processamento destes dados e aliviar a carga enviada através da rede, é possível mudar os dispositivos de armazenamento e processamento para a extremidade da rede, reduzindo os tempos de resposta e beneficiando, particularmente, as aplicações sensíveis a atraso. Esta solução é conhecida como computação em Nevoeiro.

A computação em Nevoeiro é um paradigma importante que ajuda a satisfazer os requisitos que não são totalmente cobertos pela Nuvem; no entanto, o uso desta tecnologia cria novos desafios. O Nevoeiro necessita de suportar a orquestração de aplicações e serviços disponíveis a pedido do utilizador, com adaptabilidade, e garantindo uma performance flexível e cumpridora das restrições temporais. Na prática, as abordagens tradicionais de orquestração de serviços aplicadas aos serviços em Nuvem não são adequadas para a larga escala e dinamismo dos serviços de Nevoeiro. Assim, existe a necessidade de novos mecanismos para a coordenação de recursos, aplicações e serviços de Nevoeiro. Este trabalho propõe uma framework de orquestração inteligente baseada numa abordagem híbrida que combina orquestração centralizada e coreografia distribuída para lidar com a gestão de recursos e, ainda, uma arquitetura para um orquestrador de serviços que inclui uma representação dos seus diferentes módulos e respetivas interações.

Para lidar com as restrições temporais das aplicações sensíveis a atraso, o orquestrador deve otimizar onde colocar as aplicações e serviços. Um dos módulos da arquitetura do orquestrador de serviços proposto neste trabalho é encarregue da decisão da colocação final dos serviços da aplicação de modo inteligente e consciente de contexto com o objetivo de minimizar a latência. Este trabalho

apresenta três mecanismos para a colocação de serviços no ecossistema de Nevoeiro, nomeadamente, um baseado em programação linear inteira, um baseado em algoritmos genéticos e um baseado em partição de grafos, utilizando o algoritmo PageRank. Todos os mecanismos usam uma combinação de métricas que avalia as características tanto das aplicações como da infraestrutura da rede e que guia o processo de colocação. Da perspetiva da rede, é considerado o tempo de propagação; da perspetiva da aplicação, é utilizada a popularidade das aplicações, medida pela quantidade de pedidos. Esta abordagem, contrariamente aos trabalhos anteriores, leva à priorização das aplicações populares durante o processo de colocação e, consequentemente, beneficia um maior número de utilizadores finais.

Os mecanismos são testados via simulação com diferentes cenários e condições. Os resultados mostram que é possível obter uma latência menor para aplicações populares e, simultaneamente, reduzir a latência geral do sistema. Além disso, em cenários dinâmicos, um esquema de perfil baseado na popularidade das aplicações permitiu reduzir não só a latência, como também a variação de atraso das aplicações populares.

Palavras-chave: Nevoeiro; Latência; Orquestração de Serviços; Colocação de Serviços; Popularidade

Foreword

During this PhD program, the practical knowledge acquired working in academic projects, and the theoretical and deep concepts acquired during the research, contributed to the development of this thesis. The work detailed in this thesis was performed at the Laboratory of Communications and Telematics (LCT) of the Centre for Informatics and System of the University of Coimbra (CISUC), within the context of the following projects:

SusCity - Sustainable Cities; financed by the FCT - Foundation for Science and Technology within the scope of project (MITP TB/CS/0026/2013). The goal of this project was to design tools and services to promote the efficient use of urban resources within the scope of Smart Cities. This project allowed to identify use cases of services that require low latency levels, and the platforms and technologies to support those requirements.

SORTS - Supporting the Orchestration of Resilient and Trust-worthy Fog Services; financed by the CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES-FCT/8572/14-3) and by the FCT - Foundation for Science and Technology (FCT/13263/4/8/2015/S). The aim of this project was to design, implement, and develop a service orchestrator for Fog environments, capable of maintaining resilience, trustworthiness, and low-latency in a dynamic environment, such as the Fog. The work performed in this project was the identification of requirements, the design of the orchestrator architecture, and the definition of the service placement mechanisms for the *Optimizer* and *Planner* modules of the orchestrator.

This work was funded by the following grants:

- Project grant: FCT - Foundation for Science and Technology within the scope of project *Intelligent Computing in the Internet of Services - iCIS* (CENTRO-07-ST24-FEDER-002003), from October 2014 to June 2015.
- Project grant: FCT - Foundation for Science and Technology within the scope of project *DenseNet* (PTDC/EEI-SCR/6453/2014), from July 2016 to February 2017.
- PhD grant: FCT - Foundation for Science and Technology PhD grant (SFRH/BD/119392/2016), from March 2017 to April 2021.

The outcome of the design, experiments, and assessments of several mechanisms during the course of this thesis resulted in several publications, listed below.

Journal papers: For these articles, as main author, the tasks of idea conception, evaluation of the State-of-the-Art, and writing were executed. The particular contributions of each publications are:

- Proposal of a modular framework for smart orchestration:

  Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2017). Service Placement for Latency Reduction in the Internet of Things. *Annals of Telecommunications*, 72(1):105-115. Springer.

- Analysis of the State-of-the-Art for Fog orchestration and identification of challenges:

  Velasquez, K., Perez Abreu, D., Assis, M. R. M., Senna, C., Bittencourt, L. F., Laranjeiro, N., Curado, M., Vieira, M., Monteiro, E., and Madeira, E. (2018). Fog orchestration for the Internet of Everything: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications*, 9(14):1-23. Springer.

- Proposal of an additional Fog placement mechanism for latency reduction and a profiling scheme for applications based on their popularity:

  Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2021). Service Placement for Latency Reduction in the Fog via Application Profiling. Submitted for publication to *IEEE Access*, pages 1-15, IEEE.

Conference papers: For these papers, as main author, besides conceiving the idea and performing the revision of the State-of-the-Art, implementation and writing tasks were carried out. The contributions of each publications are as listed:

- Identification of use-cases and open issues:

  Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2015). Towards Latency Mitigation in Emergency Scenarios. In *1st Cloudification of the Internet of Things (CIoT)*, pages 1-4, Paris, France. IEEE.

- Proposal of an hybrid approach for Fog orchestration and a service orchestrator architecture:

  Velasquez, K., Perez Abreu, D., Gonçalves, D., Bittencourt, L. F., Curado, M., Monteiro, E., and Madeira, E. (2017). Service Orchestration in Fog Environments. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 329-336, Prague, Czech Republic. IEEE.

- Introduction of two Fog placement mechanisms for latency reduction based on application popularity:

  Velasquez, K., Perez Abreu, D., Paquete, L., Curado, M., and Monteiro, E. (2020). A Rank-based Mechanism for Service Placement in the Fog. In *2020 IFIP Networking Conference (Networking)*, pages 64-72, Paris, France. IEEE.

Cooperation papers: For these publications, as co-author, the tasks included participating in discussions and writing.

- Perez Abreu, D., **Velasquez, K.**, Curado, M., and Monteiro, E. (2015). Resilience in iot infrastructure for smart cities. In *2015 IEEE Cloudification of the Internet of Things (CIoT)*, pages 1–4, Paris, France. IEEE.
- Perez Abreu, D., **Velasquez, K.**, Curado, M., and Monteiro, E. (2017). A resilient Internet of Things architecture for smart cities, *Annals of Telecommunications*, 72(1):19–30. Springer.
- Perez Abreu, D., **Velasquez, K.**, Pinto, A. M., Curado, M., and Monteiro, E. (2015) Describing the Internet of Things with an ontology: The SusCity project case study. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 294-299, Paris, France. IEEE.
- Perez Abreu, D., **Velasquez, K.**, Curado, M., and Monteiro, E. (2017). An iot infrastructure for smart cities: The suscity project use-case. In *3rd Energy for Sustainability International Conference (EfS)*, pages 1-5, Madeira, Portugal. InderScience Publishers.
- Fernandes, J. T., Perez Abreu, D., **Velasquez, K.** , Monteiro, E., and Gomes Martins, A. (2017). An Architecture to Support Affordable Internet of Things Applications: The SusCity project Case Study. In *2017 CLME2017/VCEM 8º Congresso Luso-Moçambicano de Engenharia/V Congresso de Engenharia de Moçambique*, pages 4-8, Maputo, Moçambique. INEGI/FEUP.
- Fernandes, J. T., Perez Abreu, D., **Velasquez, K.** , Mateus, M. Dias, J. A., Carrilho, M. C., Monteiro, E, and Gomes Martins, A. (2017). Building a Smart City IoT Platform - The SusCity Approach. In *48nd Spanish Congress on Acoustics and the Iberian Encounter on Acoustics (TECNIACUSTICA)*, pages 557-566, A Coruña, Spain. 2017. Sociedad Española de Acústica (SEA).
- Perez Abreu, D., **Velasquez, K.**, Assis, M. R. M., Bittencourt, L.F., Curado, M., Monteiro, E., and Madeira, E. (2018). A Rank Scheduling Mechanism for Fog Environments. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 363-369, Barcelona, Spain. IEEE.
- Gonçalves, D., **Velasquez, K.**, Curado, M., Bittencourt, L. F., and Madeira, E. (2018). Proactive Virtual Machine Migration in Fog Environments, In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 742-745, Natal, Brazil. 2018, IEEE.
- Perez Abreu, D., **Velasquez, K.**, Curado, M., and Monteiro, E. (2020). A comparative analysis of simulators for the Cloud to Fog continuum. *Simulation Modelling Practice and Theory*, 101(1):102029. Elsevier.
- Perez Abreu, D., **Velasquez, K.**, Paquete, L., Curado, M., and Monteiro, E. (2020). Resilient Service Chains through Smart Replication. *IEEE*

Access, 8(1):187021-187036. IEEE.

Contents

List of Figures

List of Algorithms

List of Tables

Acronyms

ALTO Application-Layer Traffic Optimization

AS Autonomous Systems

CDN Content Delivery Networks

FANET Flying Ad-Hoc Network

FF First Fit

GA Genetic Algorithm

GW Gateway

ICT Information and Communication Technology

ILP Integer Linear Programming

IoT Internet of Things

IPDV Internet Protocol packet Delay Variation

IPTD Internet Protocol packet Transfer Delay

MANO Management and Orchestration

MIB Management Information Base

NFV Network Function Virtualization

OWD One Way Delay

P2P Peer-to-PeerP

PRP Popularity Ranked Placement

PD Propagation Delay

PID Provider-defined IDentifier

QoE Quality of Experience

QoS Quality of Service

RR Round Robin

RTT Round-Trip Time

SDN Software Defined Networking

SNMP Simple Network Management Protocol

UASN Underwater Acoustic Sensor Network

VANET Vehicular Ad-Hoc Network

VM Virtual Machine

WSGA Weighted Sum Genetic Algorithm

WSN Wireless Sensor Network

YAFS Yet Another Fog Simulator

Chapter 1

Introduction

Contents

As communications evolve to give space to new applications, such as augmented reality and virtual reality, and other delay-sensitive applications, new paradigms arise to provide essential characteristics like lower latency, mobility support, and location awareness. Such is the case of Fog computing, which extends from the well known Cloud computing paradigm by bringing processing, communications, and storage capabilities to the edge of the network. By offering these novel features, also new challenges emerge that call for the design and implementation of orchestration mechanisms to deal with resource management. One of these mechanisms is related to the service placement, which consists in the selection of the appropriate execution node for the applications according to a specific optimization objective. The work presented in this thesis aims at proposing service placement mechanisms to reduce the latency in Fog environments. The motivation, objectives, and contributions of this work are listed in this chapter, and finally an outline of this document is provided.

1.1 Motivation and Problem Statement

The development of resource demanding and delay-sensitive applications (e.g., emergency communications, augmented and virtual reality, video streaming, online gaming), and the proliferation of applications within the context of the Smart City and Internet of Things (IoT) paradigms lead to the search of resources that, until certain point, was covered by the use of the Cloud computing paradigm. The Cloud represents a logically centralized pool of resources that are exploited by resource-hungry applications. However, this solution does not satisfy the needs of every application.

The decentralization of the Cloud, by bringing the services and applications towards the IoT devices and near-user edge devices in order to provide lower latency, mobility support, and location awareness, led to the advances in newer paradigms such as Fog computing [Vaquero and Rodero-Merino, 2014]. The Fog is an extension of the Cloud that brings resources closer to the users, to the edge of the network, conceived to address applications and services that do not fit well in the Cloud paradigm. Applications that require characteristics such as low latency, geo-distribution, mobility, and large-scale distributed systems benefit from the Fog.

The Fog is located in the vicinity of the user, on the frontier with the IoT, where there is a plethora of heterogeneous devices that have to work harmoniously to keep the services running with an acceptable Quality of Service (QoS). This fact calls for the automation of management functions to deal with the complexity of the scenario. Hence, orchestration functions are required in order to automate the management in such a dense, heterogeneous, and complex environment.

Orchestration denotes a single centralized executable process that coordinates the interaction between different applications or services. Hence, orchestration uses a centralized approach to applications and services composition [Saraiva de

Sousa et al., 2019]. Even though different orchestration approaches have been applied to other scenarios such as the Cloud, the Fog has particular characteristics, including its distributed nature, the heterogeneity of its devices, and the constraints in their resources, that call for novel orchestration mechanisms.

It is necessary to perform a new revision on orchestration solutions to adapt them to the Fog, taking into consideration its characteristics. Furthermore, a review of decentralized choreography principles can be added to the solutions, in order to provide them dynamism and real-time response at the lower levels of the infrastructure. Choreography could be defined as a global description of the participating applications and services, which is specified by the exchange of information, rules of cooperation and agreements among two or more endpoints; thereby, this strategy uses a decentralized approach for applications and services composition [Leite et al., 2013; Bittencourt et al., 2018]. The service orchestrator must be capable of maintaining resilience, trustworthiness, and low latency in a dynamic environment, such as the Fog, and also guaranteeing acceptable levels of QoS, even with massive amounts of data being exchanged.

Regarding these massive amounts of data, about two thirds of the world population will have Internet access by 2023 [Cisco, 2020]. The number of devices and connections is growing faster than the population, increasing the IP traffic, which is expected to triple by 2022 [Cisco, 2019]. Of this traffic, 82% will correspond with IP video traffic, with an expected increase of virtual reality and augmented reality (about 12 times) and Internet video-to-TV traffic (about 3 times). With this extensive load of data, a rapid exhaustion of the infrastructure resources, and, therefore, a decrease in the quality of the communications is foreseeable, which represents another challenge for the orchestrator. Given the predominance of streaming video in the predictions of the IP traffic, latency becomes one key aspect to consider to meet delivery constraints of applications.

The service orchestrator is assigned with the task of managing the resources of the network efficiently, while offering low latency levels. Nevertheless, in more decentralized scenarios, like the Fog, more complex managing approaches are required. The Fog needs to support the orchestration of applications and services on demand, with adaptability, while providing flexible performance and low latency. To improve latency levels it is necessary to design and implement smart solutions, including a set of *Planning Mechanisms* that must be implemented, particularly those related to *Service Placement*. Only bringing the service instances to the edge of the network is not enough, since the perimeter of the Cloud can be broad, and in a dense environment such as the Fog, there could be multiple choices to place the service instances. Thus, it is relevant to select a metric that allows to guide the placement process. The metric (or set of metrics) should reflect not only the current state of the network, but also characteristics intrinsic to the applications themselves, to differentiate them according to their requirements.

The popularity of the applications (and ultimately, of their services) seems to be a good option since it would lead the placement of the most popular applications towards the locations where the most data and service-hungry users are

stationed, i.e., at the edge of the network. The popularity can be measured by the number of requests for a given application. On the other hand, popularity as a metric is not enough to guarantee the QoS requirements for the applications at the communication infrastructure level, since it does not take into consideration the current status of the network. Thus, combining the popularity of the applications with a network metric, such as the propagation delay, will lead to a context-aware orchestrator regarding the service placement process.

1.2 Objectives

The main goal of this work is to propose mechanisms to reduce the latency in Fog environments. The objectives can be listed as follows:

- Analyze the characteristics of the Fog, focusing on the aspects that differentiate it from the Cloud, and that require the development of novel management techniques. The analysis should also make emphasis on how the service orchestrator could make an impact on reducing the latency for final users;
- Design an orchestration architecture for the Fog, taking in consideration all the challenges newly imposed by the Fog. The solution must be multi-tiered, including notions of the entire Cloud-to-IoT continuum, setting tasks to handle all the challenges, but detailing the optimizing mechanisms regarding latency reduction;
- Conceive a mathematical model to find the optimal location of the services in order to minimize the latency for the final user;
- Propose a set of service placement mechanisms to minimize the response time of applications in Fog environments. The idea is to come up with different strategies to determine the optimal location for a service in a given scenario;
- Define a set of metrics to guide the service placement decision, and leverage the determination of where to place the services. The metrics should include aspects of the applications as well as the network infrastructure; and
- Evaluate the performance of the mathematical approach and the alternative service placement mechanisms, by means of simulation, under varying scenarios. The evaluation must include static and dynamic conditions.

While working on these objectives several contributions were produced, and are listed in the section below.

1.3 Contributions

Taking in consideration the objectives previously described, the following contributions were achieved:

- A hybrid approach for Fog orchestration, combining centralized orchestration for the upper levels closer to the Cloud, and a distributed choreography for the lower levels closer to the edge. This would allow to have a unified centralized vision of the entire system to apply general optimizations, while the choreography support for the lower levels will provide more dynamism to the proposed solution. This is presented in Chapter 3.
- A smart orchestration framework that follows the hybrid approach. The framework includes capabilities for the Service Repository, the Information Collection, and the Service Orchestrator. This is described in Chapter 3.
- A Service Orchestrator architecture, enclosed in the orchestration framework, including the depiction of its modules that handle the different orchestration strategies, including those related to service placement, but providing guidelines for other tasks such as security and resource management. This is presented in Chapter 3.
- A mathematical model based on Integer Linear Programmings aimed at maximizing the placement of popular applications while minimizing their latency. This is depicted in Chapter 4.
- A heuristic based on graph partition to create communities to place applications, that allows balancing the load among the nodes closer to the vicinity of the final user, to avoid overloading the Gateways. This is presented in Chapter 4.
- A heuristic based on a multi-objective Genetic Algorithm for application placement, which fitness function combines application popularity and network propagation delay with resource usage. The heuristic is described in Chapter 5.
- An application profiling scheme based on application popularity, measured by the number of requests. The profiles are defined according to how applications gain or lose popularity over time. This is introduced in Chapter 5.

The following chapters describe the process followed to achieve the objectives depicted in Section 1.2. The outline of this document is described in the next section.

1.4 Outline of the Thesis

This dissertation is distributed in 6 chapters, organized as follows. Chapter 2 presents the research context, including concepts on latency and Fog environments and its rising challenges, to later introduce a revision on the State-of-the-Art. Chapter 3 proposes a novel hybrid approach for orchestration in the Fog, including an architecture for the Fog orchestrator able to be adapted to different needs, such as latency reduction, mobility support, or increased resilience. Chapter 4 introduces two mechanisms for service placement to reduce latency, one based on Integer Linear Programming (ILP) and one heuristic based on

graph partition, particularly on the PageRank algorithm. Both mechanisms are evaluated via simulation using Yet Another Fog Simulator (YAFS). Chapter 5 presents a scheme to profile applications based on their changing popularity. The PageRank-based heuristic and a new mechanism based on a Genetic Algorithm (GA) are evaluated under dynamic conditions, applying the proposed profiling scheme, also using YAFS for the simulations. Finally, Chapter 6 shows a synthesis of the work, listing its contributions and offering future research paths that can be pursued to advance this work.

Chapter 2

Latency Reduction in Fog Environments

Contents

To fully grasp the problem of latency it is important to understand its basic concepts and the technologies designed to deal with it. This chapter describes the concept of latency in computer networks, its sources and metrics, to later introduce the paradigm of Fog Computing as an emerging technology to deal with latency in telecommunications. Finally, a revision of related work on reducing latency in Fog environments is discussed.

2.1 Latency in Communication Networks

Latency is the time it takes for a single bit to reach its destination from the time it was first requested [Briscoe et al., 2014]. The measured bit depends on the type of application in order to provide end-users a higher perception of responsiveness. The value of latency is obtained by the addition of different delays that affect the total time it takes for a bit to be delivered. Some of the components [Peterson and Davie, 2011] that affect the total delay of the network are depicted in Figure 2.1.

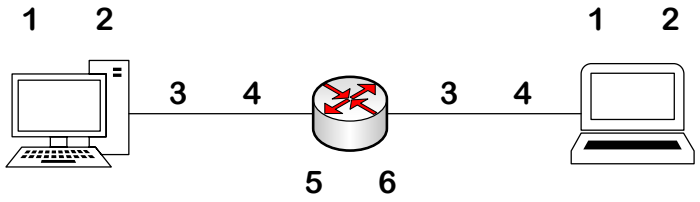


Figure 2.1: Latency Components

1. Coding: The time it takes to code the data, for instance, using a video codec;
2. End System Processing: The delay caused by the end host application and operating system, including the time employed by the communication protocols to perform their tasks;
3. Transmission: Refers to the time from when the first bit of a message is put in the link until the last bit is put on the link, determined by the data rate of the link;
4. Propagation: The time it takes for a bit to travel from one place to another through the transmission media;
5. Queuing: The time a packet stays in a queue in a networking element (e.g., switches and routers) before it gets processed; and
6. Node Processing: The time necessary for a packet to be handled by a network node, for instance, to determine the output interface according to the routing table of a router.

Many services and applications have severe restrictions on terms of latency in order to offer a good Quality of Experience (QoE) to final users. Such applications include emergency communications, eHealth, virtual/augmented reality, and on-line gaming, among others.

2.1.1 Sources

The sources of delay have been categorized in the following classes [Briscoe et al., 2014; Ford, 2014]:

- Structural delay: Depending on the structure of the network. Deals with service and content placement, optimizing routes, and name resolution among other factors. Corresponds with propagation and transmission delays;
- Interaction between endpoints: Results of the processing of activities between endpoints, such as protocol initialization, packet loss recovery, and security context initialization. Related to end system processing and transmission delays;
- Intra-end host delays: The latency resulting from processing in the applications and operating system in the end hosts. Corresponds with end system processing time and coding delay;
- Delays along the transmission paths: Composed by the sum of delays caused by different factors, for instance signal propagation, medium acquisition, serialization, link error recovery, switching/forwarding, and queuing. Related with transmission and queuing delays; and
- Delays related to link capacities: Originated by different causes, like limitations of the link capacity, sending redundant information, the underutilized capacity of the links, and the collateral damage, which is a delay generated by external flows affecting another. Also linked to the transmission and queuing delays.

This classification can help in identifying specific techniques that can work towards the mitigation of latency in different scenarios.

2.1.2 Metrics

When talking about latency, the metric used is a time unit (i.e., seconds), however it is important to clarify what is the time to measure, and the process to achieve such measurement. There does not seem to be a unique means to accomplish this, on the contrary, a combination of different measurements over different types of messages and conditions is commonly used.

Two of the most frequently used latency metrics are the Round-Trip Time (RTT) and the One Way Delay (OWD), both measurements of end-to-end delay [Wang et al., 2004]. The RTT measures the time it takes for a packet to reach its destination and come back to its source, while the OWD only accounts the time it takes to get to the destination. It is important to notice that the RTT does

not represent twice the time of the OWD, since the reply might travel through a different path or face different network conditions. Another common way to measure delay is to quantify the download completion time (for download content applications) or the service time, that is the time it takes to complete the service, measured from the initial request [Dukkipati and McKeown, 2006; Sadfi et al., 2005].

Additionally, other gauges commonly used are the Internet Protocol packet Transfer Delay (IPTD) and the Internet Protocol packet Delay Variation (IPDV) [ITU-T, 2013]. The IPTD is similar to the OWD and measures the one-way time interval from the moment the first bit of the packet traverses the entry point of the network until the moment the last bit crosses the exit point at the destination. The IPTD is usually just called *delay* or *latency*. The IPDV is the difference between the IPTD of a given packet and the reference IPTD of a population of interest packets. The IPDV is also referred as *jitter*. Both IPTD and IPDV are commonly reported in milliseconds.

Network latency has improved far more slowly than other performance metrics for commodity computers [Rumble et al., 2011]. The industry used to place the expected response time in the order of 50ms [Siltanen, 2013]. However, newer technologies such as 5G are stretching this bar to the order of 1ms for some applications [Tong and Zhu, 2014; Siddiqi et al., 2019]. With margins this low, it is clear the importance that is being given to maintaining low levels of latency for some applications.

2.2 Fog Environments

On the topic of reducing latency, some new technologies have emerged. This is the case of *Fog Computing*. The Fog is an environment with a plethora of heterogeneous devices that work in a ubiquitous and decentralized manner, communicating and cooperating among themselves [Vaquero and Rodero-Merino, 2014]. Thus, the Fog emerges as an extension of the Cloud paradigm escalating from the core of the network towards its edge; it is comprised of a heavily virtualized platform able to perform storage, processing, and networking activities between the Cloud servers and the end devices [Yi et al., 2015a,b]. The Fog comes to support novel applications and services that are not completely fit for the Cloud, granting them ubiquity, high resilience, low latency, decentralized management, and cooperation [Chiang and Zhang, 2016; Hung et al., 2015].

The OpenFog consortium [OpenFog Consortium, 2017] defines Fog as:

> *"A horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum".*

Figure 2.2 describes the Fog landscape. At the top of the picture is the Cloud level, that comprises centralized datacenters and offers powerful computational and storage resources. At the edge of the Cloud is the Fog level, that provides more limited storage and computational resources in a more hetero-

geneous and mostly wireless connected environment, that allows user access to the Information and Communication Technology (ICT) and its services. The bottom level corresponds to the IoT that is composed of sensors and actuators enabling the data collection for applications such as smart homes, smart buildings, eHealth and smart transportation.

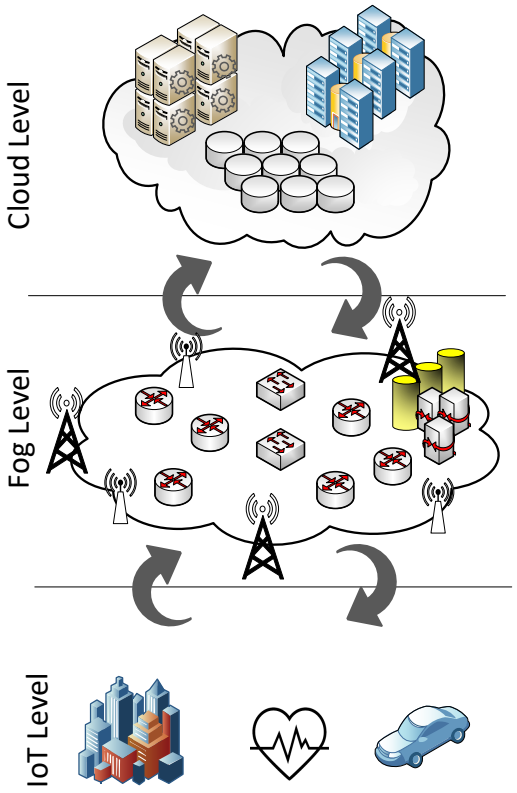


Figure 2.2: Fog Environment

The scenario at the edge of the Cloud is vastly different from the one at its core. Thus, to fully understand its peculiarities, the next subsection reviews the main aspects where Cloud and Fog diverge.

2.2.1 Transitioning from Cloud to Fog

The Fog introduced a paradigm shift from the traditional concept where the core of the network is in charge of providing information that will be consumed at its edge. To address the challenges arising from the transition from Cloud to Fog computing, the latter has to fulfill the following key features [Dastjerdi and Buyya, 2016; Vaquero and Rodero-Merino, 2014; OpenFog Consortium, 2017; Cisco, 2015]:

- Heterogeneity and interoperability, to deal with a broad diversity of physical and virtualized devices deployed in wide-ranging environments;

- Edge location, location awareness, and low latency, to guarantee that the most delay-sensitive data is processed closest to those requesting it;
- Wireless communication, to reach a variety of devices at the edge avoiding the installation of a fixed communication network and contributing to reduce the amount of traffic in the core network;
- Real-time support, to satisfy services and applications with time-sensitive requirements; and
- Mobility support, to allow continuity in the services provided to devices and final users.

In the Fog, final users (e.g., mobile devices and IoT sensors) generate ample quantities of data at the edge of the network, making them producers and consumers at the same time. The treatment of this unprecedented quantity of data represents a challenge for traditional paradigms, like Grid and Cloud computing; thus the Fog arises to overcome these limitations [Dastjerdi and Buyya, 2016; Cisco, 2016].

Although Cloud and Fog computing share overlapping features, the Fog becomes a non-trivial extension of the Cloud that must deal with characteristics inherent to its placement in the overall network infrastructure, such as location awareness, geographical distribution, low latency, real-time, and mobility support [Yi et al., 2015a; Bonomi et al., 2014, 2012; Botta et al., 2016]. Another significant difference regarding the Cloud is that the Fog encompasses a huge set of heterogeneous devices, mostly wirelessly connected, that are more constrained in terms of resources in comparison with the servers that reside in the Cloud [Varshney and Simmhan, 2017; Jiang et al., 2018].

Fog computing provides to the Cloud an alternative to manipulate exabytes of data generated daily from the IoT [Cisco, 2016]. With the ability to process the data near where it is generated and required, it is possible to tackle the challenges regarding data volume, interoperability, and time-sensitivity. By eliminating the RTT related to traveling to the Cloud and back, it is feasible to accelerate the awareness and response time of services and applications. Furthermore, by preventing the need to send the data to the Cloud, or at least by aggregating it first, capacity of the communication channel is saved by reducing the amount of traffic in the core network.

2.2.2 Application Scenarios

The combination of the Fog and IoT paradigms could be used in many scenarios to achieve and improve application and service requirements. In this subsection, some examples of the collaborative use of Fog and IoT are described within four specific areas: *Automation*, *eHealth*, *Smart Cities*, and *Infotainment*.

Automation
Automation systems refer to the integration of cyber technologies that make devices Internet-enabled to implement services for different industrial tasks such

as Internet-based diagnostics, maintenance, and efficient and cost-effective operation [Jazdi, 2014].

Constraints of this scenario include data and service security, scalability, and minimizing latency and jitter. Moreover, operator infrastructures can contain devices from multiple manufacturers communicating with different technologies, sometimes proprietary, creating additional problems regarding interoperability and service deployment that must rely on manual intervention by network managers [Rotsos et al., 2016].

eHealth

Another scenario of applicability of Fog computing is eHealth. The processing of data generated by wearable devices ([Fitbit, Inc., 2020; Apple Technology Company, 2020]) requires low latency among other requirements related to the speed of events, such as interoperability, scalability, and security. Fog computing brings processing to the edge of the network, significantly improving technical factors that allow shorter responses in emergency scenarios. Fog can also interface with other paradigms such as Cloud computing, allowing persistent data to be driven by the Clouds for permanent storage and performing tasks such as analytics.

Smart Cities

The Smart City paradigm emerged to describe the use of new technologies in everyday urban life, providing the management of its services (e.g., energy, transportation, lighting, public safety) using ICT. These technologies implement a logical/virtual infrastructure to control and orchestrate physical objects to accommodate the city services to the citizen needs [Borgia, 2014].

In the context of Smart Cities, mobility is a key requirement that should be explored allowing devices and services to capture information about the environment and act in real-time. The Orchestrator must be able to maintain low latency, high resilience, and trustworthiness according to the applications and users expectations, even in times when the infrastructure is stressed under heavy loads of traffic.

Infotainment

The term *Infotainment* refers to a combination of Information and Entertainment served together [Saini et al., 2017], and is used here to group the applications like virtual reality applications, gaming applications, video streaming, financial trading applications among others, which require very low latency levels (i.e. around tens of milliseconds) [Chiang and Zhang, 2016].

These applications generate and consume heavy traffic loads and have rigorous demands regarding latency. To efficiently handle these services, new management functions at Fog level must be conceived to deal with the scenario requirements efficiently; for flexible connectivity in heterogeneous and highly mobile environments, with strong latency guarantees, network operators require

innovative orchestration mechanisms that support dynamic multi-technology resource management [Rotsos et al., 2017].

Once the features of the scenario are defined, it is important to realize the set of challenges they impose, and that have to be orchestrated. A discussion on the challenges that a Fog orchestrator must overcome is presented in the following subsection.

2.2.3 Research Challenges in Fog Orchestration

As in the Cloud, there is a need to manage the resources in the Fog. Many solutions have already been developed with this purpose for the Cloud; but even though the Cloud and the Fog use the same kind of resources (storage, network, and processing) and share the need for many of the same mechanisms (e.g., virtualization), the same procedures can not be migrated from one environment to the other, given that there are fundamental differences among them [Bonomi et al., 2014; Hao et al., 2017].

Unlike the Cloud, which is centralized, the Fog is aimed at services and applications with distributed deployment [Aazam and Huh, 2014]. The management of the devices in the micro datacenters between the Cloud and the underlying IoT is another requirement for the Fog [Sheng et al., 2015; Aazam and Huh, 2015b]. Fog devices are placed between smart devices and the Cloud; in this area, the homogeneity of resources that can be found in the Cloud disappears to give room for a more heterogeneous environment, where the characteristics and manufacturers of the resources vary [Stojmenovic, 2014].

Furthermore, the devices in the Fog are resource-constrained in comparison with the resource-rich Cloud counterpart [Luan et al., 2016]; energy, processing, and storage resources are limited in the Fog. The Fog also deals with mobility, where mobile and wireless nodes are commonly less reliable in their connectivity behavior [Aazam and Huh, 2015a]; this can cause drops in the QoE or even interruption of the services provided. Security also has to be analyzed from a new perspective. Fog devices face threats that are not present in more controlled Cloud scenarios, regarding both security and privacy [Stojmenovic and Wen, 2014]. Additional challenges have been identified [Wen et al., 2017] and include dealing with a larger scale of devices and a highly dynamic environment.

A Fog Orchestrator must be able to maintain low latency, high resilience, and trustworthiness according to the applications and users expectations, even in times when the infrastructure is stressed under heavy loads of traffic. The key issue here is that such loads are intrinsically specific to a particular application and cannot be reutilized from other scenarios or domains; thus further research in this direction to achieve the requirements mentioned above is required.

2.2.4 Latency in Fog Environments

One of the characteristics of Fog environments is that they have the potential to provide low levels of latency [Vaquero and Rodero-Merino, 2014; OpenFog Consortium, 2017]. This allows the deployment of a different kind of services with real-time and low latency restrictions that are not necessarily fit for the Cloud; but also requires a new set of mechanisms that guarantee that these low latency levels are met.

Smart routing and forwarding mechanisms should be designed, aiming at faster response time. A multipath approach [Chen et al., 2013; Pu, 2018] could be employed to achieve this goal, especially when dealing with huge bulks of data; however, for small but critical tasks, the use of redundant packets has proven to be efficient [Vulimiri et al., 2012, 2013].

Another possibility is designing intelligent service placement mechanisms [Moens et al., 2014; Xiong et al., 2016] for the Orchestrator. It is also important to take into consideration the mobile nature of the devices (e.g., sensors in cars), for which location awareness [Steiner et al., 2012; Guerrero et al., 2019a] and dynamism support [Zhang et al., 2012; Wang et al., 2017] must also be included.

Given that the tendency is shifting time-constrained services and applications towards the edge of the Cloud, into the Fog, it is imperative to guarantee that the time restrictions are met [Dolui and Datta, 2017]. The service orchestrator must incorporate novel mechanisms, different from those already available for other distributed systems such as Cloud, that are sensitive to time constraints, and that support other features such as dynamism and geo-distribution [Yi et al., 2015a].

Another issue to take in consideration is the more limited resources regarding the bandwidth of the links in comparison with Cloud systems, given their wireless nature and narrower capacity [Osanaiye et al., 2017].

These challenges call for the make-up and development of new orchestration mechanisms for the Fog to handle the strict latency constraints of services and applications. These mechanisms are to be executed under the control of a well-designed orchestrator that takes advantages of the characteristics inherent to the Fog. Some of the approaches previously explored to handle Fog orchestration, and particularly latency, are introduced in the following section.

2.3 Orchestrating the Fog to Reduce Latency

The Fog requires a well-constructed orchestrator able to deal with the full management function for this complex environment, and properly handle all the challenges previously described. Several efforts have been carried out in the field of Fog orchestration, and particularly aimed at latency reduction, and the most relevant are outlined below.

2.3.1 Orchestration

The concept of orchestration frequently comes around when discussing service oriented architectures, virtualization, and management of resources in network infrastructures. The need for automated procedures to deal with different topics has been explored before in diverse contexts, such as Cloud computing [ETSI GS NFV-MAN, 2014; OASIS Standards Group, 2013]. More recent works are focused on orchestration in Fog and edge environments, and a collection of them is presented in this subsection.

Some efforts have been carried out by standardization organizations to define ground rules to orchestrate the perimeter, or edge, of the network. ETSI [ETSI GS MEC, 2016] defines a framework that includes entities at the system level, at the host level, and at the network level. A reference architecture for orchestration is also described; its core component, the orchestrator, has a general view of the entire system, while on the other hand, the hosts provide a virtualization infrastructure to run applications. The OpenFog Consortium also proposes an architecture for Fog Computing and its orchestration [OpenFog Consortium, 2017]. Their proposal describes *pillars* as key attributes needed for a system to provide the distribution of computing, storage, and communication functions near the final users. Orchestration is defined as a transversal function among the pillars or the Fog architecture, including tasks related to security, manageability, performance, and scaling.

The OASIS Standards Group describes Tosca [OASIS Standards Group, 2013], a topology and orchestration specification for Cloud applications. Their main focus is the enhancing of portability and operational management of Cloud applications that should be taken in consideration during the service orchestration process. Santoro et al. [Santoro et al., 2017] propose an orchestration platform to manage applications workload and to control resource usage in Fog environments. Their proposal is based on open source technologies and is evaluated in a testbed with different use cases, including face detection and vehicle tracking. Hoque et al. [Hoque et al., 2017] analyze different container orchestration tools (e.g., Kubernetes, Docker Swarm, Apache Mesos-Marathon) with special emphasis on their Fog requirements support to ultimately propose a container orchestration framework for the Fog. de Brito et al. [de Brito et al., 2017] propose an architecture for service orchestration for the Fog, including a prototype implemented as proof-of-concept. They define the orchestrator and its components, as well as the interaction among them.

A distributed computing architecture and a container orchestrator is presented by Taherizadeh et al. [Taherizadeh et al., 2018]. The architecture allows the offloading of applications from the edge of the network to other nodes in the Fog. The architecture also provides mobility support for when the user moves from one geographical location to another. Kim et al. [Kim et al., 2018] offer a prototype for network management using container technology. Their proposal covers security, scalability, and performance tasks for Cloud/Fog environments. Alam et al. [Alam et al., 2018] introduce a modular framework that provides orchestration via the use of containers (i.e., Docker) to simplify management

while enabling distributed deployment of applications.

Particular mechanisms to handle orchestration tasks have also been proposed. Santos et al. [Santos et al., 2017] describe a framework for management of Fog functionalities in 5G-enabled Smart Cities. The orchestrator is accompanied by a Peer-to-PeerP (P2P) Fog protocol based on Open Shortest Path First (OSPF) for the exchange of information between the Fog nodes. Viejo and Sanchez [Viejo and Sanchez, 2019] propose security protocols for orchestration and delivery of IoT services in the Fog. The procedures include setup and user registration, secure orchestration, and secure delivery. For the secure orchestration procedure, a subset of Fog nodes, following a tree hierarchy, is selected to run a specific procedure, receiving cryptographic material to be used during the service delivery. Santos et al. [Santos et al., 2020] present an orchestration mechanism aimed at the selection of Fog nodes for video content download. The mechanism receives user feedback to assess the video streaming from the Fog nodes.

Table 2.1 summarizes the works presented in this subsection. Column *Architecture* indicates the presence of an architecture proposal for the orchestrator; column *Approach* indicates if the solution is centralized or distributed; column *Environment* states if the solution is focused only in Cloud, only in Fog, or in the Cloud-to-Fog continuum (Cloud/Fog); finally, column *Implementation* gives information on the implementation details provided in the proposal.

From Table 2.1 three main categories can be identified: (1) works on providing guidelines for the orchestration process [ETSI GS MEC, 2016; OpenFog Consortium, 2017; OASIS Standards Group, 2013]; (2) works that propose a configuration framework of different technologies and tools that combined would help in the orchestration [Santoro et al., 2017; Hoque et al., 2017; de Brito et al., 2017; Taherizadeh et al., 2018; Kim et al., 2018; Alam et al., 2018]; and (3) works that are focused on proposing a particular mechanism that helps in a given task of the orchestration process [Santos et al., 2017; Viejo and Sanchez, 2019; Santos et al., 2020].

The works in the first category attempt to define the tasks that should be accomplished by an orchestrator, although do not present specific proposals on how to handle those tasks. Works on the second category are more focused on the proposal of different software tools, such as container orchestration tools (e.g., Kubernetes, Docker Swarm, Apache Mesos-Marathon) to manage, for instance, application workloads and to control resource usage in Fog environments [Santoro et al., 2017]. For the works on the third category, they focus on a single challenge to be handled by the orchestrator (e.g., security [Viejo and Sanchez, 2019], QoE [Santos et al., 2020]) and propose different mechanisms to handle said issues.

The definition of an architecture for a Fog orchestrator and its constituent modules has been addressed [ETSI GS MEC, 2016; OpenFog Consortium, 2017; de Brito et al., 2017; Taherizadeh et al., 2018; Santos et al., 2017]. It is also noticeable that although many works are solely focused on the Fog [ETSI GS NFV-MAN, 2014; Santoro et al., 2017; Hoque et al., 2017; de Brito et al., 2017; Kim et al., 2018], more recent works are moving towards a transversal orchestration

Table 2.1: Characteristics from Related Work on Orchestration

Work	**Architecture**	**Approach**	**Environment**	**Implementation**
ETSI GS MEC [2016]	Yes	Centralized	Fog	Superficial guidelines
OpenFog Consortium [2017]	No	Centralized	Cloud/Fog	Superficial guidelines
OASIS Standards Group [2013]	Yes	Centralized	Cloud	Superficial guidelines
Santoro et al. [2017]	No	Centralized	Fog	Open source technologies
Hoque et al. [2017]	No	Centralized	Fog	Configuration framework
de Brito et al. [2017]	Yes	Centralized	Fog	Configuration framework
Taherizadeh et al. [2018]	Yes	Centralized	Cloud/Fog	Configuration framework
Kim et al. [2018]	No	Distributed	Cloud/Fog	Configuration framework
Alam et al. [2018]	No	Centralized	Cloud/Fog	Configuration framework
Santos et al. [2017]	Yes	Distributed	Cloud/Fog	P2P protocol exchange info
Viejo and Sanchez [2019]	No	Centralized	Cloud/Fog	Security protocol
Santos et al. [2020]	No	Centralized	Cloud/Fog	Node selection video download

across the Cloud-to-Fog continuum [OpenFog Consortium, 2017; Taherizadeh et al., 2018; Alam et al., 2018; Santos et al., 2017; Viejo and Sanchez, 2019; Santos et al., 2020]. The works are also almost entirely focused on centralized approaches, with some exceptions using a distributed approach [Kim et al., 2018; Santos et al., 2017].

Among the tasks performed by an orchestrator aimed at reducing the latency, as seen in Section 2.2.4, are smart routing and smart service placement. The most relevant works on these topics are discussed in the following subsections, and the open issues found in the analysis of the related work are discussed in Subsection 2.3.4.

2.3.2 Smart Routing

Finding the best route could render in better response times. The particularities imposed by the Fog, and detailed in Section 2.2.3, call for a revisit of existing routing mechanisms to adapt them to the newly established Fog requirements.

Vulimiri et al. [Vulimiri et al., 2012] argue that the use of redundancy is effect-

ive to reduce latency, particularly for small but critical tasks. They propose initiating redundant operations and using the first result which completes. The idea is further studied [Vulimiri et al., 2013], characterizing the situations in which redundancy can help reducing latency. Okay and Ozdemir [Okay and Ozdemir, 2018] present a survey on routing for Fog platforms to later introduce a hierarchical Software Defined Networking (SDN)-based routing architecture in which Fog controllers manage local frequent events while a Cloud controller takes global actions for rare events. Kadhim and Seno [Kadhim and Seno, 2019] propose a multicast protocol based on SDN for Vehicular Ad-Hoc Networks (VANETs). The protocol includes deadline and bandwidth constraints through a priority based algorithm to schedule multicast requests based on application types. OMNETT++ is used for validation.

Lu et al. [Lu et al., 2018] introduce a position-based routing scheme for inter-vehicle communication in VANETs. They use a greedy forwarding strategy to send data packets between two junctions, the main goal is to improve packet rate and end-to-end delay. Authors used ns-2 for the experimental evaluation. Ullah et al. [Ullah et al., 2020] present a taxonomy of position based routing protocols for VANETs; furthermore, they propose an architecture that supports position based routing by using road junctions and Fog nodes for path selection, to improve transmission time and communication costs. Manasrah et al. [Manasrah et al., 2019] propose an optimized service broker routing policy to minimize response time and costs. Their approach is using Genetic Algorithms to select the optimal datacenter for each user based on the response time and overall cost. CloudAnalyst was used in the evaluation. Borujeni et al. [Borujeni et al., 2018] present two Fog-based algorithms for Wireless Sensor Network (WSN), to find optimal cluster heads according to the remaining energy of the Fog nodes. Evaluation was carried out via simulations, using CloudAnalyst.

Qiuli et al. [Qiuli et al., 2019] propose a routing protocol for Underwater Acoustic Sensor Network (UASN) to reduce packet loss rate and transmission delay. Random selection and hotspot avoidance mechanisms are included to help solve the problems of hotspot and load unbalance. OMNET++ was used for the validation. Pu [Pu, 2018] presents a jamming-resilient multipath routing protocol so that intentional jamming or other failures do not interrupt the performance of the network. The protocol relies on link quality, traffic load, and spatial distance to achieve its goal. Simulations were used for evaluation purposes, using OMNET++ as simulation tool. Wang et al. [Wang et al., 2020] propose a Fog architecture accompanied by a routing algorithm for WSNs sensors that takes into consideration hop count and energy consumption. The routing layer of their Fog architecture is responsible for each sensors final path according to the scheduling method used in the Fog. Matlab was used for the evalauation. Naranjo et al. [Naranjo et al., 2017] suggest a routing protocol that prolongs the stable period of Fog sensor networks by maintaining balanced energy consumption. The protocol organizes the nodes and selects the cluster head in the WSNs. The evaluation was carried out using Matlab.

Table 2.2 summarizes the works on smart routing presented in this subsection.

Table 2.2: Characteristics from Related Work on Smart Routing

Work	**Network technology**	**Optimization factor**	**Environment**	**Evaluation**
Vulimiri et al. [2012]	*General*	Latency	*Unspecified*	*Unspecified*
Vulimiri et al. [2013]	*General*	Latency	*Unspecified*	ns-3
Okay and Ozdemir [2018]	SDN	Routing delay	Cloud/Fog	ONE
Kadhim and Seno [2019]	SDN/VANET	Delay	Fog	OMNET++
Lu et al. [2018]	VANET	Delay	Fog	ns-2
Ullah et al. [2020]	VANET	Packet delivery ratio	Fog	*Unspecified*
Manasrah et al. [2019]	*General*	Response time Cost	Cloud/Fog	CloudAnalyst
Borujeni et al. [2018]	WSN	Energy consumption	Cloud/Fog	CloudAnalyst
Qiuli et al. [2019]	UASN	Reliability	Fog	OMNET++
Pu [2018]	FANET	Latency	*Unspecified*	OMNET++
Wang et al. [2020]	WSN	Throughput Delay	Cloud/Fog	Matlab
Naranjo et al. [2017]	WSN	Energy consumption	Fog	Matlab

Former works were focused on redundancy [Vulimiri et al., 2012, 2013], sending the same duplicated data through different paths, which can lead to excessive traffic load. The Fog has already been considered as a main scenario for routing research [Kadhim and Seno, 2019; Lu et al., 2018; Ullah et al., 2020; Qiuli et al., 2019; Naranjo et al., 2017], although the Cloud-to-Fog continuum has also been studied [Okay and Ozdemir, 2018; Manasrah et al., 2019; Borujeni et al., 2018; Wang et al., 2020].

One of the main characteristics of the Fog is its mobility support (see Subsection 2.2.1). Many works are aimed at VANETs [Kadhim and Seno, 2019; Lu et al., 2018; Ullah et al., 2020], and Flying Ad-Hoc Networks (FANETs) [Pu, 2018], that handle high mobility, for which routing paths must be frequently updated. WSNs [Borujeni et al., 2018; Wang et al., 2020; Naranjo et al., 2017] and UASNs [Qiuli et al., 2019], are network technologies that are unstable regarding their topology, thus potentially requiring to modify the routing paths on each activation/deactivation of the sensor nodes.

Some works use another Fog characteristic (*location awareness*) to apply geographical location for position-based routing [Lu et al., 2018; Ullah et al., 2020], while others use basic networking principles such as multipath transmission [Pu, 2018; Kadhim and Seno, 2019] and clustering [Naranjo et al., 2017].

Regarding the optimization factor used in the analyzed works, the main focus is to maintain the routing paths under changing topology conditions [Ullah et al., 2020; Borujeni et al., 2018; Qiuli et al., 2019], but also to optimize other network

metrics such as energy consumption [Naranjo et al., 2017; Borujeni et al., 2018], reliability [Qiuli et al., 2019], network load [Manasrah et al., 2019] and delay [Lu et al., 2018; Wang et al., 2020; Kadhim and Seno, 2019; Okay and Ozdemir, 2018; Vulimiri et al., 2012, 2013].

The simulation tools used in most of the works in this selection are simulators for general networking purposes, such as ns-2 [Lu et al., 2018], ns-3 [Vulimiri et al., 2013], OMNET++ [Kadhim and Seno, 2019; Qiuli et al., 2019; Pu, 2018], and ONE [Okay and Ozdemir, 2018]; there is no Fog-based simulation tool used for these works (only a couple of works using a Cloud-based simulator [Manasrah et al., 2019; Borujeni et al., 2018]). Matlab was also used as evaluation tool [Wang et al., 2020; Naranjo et al., 2017]. Open research issues identified in this group of works are presented in Subsection 2.3.4.

2.3.3 Service Placement

Although the placement problem has been vastly addressed in the past for Cloud environments [Moens et al., 2014; Espling et al., 2016; Ghaznavi et al., 2015; Steiner et al., 2012], new strategies are required for the Cloud/Fog continuum [Souza et al., 2018].

Skarlat et al. [Skarlat et al., 2017b] analyze the placement of IoT services in the Fog, according to their QoS requirements. They define an ILP model aimed at maximizing the utilization of the Fog landscape, while also keeping the resource usage constraints. Evaluation is carried out using iFogSim [Gupta et al., 2017]. On another work, Skarlat et al. [Skarlat et al., 2017a] present a heuristic based on a GA to solve the service placement problem. The main purpose of this work is to maximize the number of services placed, and the usage of Fog devices. Once again, iFogSim is used as an evaluation tool. The proposed solutions are compared with a greedy First Fit (FF) approach. The GA provided lower deployment delay by exploiting more Cloud resources. Wang et al. [Wang et al., 2017] introduce an optimization approach and a heuristic for online scenarios. Both approaches are evaluated using simulations. Their proposal is $O(1)$ competitive for a broad family of cost functions, meaning its competitive ratio is given by a constant.

Taneja and Davy [Taneja and Davy, 2017] describe a Module Mapping algorithm aimed at the optimization of resource utilization for Cloud/Fog scenarios. The solution is compared with placing the applications entirely in the Cloud, and the validation is carried out by simulation using iFogSim. Response times and network usage are reduced when using the Fog compared with a Cloud-only approach. Lera et al. [Lera et al., 2019a] present a solution for service placement aimed at improving the availability by placing as many interrelated services as possible within the proximity of the user. The proposal is compared with an ILP approach by means of simulation using YAFS [Lera et al., 2019b], and showed improved QoS and availability. Guerrero et al. [Guerrero et al., 2019a] propose to place popular services closer to the users (according to the hop-count). The decision is made in a decentralized fashion by each of the devices. Simulation (using iFogSim) showed that more popular applications had lower latency while

less popular applications were affected by a larger delay.

Mahmud et al. [Mahmud et al., 2019] use fuzzy logic to place applications in the Fog, with the objective of maximizing the QoE. Experimental evaluation is performed using iFogSim. Results show a reduced deployment time and improved QoE. Brogi and Forti [Brogi and Forti, 2017] propose a model for the deployment of IoT applications in the Fog. Authors also introduce a tool, FogTorch, in which the model is prototyped. Guerrero et al. [Guerrero et al., 2019b] compare three evolutionary algorithms to optimize latency, service spread, and resource usage. The algorithms are tested using Python. Martin et al. [Paul Martin et al., 2020] combine service reliability and monetary cost in a multi-objective optimization for service placement in the Fog. They use a multiobjective optimization algorithm to minimize cost while at the same time maximizing reliability. Validation is carried out via simulations, using iFogSim, and also with a Fog testbed.

Deng et al. [Deng et al., 2016] formulate a workload allocation problem to optimize power consumption considering constrained service delay. Their proposal is based on sacrificing modest computational resources to save bandwidth and reduce delay. The evaluation is performed via simulation using Matlab. Venticinque and Amato [Venticinque and Amato, 2019] present a methodology to address the service placement problem. They use a case study based on an IoT application in the smart energy domain, for which they extended a software platform developed and released open source as part of the CoSSMic European project [SINTEF, 2013]. Their objective is to optimize the Fog landscape utilization while satisfying QoS requirements.

Liu et al. [Liu et al., 2018] use queueing theory for multi-objective optimization for offloading in Fog environments. Their objectives are minimizing energy consumption, execution delay, and cost. A mathematical model is also provided to complement the proposed algorithm. He et al. [He et al., 2018] present resource allocation schemes, jointly with admission control and offloading mechanisms, to support data analytics while maximizing service utilities. For the allocation, their solution decides how many and which nodes in the Fog would execute jobs. Authors also provide a simulator in which they evaluated their proposal.

The works reviewed in this section are summarized in Table 2.3. The proposals are mostly based on Fog environments, with some exceptions considering the Cloud-to-Fog continuum [Taneja and Davy, 2017; Deng et al., 2016].

An important issue to consider is the *decision factor*, which refers to the metric guiding the placement process. Most metrics used are based on network aspects such as resource usage [Skarlat et al., 2017b,a; Taneja and Davy, 2017; Guerrero et al., 2019b], power consumption [Deng et al., 2016; Liu et al., 2018], availability [Lera et al., 2019a], and latency or some sort of time-related metric like response time [Guerrero et al., 2019a,b; Deng et al., 2016; Venticinque and Amato, 2019; Liu et al., 2018; He et al., 2018]. In the case of metrics that concern the applications and their services, the only aspect could be the QoE, that measures the quality as experienced by the user, but no element that describes the applications and their services is used during the placement process.

Table 2.3: Characteristics from Related Work on Service Placement

Work	**Decision Factor**	**Approach**	**Environment**	**Evaluation**
Skarlat et al. [2017b]	Resource usage	ILP	Fog	iFogSim
Skarlat et al. [2017a]	Resource usage	ILP + GA	Fog	iFogSim
Wang et al. [2017]	Cost	LP + Heuristic	Mobile micro-clouds	*Unspecified simulator*
Taneja and Davy [2017]	Resource usage	Module mapping algorithm	Cloud/Fog	iFogSim
Lera et al. [2019a]	Availability	2-phase placement + communities	Fog	YAFS
Guerrero et al. [2019a]	Latency	Decentralized placement	Fog	iFogSim
Mahmud et al. [2019]	QoE	Fuzzy logic	Fog	iFogSim
Brogi and Forti [2017]	QoS	Mathematical	Fog	FogTorch
Guerrero et al. [2019b]	Latency, service spread, resource usage	GA	Fog	Python
Paul Martin et al. [2020]	Cost, reliability	Heuristic	Fog	iFogSim Testbed
Deng et al. [2016]	Delay, power consumption	Mathematical model	Cloud/Fog	Matlab
Venticinque and Amato [2019]	Response time, resource usage	BET method	Fog	Testbed
Liu et al. [2018]	Energy consumption, delay, cost	Mathematical model	Fog	*Unspecified simulator*
He et al. [2018]	Computational, cost, latency, bandwidth	Benchmark	Fog	*Own simulator*

It is also relevant to notice the use of mathematical programming techniques, like Linear Programming (LP) and ILP [Skarlat et al., 2017b,b; Wang et al., 2017], in the design of models to find the optimal location for services. Genetic Algorithm is another technique that is commonly used for the placement heuristics [Skarlat et al., 2017b; Guerrero et al., 2019b].

Most works use simulation for the validation, varying the tool used, largely iFogSim [Skarlat et al., 2017a,b; Taneja and Davy, 2017; Guerrero et al., 2019a; Mahmud et al., 2019; Paul Martin et al., 2020], but also YAFS [Lera et al., 2019a], FogTorch [Brogi and Forti, 2017], and Matlab [Deng et al., 2016], being Matlab the only not Fog-based simulation tool.

The open issues identified thus far are presented in the following subsection.

2.3.4 Open Issues

After studying the related literature it is feasible to identify possible research paths, in order to solve the problem of latency mitigation in communication networks, particularly in Fog environments.

From the analysis of the evaluated State-of-the-Art, some observations arise. Regarding the orchestration, the main strategy is using a centralized approach, which generates a single point of failure. An alternative is using a distributed approach, which leads to a more complex management with a restricted vision of the entire system. One possibility that has so far not been explored (to the best of our knowledge) is using a combined approach that takes advantages of the centralized unified vision of the entire system to apply better optimization solutions, without suffering of the single point of failure drawback. Another issue to take into consideration is that being the Fog an extension of the Cloud, it is natural to consider this environment as a whole, not limiting the orchestration solution to the Fog but encompassing the entire Cloud-to-Fog continuum.

The works on smart routing are mainly focused on solving the loss of connectivity or modification of routes due to mobility or changes in the infrastructure, to maintain the survivability of the applications, and not to improve their performance. The improvement of another factor, such as the latency, comes as a consequence but not as the main objective. On the other hand, the works on service placement are mainly focused on enhancing a particular factor (e.g., energy consumption, cost, latency).

In regards of the techniques to reduce latency by means of service placement, there is a lack of works that take into consideration the use of characteristics intrinsic to the applications and their services that help guiding the placement process. Most of the works are focused on using network metrics during their selection process. Using an application metric could take advantage of application profiling analysis since different applications might have different requirements and/or behavior. Even more, there is a possibility of using a combination of metrics during the placement process to help in the selection of the optimal location for a given service.

Thus, for the orchestration, one unexplored approach is using a hybrid solution to handle the management tasks, including a centralized instance in the Cloud able to apply a general vision into optimization mechanisms while allowing the distributed cooperation of Fog nodes in lower levels for a quick response time for dynamic reaction. Regarding the service placement, one possibility is studying the use of hybrid metrics, combining application and network related metrics that can guide the placement process to an optimal solution to minimize latency. Both metrics can change during time, adjusting themselves to reflect the current conditions of the network and the users' demands, unlike other metrics used thus far, such as hop count [Guerrero et al., 2019a]. This work presents an architecture for a Fog orchestrator following a hybrid approach, as well as different smart service placement mechanisms that use the popularity of the applications as well as the propagation delay of the network (source of delay

related with service and content placement, see Subsection 2.1.1) to guide the placement process with the objective of minimizing latency.

2.4 Chapter Summary

The research context, including concepts on latency and Fog environments was provided. Regarding the Fog, a study on its differences from the Cloud was performed, including the new emerging challenges that derive from the characteristics inherent to the Fog. This led to the conclusion of the need on novel orchestration approaches that encompass the challenges introduced by the Fog, taking into consideration its distributed nature.

Once the research niche was defined, a revision on the State-of-the-Art was carried out. Different works on orchestration, smart routing, and smart service placement were revised. The analysis that followed prompted the identification of research paths, that led to the mechanisms for latency reduction on Fog environments proposed later in this work.

For the orchestration, using a centralized approach for the upper levels and a distributed one for the lower levels is a possible way to manage this scenario, taking advantage of its characteristics. Regarding the latency reduction, service placement mechanisms based on mixed metrics that appraise both the applications and the current state of the network appear to be a novel solution to tackle this issue.

The findings presented in this chapter, particularly those related to the identification of research challenges in Fog orchestration, are reported in the following publication:

> Velasquez, K., Perez Abreu, D., Assis, M. R. M., Senna, C., Bittencourt, L. F., Laranjeiro, N., Curado, M., Vieira, M., Monteiro, E., and Madeira, E. (2018). Fog orchestration for the Internet of Everything: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications*, 9(14):1-23. Springer.

Chapter 3

An Orchestrator for the Fog

Contents

MANAGING the resources at the Fog is not an easy task. So far, a centralized approach with a single manager, or Orchestrator, is commonly used. This allows having a unified vision to apply general optimizations to the entire system. However, the response time for managing tasks is increased when applying this approach, besides generating the typical problem of centralized solutions, a single point of failure. It is important to be able to apply general optimizations, but it seems promising to combine this approach with a distributed one, that allows having quicker response times for dynamic reaction. In this chapter, a solution to manage resources in the Fog using a hybrid approach is presented. In the IoT at South-Bound Fog Levels, a distributed management of applications and services is proposed applying choreography techniques to enable automated fast decision making. A centralized approach to orchestrate applications and services taking advantage of a global knowledge of the resources available in the network is suggested for the North-Bound Fog and Cloud Levels. Furthermore, a smart orchestration framework is proposed, with special emphasis in its main module, the Service Orchestrator, for which implementation details are provided.

3.1 A Hybrid Approach for Service Orchestration in the Fog

To cope with the Fog challenges described in Chapter 2, while still guaranteeing an efficient resource management, a service orchestrator solution for the Fog using a hybrid approach combining orchestration and choreography is proposed in this section. Orchestration denotes a single centralized executable process that coordinates the interaction between different applications or services [Wen et al., 2017; Jiang et al., 2018], using a centralized approach to manage applications and services. On the other hand, choreography could be defined as a global description of applications and services, enabled by the exchange of information, cooperation and agreements between two or more endpoints, using a decentralized approach to manage applications and services [Bittencourt et al., 2018].

In order to fully understand the approach of orchestration and choreography of services in the Fog, a complete review of the general scenario is provided. The scenario, depicted in Figure 3.1, shows three levels: (1) IoT Level, (2) Fog Level, and (3) Cloud Level. The Virtual Clusters are at the IoT Level; these are composed by the grouping of terminal communication devices (e.g., smartphones, vehicles) that communicate among each other and also with the devices belonging to neighboring Virtual Clusters. This allows the movement of the IoT devices granting a higher level of freedom for the users.

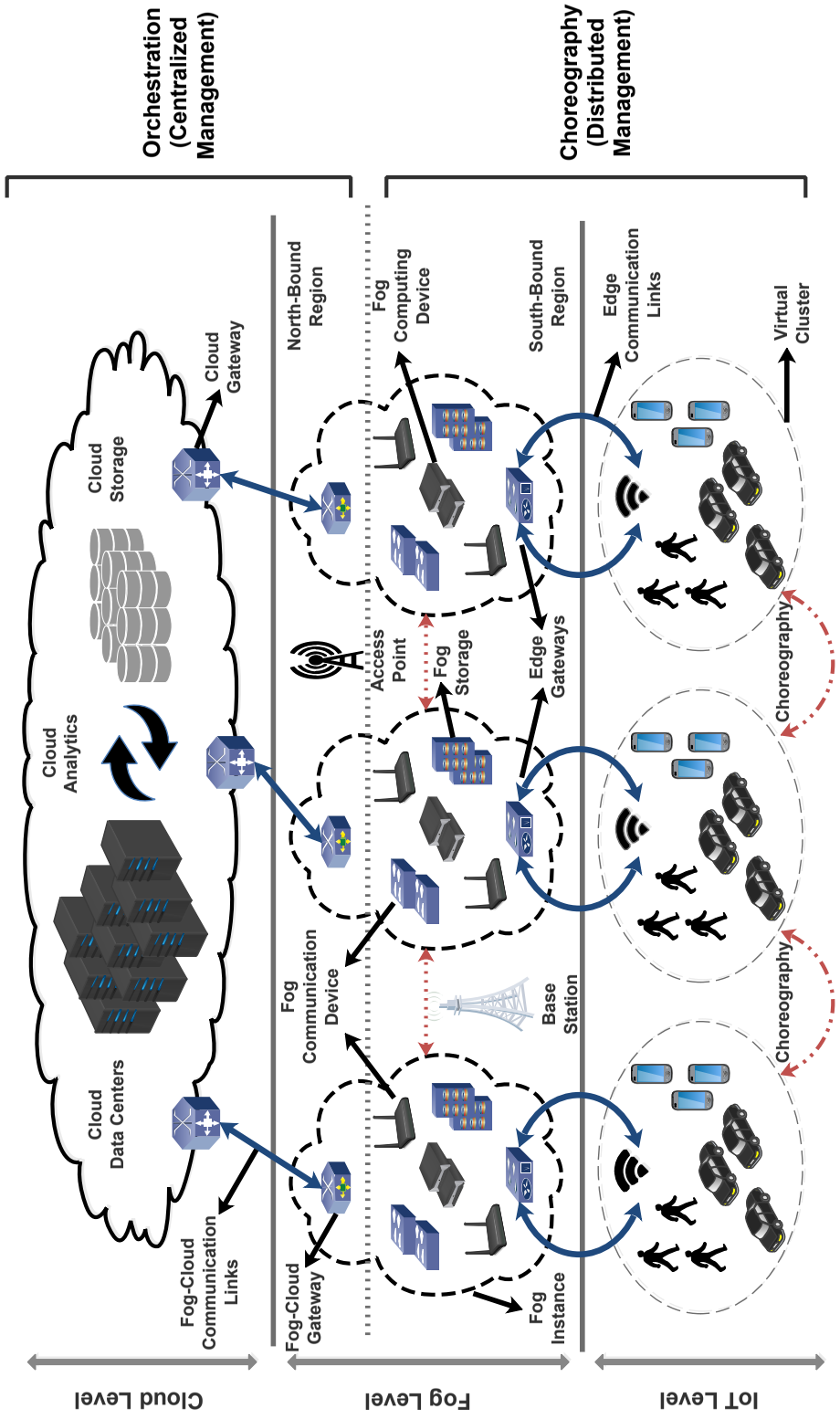


Figure 3.1: Logical Network Infrastructure for Service Orchestration

Communication among Virtual Clusters is achieved by the usage of choreography mechanisms that allow a quick reaction to possible changes in the topology (e.g., movement from one Virtual Cluster to another) as well as providing continuity to the services and thus higher resilience. Furthermore, this approach also benefits real-time and delay-sensitive applications by allowing a faster response among the devices (i.e., shorter paths), without the need of intervention of another device at a higher level of the infrastructure.

Edge Communication Links and Edge Gateways enable the communication of the IoT and Fog Levels. The Fog Level is sub-divided in the South-Bound Region and North-Bound Region. The South-Bound Region, closer to the IoT, is composed of Fog Instances. Each Fog Instance is a set of Fog Computing Devices, that allow different actions such as the migration of services for processing from the IoT devices (i.e., code offloading); Fog Communication Devices, that allow the connection between the various levels of the infrastructure, and also among the Fog Instances via Access Points and Base Stations; and Fog Storage Devices, that enable caching of content for nearby Fog and IoT users. This last part is also achieved by using choreography mechanisms.

For the resource management at the North-Bound Region of the Fog Level and Cloud Level, an orchestration approach is used. The communication between the North-Bound Region of the Fog and the Cloud is done via the Fog-Cloud Gateways and Fog-Cloud Links. The Cloud offers a massive amount of resources for heavy computation and storage requirements, usually known as Cloud Analytics. By using an orchestration approach at these levels, it is possible to have a global view of the system.

With the adoption of a hybrid approach, different advantages are achieved. For instance, it allows independence of the lower levels for their decision-making processes, which enables a quicker reaction in case of failures or topology changes, and also lower response time for time-constrained applications. Furthermore, in the upper levels, having a global view of the system allows applying long-term actions aiming at the optimization of the overall system.

3.2 A Modular Framework for Smart Orchestration

A modular framework based on the hybrid orchestration approach was designed. The framework will enable the use of smart orchestration schemes, including service placement, that guarantee the reduction of the latency according to the current state of the network and the location of users and servers, while taking advantage of the Fog scenario. The strategy behind the framework is gathering information from the underlying network and using this information to place the services in the most convenient locations.

The framework, depicted on Figure 3.2, is composed by three main modules, (1) Service Repository, (2) Information Collection, and (3) Service Orchestrator, plus the locations where the service instances are going to be placed. The upper levels, located at the Cloud, operate using orchestration, while the lower levels use choreography. The following subsections describe each module and

the interactions among them.

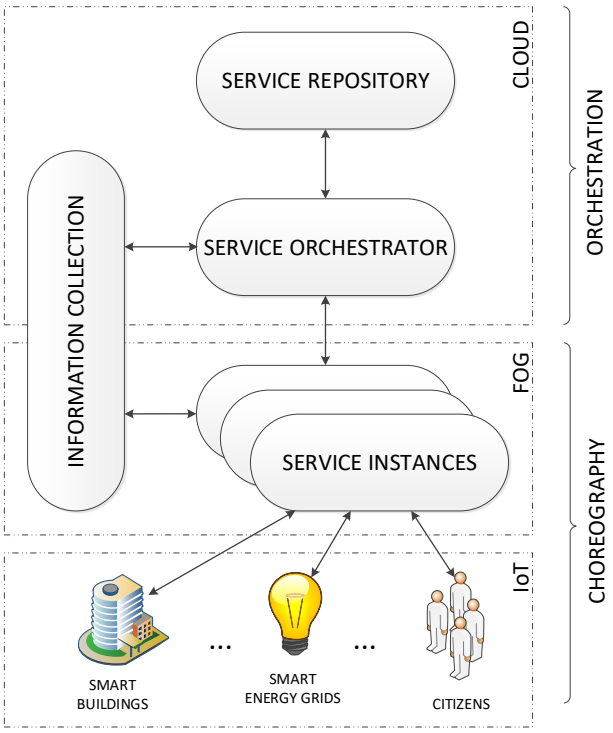


Figure 3.2: Modular Framework for Smart Orchestration

3.2.1 Service Repository

The first module is the *Service Repository*. This is the place where the available services are stored and categorized via a *catalog*. By physically locating this module in the Cloud, it will provide the framework with additional freedom to move the instances to different locations according to the needs of the users. Besides the actual services, this module also stores metadata about the services (e.g., requirements for a specific content) that will influence the location of each service.

3.2.2 Information Collection

The following module is the *Information Collection* module. This module will be in charge of gathering information from the network and about the interaction among the users and the services requested. This information, together with the metadata from the services, is going to influence the future tasks related with the service instances. To implement this module, an option is using the Application-Layer Traffic Optimization (ALTO) protocol [Alimi et al., 2014; Seedorf and Burger, 2009]. The ALTO protocol offers a mechanism to perform better-than-random peer selection in P2P networks, by obtaining information about the underlying network that helps to create an optimal overlay network, for instance, grouping devices that are located closer to each other, thus reducing

the response time in their communication. Among the pieces of information that can be obtained with the ALTO protocol are routing policies from Autonomous Systems (AS), bandwidth capacity of the nodes, and hop count. Although it was originally intended for P2P networks, this protocol can be applied to any kind of network.

The ALTO protocol uses two main information elements, called *network map* and *cost map* [Faigl et al., 2014]. The network map contains the description of groups of hosts without including information about the connection between those groups. For each group of hosts a Provider-defined IDentifier (PID) is defined; the PID could denote a subnet, a group of subnets, an AS, or a group of ASs. Thus, the network map describes the topology of the network. The cost map is associated to a network map and defines unidirectional connections between PIDs with an associated cost value for each one-way connection. The cost map also includes the definition of the metric type (e.g., hop count, routing cost) and the unit type (e.g., numerical, ordinal).

Another option to get information about the network is using the Simple Network Management Protocol (SNMP), that enables the exchange of management information between network devices [Zhou et al., 2015]. The SNMP model comprises four key elements: (1) the management station; (2) the agent; (3) the Management Information Base (MIB); and, (4) the network management protocol. The management station is the core of the system, in charge of network configuration and troubleshooting, among other tasks. The agents could be hosts, routers, and bridges; and are responsible for monitoring the network conditions, thus collecting information that in turn is sent to the management station. With this information, the topology of the network and its condition can be obtained. The MIB serves as a collection of objects that represents the knowledge shared, and the management protocol defines the exchange of information.

3.2.3 Service Orchestrator

The next module is called *Service Orchestrator*, which is the core of the framework. This module is going to implement the different global orchestration strategies, including those that make the decisions about the current location for every service instance. By using the data provided by the Information Collection module and the metadata from the services, it will be able to request the corresponding service instance from the Service Repository, and ultimately locate it in the appropriated location. This model gives the freedom to even combine some service instances in order to create more complex services. There are two possibilities to start the tasks of the Service Orchestrator, and they differ in how the initial placement is done. The first one is doing the initial placement randomly; and the second one is using a prediction about the state of the network. In both cases, subsequent placements are done by taking into account the information about the network.

Algorithm 3.1 shows the functionality for the Service Orchestrator using the random start approach. In this case, once the Service Orchestrator starts, it

places the services in random locations, to later adapt to the conditions of the network, that will be informed by the Information Collection module.

Algorithm 3.1: Service Orchestrator - Random Start

```
1  locations ← getLocations()
2  instances ← getInstances()
3  metrics ← getMetrics()
4  instances ← requestInstance(service)
5  servicePlacement(random, instances, locations, metrics)
6  do
7  |  wait(time)
8  |  maps ← requestMaps()
9  |  instances ← requestInstances(service)
10 |  servicePlacement(strategy, instances, locations, maps)
11 while TRUE;
```

The algorithm receives as input the locations of the servers (see Section 3.2.4) in line 1, where the service instances (line 2) are going to be deployed. Since this algorithm corresponds with a random start, the Service Orchestrator requests the instances of the services (line 4) and begins to place them randomly in the servers (line 5). Then, on line 7, a timer is used to allow some interactions that will feed the Information Collection module with data to create new maps. The definition of this timer is an important issue to take in consideration, since a small timer can generate unwanted oscillations in the system, but a large timer could result in the system not adapting to changes fast enough. The final tunning of the value of this timer will depend on experimental results, that can be analyzed using big data or machine learning techniques, to enable the definition of the timer according to each particular system needs. Afterwards, the Service Orchestrator requests the network and cost maps from the Information Collection (line 8) and with this data performs an informed placement of services (i.e., *instances*) in the Content Delivery Networks (CDN) servers (i.e., *locations*) for subsequent iterations of the algorithm. The strategy refers to the particular scheme used for the service placement task, such as using an evolutionary approach, a solution based on linear programming, or in greedy heuristics. This allows the Service Orchestrator to apply different strategies according to the needs of the system, defined by the AS administrator.

Algorithm 3.2 reflects the behavior of the Service Orchestrator using the prediction start approach. In this case, the Information Collection module is capable of creating a map without current information of the underlying network concerning the services that are going to be deployed, but using estimations from previous knowledge (e.g. traffic conditions) and from some data loaded by the system administrator, hence creating a "prediction" of the behavior of the network and of future interactions of the users. Loaded with these maps and with metadata from the Service Repository, the Service Orchestrator performs the placement of the services.

Algorithm 3.2: Service Orchestrator - Prediction Start

```
1 locations ← getLocations()
2 instances ← getInstances()
3 metrics ← getMetrics()
4 do
5 |   maps ← requestMaps()
6 |   instances ← requestInstances(service)
7 |   servicePlacement(strategy, instances, locations, maps)
8 |   wait(time)
9 while TRUE;
```

Once again, in line 1 the algorithm receives the locations as input, as Algorithm 3.1. On line 5, the Service Orchestrator requests the network and cost maps from the Information Collection module, that on the first iteration will consist on a prediction based on the conditions of the network and input from the system administrator (e.g., priority of the services). This information, combined with the metadata from the services, will allow the Service Orchestrator to perform a "better-than-random" placement of the services (line 7). Then, a timer is used to allow the interactions of the users with the Service Instances, which will generate new metrics for the Information Collection module, which in turn will create updated network and cost maps for the Service Orchestrator.

3.2.3.1 An Architecture for the Service Orchestrator

To manage the resources and communication in the scenario described in Figure 3.1, an architecture of the Service Orchestrator is proposed in Figure 3.3.

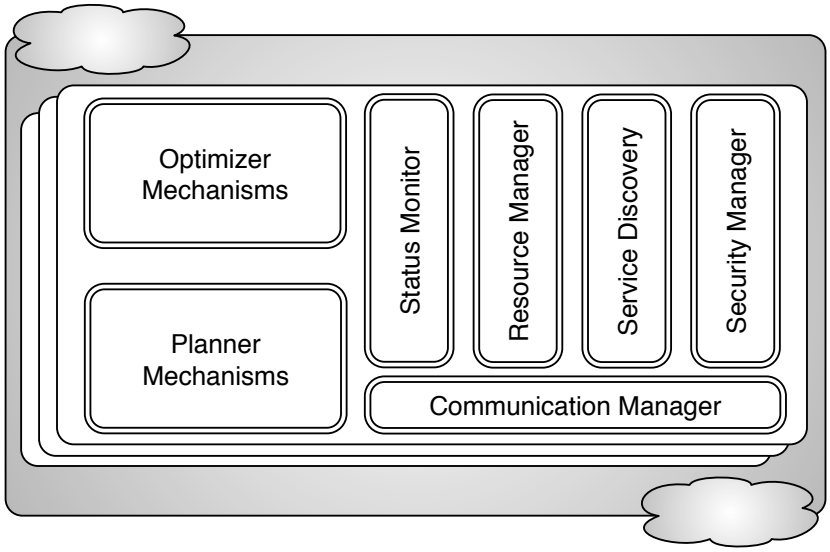


Figure 3.3: Hybrid Orchestrator Architecture

Overlapped instances of this structure ought to be replicated at different levels,

namely at Fog Instances (Fog Level) and Virtual Clusters (IoT Level), allowing the implementation of distributed choreography mechanisms; and at the Cloud Level, where a single logical instance is also deployed for global orchestration purposes. The orchestrator is composed of different modules. The *Communication Manager* controls the communication among the different orchestrator instances, deployed in Virtual Clusters, Fog Instances, and the Cloud. The *Resource Manager* monitors the resource usage of the various devices, including which of these resources are being used (and by whom) and which are available or idle. The *Service Discovery* module allows lookups for services and applications that are available and where is the nearest instance being executed. It also allows the aggregation or removal of services at any time. The *Security Manager* provides different mechanisms for authentication and privacy, according to the applications and services requirements.

The *Status Monitor* supervises the different activities in the system. It takes the information from the Resource Manager to guarantee that the required QoS and QoE levels previously agreed by the users are being met. The *Status Monitor* also gets information from the Information Collection module from the orchestration framework. The *Planner Mechanisms* schedule the execution of processes throughout the system. They set up where and when each service and application is going to be executed.

Finally, the *Optimizer Mechanisms*, only applied at the upper levels of the infrastructure, where a global view of the system allows making decisions to improve the QoS and QoE for final users. Both the Planner Mechanisms and Optimizer Mechanisms can be instantiated according to the particular requirements of the applications and services running at the Fog Instance or Virtual Cluster. Thus, resilience-focused mechanisms can be preferred over delay-sensitive or security-focused mechanisms, according to different needs.

3.2.4 Service Instances

For the *Service Instances* location, the framework combines paradigms such that the location of the service is optimized according to its service time. One of these paradigms is Fog computing. Because the Fog is located at the edge of the network, some features become attractive as part of the services the Fog can provide. Examples of these features are location awareness, low latency, support for mobility, and predominance of wireless access, as previously described in Chapter 2. The Fog paradigm is ideal to work with other technologies, like CDN. A Content Delivery Networks is a group of servers, with replicas of the content and the services from original servers, that allows to offload the work from those original servers [Krishnamurthy et al., 2001]. Once a request is placed, the CDN locates a server closer to the user (geographically, topologically, or by any parameter). By placing the non-original servers at the edge of the network (i.e., in the Fog), the latency can be reduced. For the proposed framework, the locations where the Service Instances are going to be placed, refer to CDN servers positioned in the Fog network.

The Service Orchestrator and the Service Repository, along with some compon-

ents (for instance, the server) of the Information Collection module could be deployed in the Cloud, together with the centralized component of the monitoring system in charge of collecting all the information and creating the predictions to be used. On the contrary, the Service Instances and the rest of the components of the Information Collection (e.g., the clients) should be located at the Fog (using choreography), closer to the users of the services, to take full advantage of the mobility support, location awareness, and low latency, offered by the Fog. Service Instances could have an agent deployed from the main orchestrator to help in the choreography tasks, sharing information with the main centralized orchestrator at the upper level via the Communication Manager of the orchestrator.

3.2.5 Interactions among the Modules

The interactions between the modules are described using sequence diagrams. As stated in subsection 3.2.3, two approaches have been used for the start-up process: (1) randomly placing the services for the first time; and, (2) using estimations of the network behavior to locate the service instance on the first attempt. Figure 3.4 shows the random start approach, where the initial placement is done randomly.

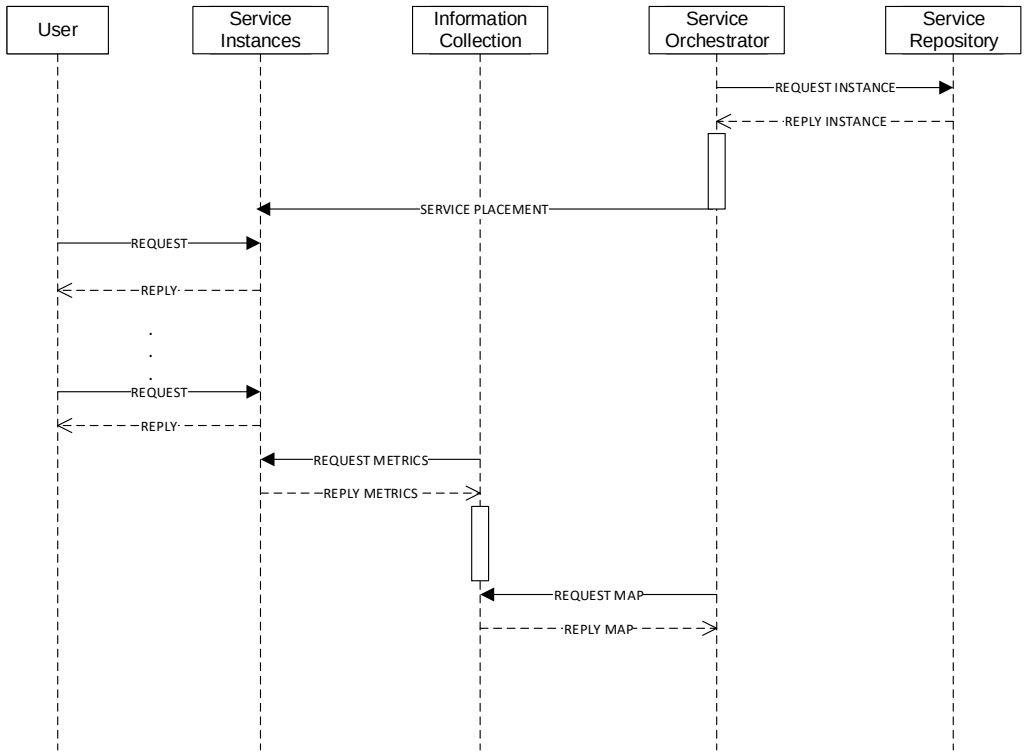


Figure 3.4: Sequence Diagram - Random Start

After some interactions with the users, the metrics are gathered by the Information Collection module and the network and cost maps are built (depicted by

the activity block in this module). Then, the maps are requested by the Service Orchestrator, that uses them to calculate the next best location for the service instances.

Figure 3.5 shows the prediction start approach. In this case, the Information Collection module calculates the network and cost maps based on the prediction about the network behavior. These maps are then delivered to the Service Orchestrator upon request, which in turn uses them to calculate the optimal locations for the service instances. After the services are placed, the interactions with the users begin, thus generating new metrics that will feed the Information Collection module to generate new network and cost maps, updated with the current information from the network.

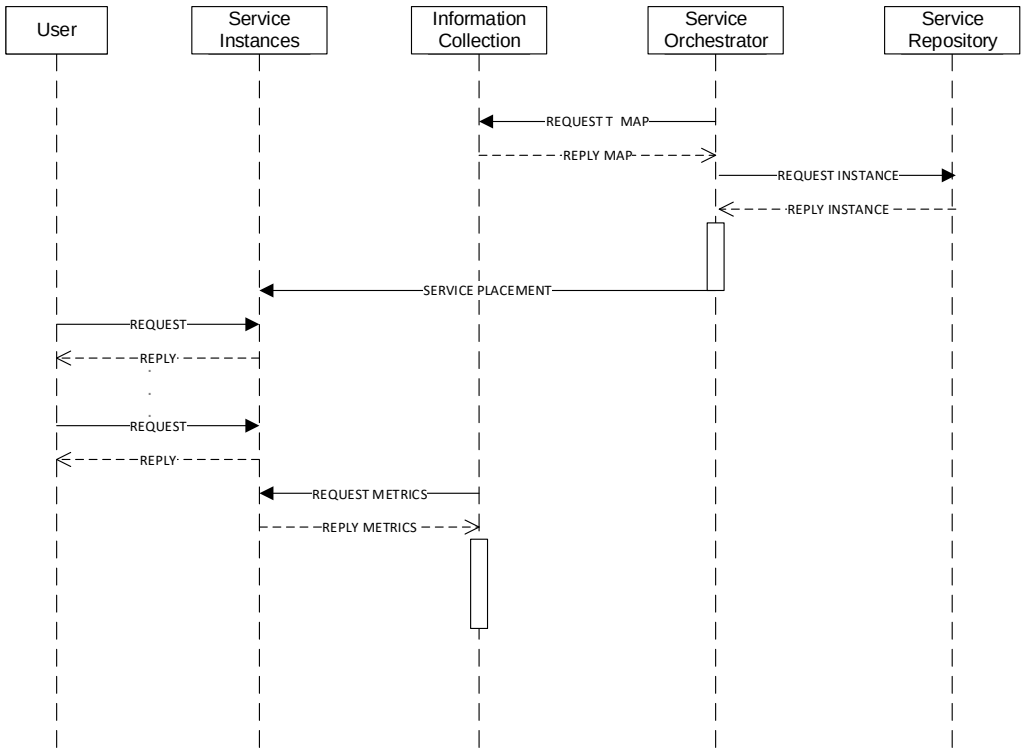


Figure 3.5: Sequence Diagram - Prediction Start

In both cases, the network and cost maps will continuously get updated by the collection of new metrics, as was depicted by Algorithms 3.1 and 3.2. This will influence future service orchestration decisions, such as those related to service placement. Additionally, the service instances will have some metadata that will help in the decision making process of the Service Orchestrator.

The hybrid orchestration approach described thus far allows the execution of complex management actions required in Fog environments. To demonstrate its usage, an example based on Smart City mobility support is provided next.

3.3 An Orchestrator Instance for Mobility Support

The orchestrator presented in Subsection 3.2.3.1 can be instantiated according to the requirements of the applications being executed in the different levels of the infrastructure. The modules of the orchestrator can be adapted to diverse needs to provide optimized solutions. This section describes the instantiation of the orchestrator to offer mobility support in a Smart City scenario.

Given the mobility of many devices in the IoT (e.g., connected vehicles, cyclists, and pedestrians), the Fog orchestrator must manage the Fog resources according to the demands required by such applications. Figure 3.6 shows the orchestrator instance that handles the mobility use-case. The Status Monitor, Resource Manager, Service Discovery, Security Manager, and Communication Manager modules remain the same, with the functionalities described in Section 3.2.3.1. The *Optimizer* is redefined with mechanisms specifically focused on mobility as well as the *Planner*.

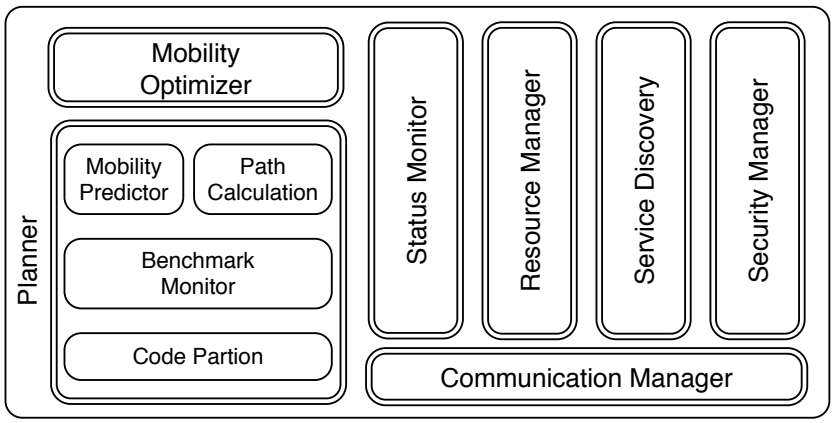


Figure 3.6: An Orchestrator Instance for Mobility Support

For this orchestrator instance, the Planner is composed of four specific mechanisms to support mobility. The *Mobility Predictor* deals with the precalculation of estimation regarding user mobility patterns. The mobility prediction could be carried out using machine learning techniques [Anagnostopoulos et al., 2011; Tang et al., 2019]. This estimation feeds the *Path Calculation* mechanisms, that determine the best route between the mobile devices and Fog Instances. The *Benchmark Monitor* works together with the Status Monitor to decide if the task offloading is actually needed, and if so, determines the optimal location for the services that belong to an application. To achieve this, the *Benchmark Monitor* evaluates the available resources both in the device and in the Fog Instances and also the expected QoS and QoE of applications and services. The *Code Partition* mechanisms determine which services are to be offloaded to the Fog Instance.

3.4 Interaction between the Elements of the Orchestrator

To understand how these mechanisms work together, a sequence diagram is shown in Figure 3.7. Three actors are involved in this example: the Status Monitor, the Planner, and the Virtual Machine (VM)/Container. The locally executed applications and services are monitored, and according to the resource demand can be migrated to a specific Fog Instance that guarantees the QoS and QoE requirements of applications and services. Once a request is received, the Planner must determine where the application (or parts of it) is going to be executed. The Status Monitor searches for information about the user device to determine if the application can be executed locally, or on the other hand, if it must identify the nearest Fog Instance that represents the best option for the user. According to the resource demand from the applications and the resources available in the Fog Instance, the Planner should select which tasks will have to be migrated from the IoT device to the Fog.

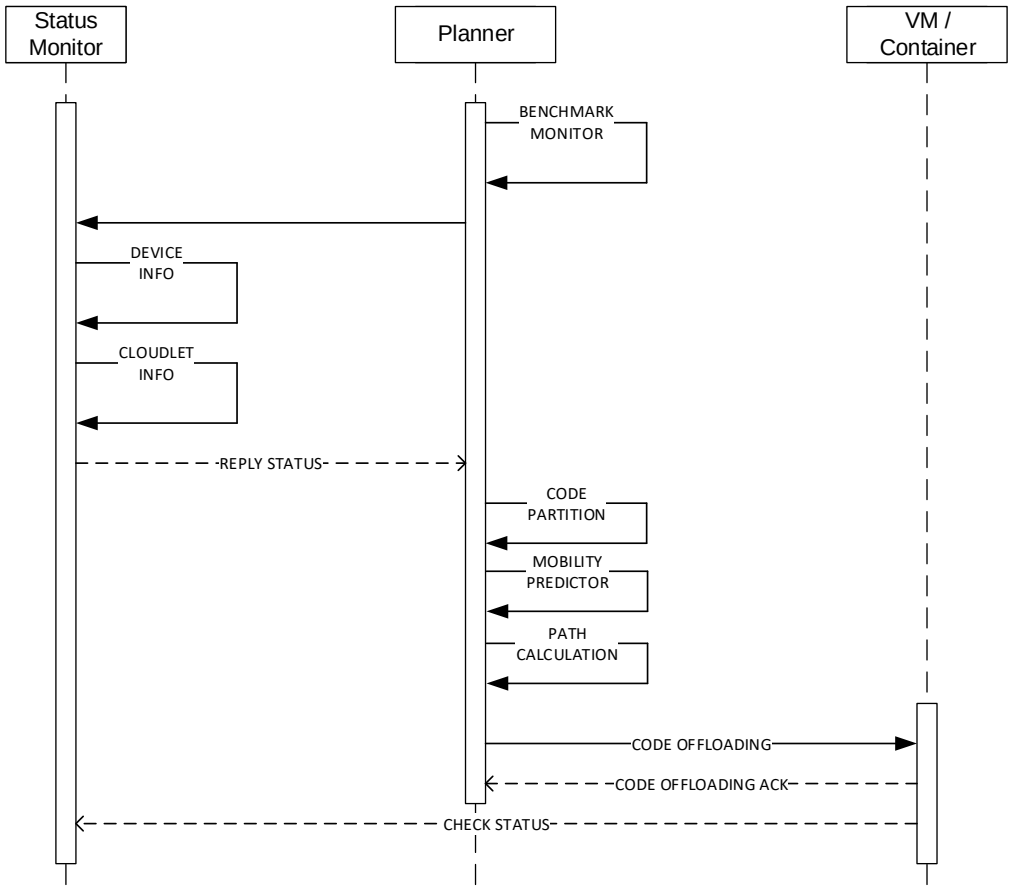


Figure 3.7: Sequence Diagram - Code Offloading in a Mobility Scenario

Given the mobility of the devices, the application should preferably be executed in the Fog Instances that offer the best conditions to the user while he is moving. In this context, it can be required to migrate an application from one Fog

Instance to another to offer the best execution conditions while the device is in movement. The Planner can select the Fog instances to act in the path. In the selection process, the Planner uses the device mobility data (provided by the Status Monitor), trajectory predictions (from the Mobility Predictor mechanisms) and data obtained using choreography from other Fog Instances (e.g., location and available resources).

From a set of preselected Fog Instances, the Planner can define which Fog Instance best fits the user in a specific point of its future path (Path Calculation). Another functionality that can be inferred from such path is the possibility to minimize the number of migrations among the Fog Instances. The applications can be distributed between the Fog Instances that will be available for longer within the device's route, before needing a new migration. The next step is offloading the code to the VM located on the selected Fog Instance.

In a normal flow of execution, the Status Monitor will oversee the methods of applications running in a VM/Container. The Planner will keep in touch with the application in order to identify the methods that can be migrated to the Fog without compromising its execution. The Status Monitor will oversee, in real-time, the resource demand for each method selected by the Planner. The data about the application resource demand and the available resources, both locally and remotely, is evaluated by the Status Monitor module. If the resources available in the device satisfy the application demands, then it will be executed locally. On the other hand, if the application requires a vast amount of resources, and the Fog Instance with which the device already established a connection offers enough resources, the Planner can conclude that offloading the application is the most beneficial procedure to the user.

After the process of offloading is completed, the application is executed in the Fog Instance inside of a VM/Container as long as this state is the most beneficial for the application. The Status Monitor registers data from the connection quality and idle resources in neighboring Fog Instances. This data helps the Planner to select the best location to place the application given the user movement in case of a needed migration.

This example is focused on the mobility support, as the other modules inside the orchestrator (e.g., Communication Manager and Resource Manager) will continue to carry out their basic tasks. Additionally, this example illustrates how the orchestrator can be adapted to handle specific use-cases inside a scenario as complex as a Smart City.

3.5 Chapter Summary

A hybrid approach to manage the Fog is proposed, using the combination of orchestration and choreography styles of management for different regions of the infrastructure. The solution outlined enables a global view which allows general optimization, and also automated dynamic reactions at the lower levels. Additionally, a modular framework for smart orchestration is presented. The framework defines the location for the smart orchestrator at the Cloud level,

with the service instances at the Fog, cooperating among them using choreography. For the service placement tasks, the framework is able to locate services in convenient servers at the Fog level (edge of the network) and continuously migrate these services according to the changing conditions of the network and status of the users, by learning information from the communication environment.

An architecture for the smart orchestrator is also outlined, including its modules and specific jobs for monitoring, resource managing, and optimizing the different administrative tasks in the Fog, including those of service placement. Moreover, by using different instances of the orchestrator, it is possible to apply different optimization goals according to the requirements of the applications. An example instance for mobility support is provided. The proposals presented in this chapter are partially reported in the following publications.

Evaluation of use-cases and open issues:

> Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2015). Towards Latency Mitigation in Emergency Scenarios. In *1st Cloudification of the Internet of Things (CIoT)*, pages 1-3, Paris, France. IEEE.

Description of the smart framework for Service Orchestration in the Fog:

> Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2017). Service Placement for Latency Reduction in the Internet of Things. *Annals of Telecommunications*, 72(1):105-115. Springer.

Characterization of the hybrid approach combining orchestration and choreography in the Cloud to Fog continuum:

> Velasquez, K., Perez Abreu, D., Gonçalves, D., Bittencourt, L. F., Curado, M., Monteiro, E., and Madeira, E. (2017). Service Orchestration in Fog Environments. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 329-336, Prague, Czech Republic. IEEE.

Use-case for mobility support for the Service Orchestrator:

> Gonçalves, D., Velasquez, K., Curado, M., Bittencourt, L. F., and Madeira, E. (2018). Proactive Virtual Machine Migration in Fog Environments, In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 742-745, Natal, Brazil. 2018, IEEE.

Chapter 4

Popularity-based Service Placement

Contents

The architecture presented in the previous chapter could be adapted to different requirements, while taking advantage of the Fog. The *Planner Mechanisms* module implements different mechanisms to deal with resource management. One of these mechanisms is related to the service placement, which consists in the selection of the appropriate execution node for the applications, according to specific optimization objectives. Different metrics can be used to guide the placement process. So far, works related to placement in the Fog have used metrics associated to the communication infrastructure (e.g., energy consumption, propagation delay, hop count) or to the application (e.g., QoS). On the contrary, this chapter proposes a novel approach that combines two metrics, one related to the communication infrastructure (i.e., propagation delay) and one related to the applications (i.e., popularity) with the objective to prioritize popular applications and minimize their latency.

Two mechanisms are presented in this chapter, the first one is an ILP formulation to find the optimal solution for service placement based on popularity, aimed at the reduction of latency in Fog environments. This approach will provide the lowest theoretical latency bound. The second mechanism is a heuristic called Popularity Ranked Placement (PRP), based on graph partition, to offer an option close in quality to the optimal solution, while being significantly less time-consuming. Both mechanisms are evaluated via simulation.

4.1 An Optimal Solution for Service Placement using Integer Linear Programming

This section introduces an ILP model to maximize the placement of popular applications, while minimizing the latency for final users in Fog environments.

A lexicographic formulation of the optimization problem is considered. First, with the goal of serving the largest number of users, the problem consists of maximizing the selection of accepted requests, given the popularity of the applications. Then, the second problem consists of placing the services that belong to an application in the Fog nodes that minimize latency, given the selection of the requests obtained in the first problem. The constraints from the first problem are kept for the second in order to ensure feasibility. The goal is to reduce the latency; hence the overall solution of solving both problems should provide a lower latency to the maximum amount of users.

4.1.1 Parameters and Variables

Table 4.1 summarizes the parameters and variables for the model. Regarding the parameters, the set of entry points from where requests are generated is labeled as GW; these correspond to the gateways from where clients access the Fog environment. The instance matrix I reflects the micro-services that compose the applications. This means that an application can be built by the combination of a set of services. $I_{a,s} = 1$ if service $s \in S$ belongs to application $a \in A$, and

0 otherwise. The cost matrix C contains the propagation delay (as a source of latency impacting placement decisions, see Subsection 2.1.1) from the shortest path that connects each pair of nodes $n \in N$ and gateways $gw \in GW$.

Table 4.1: Parameters and Variables for the ILP Model

Parameters	
Parameter	**Description**
S	Set of services to be placed
N	Set of nodes where the service can be executed
GW	Set of gateways
A	Set of applications. An application is composed by a set of services
R	Set of requests for all the applications
Q_a	Sum of requests for $a \in A$
Ω_n	CPU capacity for $n \in N$ (GHz)
Φ_n	Memory capacity for $n \in N$ (GB)
ω_s	CPU requirement for $s \in S$ (GHz)
φ_s	Memory requirement for $s \in S$ (GB)
I	Instance matrix. An $\|A\| \times \|S\|$ matrix
C	Cost matrix. An $\|N\| \times \|GW\|$ matrix
Variables	
Variable	**Description**
K	Acceptance matrix. An $\|A\| \times \|R\|$ matrix
P	Placement matrix. An $\|A\| \times \|R\| \times \|S\| \times \|N\|$ matrix

Concerning the variables, the acceptance matrix, K is a binary matrix that indicates which requests are accepted for each application. $K_{a,r} = 1$ if the r-*th* request for application $a \in A$ is accepted, and 0 otherwise. P represents the placement matrix, indicating the final location for the services. The matrix relates the requests per application, and the nodes where the services belonging to those applications are finally placed; $P^{a,r}_{s,n} = 1$ indicates that service $s \in S$ is executed on node $n \in N$ to satisfy request $r \in R$ for application $a \in A$, and $P^{a,r}_{s,n} = 0$ otherwise.

4.1.2 Maximizing the Placement of Popular Applications

The first step in the optimization solution is to select the applications with the highest amount of requests, which are considered as the most popular. By prioritizing the most popular applications a larger number of final users will benefit from lower latency. Equation (4.1) depicts the main objective function of the ILP model. Since the main objective is to maximize the selection of popular applications, the set of requests per application is used to determine which applications have higher priority in the selection.

$$\max \sum_{a \in A} Q_a \times \sum_{r \in R} K_{a,r} \tag{4.1}$$

The first constraint in the model, shown in Equation (4.2), is meant to ensure that the selection of applications is only carried out when the request can be fulfilled entirely. The constraint in Equation (4.3) guarantees that the services that belong to an application selected are executed in only one server. Equation (4.4) forces that all the services belonging to an application can be executed before selecting it for execution.

$$P_{s,n}^{a,r} \leq K_{a,r} \quad \forall a \in A, r \in R, s \in S, n \in N \tag{4.2}$$

$$\sum_{n \in N} P_{s,n}^{a,r} \leq 1 \quad \forall a \in A, r \in R, s \in S \tag{4.3}$$

$$K_{a,r} \times I_{a,s} = \sum_{n \in N} P_{s,n}^{a,r} \quad \forall a \in A, r \in R, s \in S \tag{4.4}$$

Memory and CPU restrictions are also imposed as constraints to the model, to enforce resource limits in the execution nodes. Equation (4.5) describes the CPU constraint and Equation (4.6) does the same for the memory constraint. For both equations, the sum of processing requirements from all services can not surpass the available resources in the nodes.

$$\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} P_{s,n}^{a,r} \times \omega_s \leq \Omega_n \quad \forall n \in N \tag{4.5}$$

$$\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} P_{s,n}^{a,r} \times \varphi_s \leq \Phi_n \quad \forall n \in N \tag{4.6}$$

While the resources of the nodes are usually fixed and well known, an estimate is used for the resource requirements of the services (i.e., CPU and memory). Service profiles can be created by gathering information from previous executions. In this work, services requirements are generated using a bounded random approach (see Section 4.3).

4.1.3 Minimizing the Latency

The second optimization goal is to reduce the latency of the most popular applications, selected on the first optimization problem. Equation (4.7) depicts the formulation. The main idea is to minimize cost, represented by the propagation delay of the links towards a node ($C_{n,gw}$). The propagation delay is one of the latency components related with placement tasks, as stated in Chapter 2, Section 2.1. The cost can be changed to another gauge, e.g., energy consumption, to model a different requirement but continue to use the same optimization model.

$$\min \sum_{a \in A} \sum_{r \in R} \sum_{n \in N} \sum_{s \in S} \sum_{gw \in GW} P_{s,n}^{a,r} \times C_{n,gw} \tag{4.7}$$

In order to guarantee that the results from the first optimization step are maintained (i.e., the popular applications are still selected), the acceptance matrix resulting from the first optimization problem is now a parameter matrix and used as input for the placement process in the second optimization problem. Therefore, during this step of the optimization, the services are placed in the most convenient nodes in the Fog, aiming at the reduction of the latency of the services; see Equation (4.8).

$$\sum_{s \in S} K_{a,r} \times I_{a,s} \quad = \quad \sum_{s \in S} P_{s,n}^{a,r} \quad \forall a \in A, r \in R, n \in N \tag{4.8}$$

Constraints from the first optimization problem (i.e., Equation (4.2) through (4.6)) are kept in this step to guarantee feasibility. In the following section, an alternative heuristic based on the PageRank algorithm is proposed.

4.2 A Ranked-based Heuristic for Service Placement in the Fog

Although optimization solutions are often used for offline environments and as a theoretical threshold, they are not usually applied in online or more realistic scenarios given their lack of adaptability and also their high response times [Lee and El-Sharkawi, 2008]. Furthermore, by optimizing the selection of requests that come from the gateways, the procedure will most likely place the majority of the applications in the same Fog nodes, thus potentially creating an overload in the same nodes and links in case of heavy load. To overcome this issue, one alternative is using graph partition to create *communities* in which the applications can be deployed, balancing the load among the nodes inside the community [Lera et al., 2019a; Naas et al., 2018; Skarlat et al., 2017a].

There are several option to partition a graph, including Fluid Communities

[Parés et al., 2018], Girvan-Newman [Newman and Girvan, 2004], and label propagation [Liu and Murata, 2010]. However, only grouping the nodes is not enough, since a neighbor node could have higher latency that a non-neighbor node. Hence, ranking the nodes for the community creation process (e.g., via probability propagation) is a better option. Using the probability of the node to appear in the *shortest path* of communication for a source node would lead to a better community to eventually minimize latency. The PageRank algorithm [Page et al., 1999] can also be used to partition graphs, with the particularity that the partition process is guided by a metric. In this case, the propagation delay is used as a metric for the community creation process. The original PageRank algorithm is introduced next in this section to later on describe how it is adapted for the PRP heuristic.

4.2.1 The PageRank Algorithm

The PageRank algorithm was first introduced to rank web pages in a search engine, and it was based on a summation derived from bibliometrics research (i.e., the analysis of the citation structure among academic papers) [Page et al., 1999]. The idea behind the PageRank algorithm is to rank the nodes in a graph via probability propagation. To understand how it works, an example is provided in Figure 4.1. Consider the graph $G = (V, E)$, where $V = \{A, B, C, D\}$ and $E = \{(A, B), (A, C), (B, D), (C, A), (C, B), (C, D), (D, C)\}$. At the beginning of the algorithm, it is assumed that all the nodes in set V have the same rank $r = \frac{1}{n}$, where n denotes the cardinality of set V.

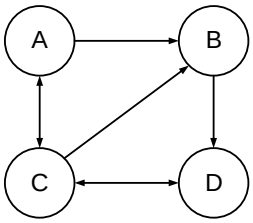


Figure 4.1: A Graph Example to Illustrate the PageRank Algorithm

Nodes are ranked following an iterative approach, according to Equation (4.9) where B_{n_i} is the set of nodes pointing into node n_i and $deg^+(n_j)$ is the number of out-links from node n_j. Let $r_{k+1}(n_i)$ be the rank of node n_i at iteration $k+1$. The ranking process begins with $r_0(n_i) = \frac{1}{n}$ for all nodes and is repeated until the PageRank scores converge to final stable values [Langville and Meyer, 2012]. The values obtained after applying the first three iterations of Equation (4.9) to the graph depicted in Figure 4.1 are shown in Table 4.2.

$$r_{k+1}(n_i) = \sum_{n_j \in B_{n_i}} \frac{r_k(n_j)}{deg^+(n_j)} \tag{4.9}$$

The implementation of the PageRank algorithm could be sped-up using matrices [Langville and Meyer, 2012]. The idea is to use an $N \times N$ matrix

Table 4.2: PageRank Values per Iterations

Iteration 0	**Iteration 1**	**Iteration 2**	**PageRank**
$r_0(A) = \frac{1}{4}$	$r_1(A) = \frac{1}{12}$	$r_2(A) = \frac{1.5}{12}$	1
$r_0(B) = \frac{1}{4}$	$r_1(B) = \frac{2.5}{12}$	$r_2(B) = \frac{2}{12}$	2
$r_0(C) = \frac{1}{4}$	$r_1(C) = \frac{4.5}{12}$	$r_2(C) = \frac{4.5}{12}$	4
$r_0(D) = \frac{1}{4}$	$r_1(D) = \frac{4}{12}$	$r_2(D) = \frac{4}{12}$	3

H in combination with a $1 \times N$ row vector π^T, which holds the PageRank value of all nodes in each iteration. The matrix H is a row normalized hyperlink matrix with $H_{ij} = \frac{1}{deg^+(n_j)}$ if there is a link from node n_i to node n_j, and 0, otherwise. Thus, nonzero elements in H represent the transition probability from one node to another. The corresponding H matrix for the graph depicted in Figure 4.1 is shown in Equation (4.10).

$$H = \begin{pmatrix} 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & 0 & 1 \\ 0 & 1 & \frac{1}{3} & 0 \end{pmatrix} \tag{4.10}$$

The nonzero elements of row i in H correspond to the out-link nodes of node i; meanwhile, the nonzero elements of column i denote the in-link nodes of node i. Regarding the rank of the nodes, the row vector $\pi^{(k)T}$ represents the PageRank vector at the *k-th* iteration of the algorithm. Consequently, Equation (4.9) can be written using a matrix notation as shown in Equation (4.11).

$$\pi^{(k+1)T} = \pi^{(k)T} H \tag{4.11}$$

From Equation (4.11), the following observations arise [Langville and Meyer, 2012]:

- Each iteration requires one vector-matrix multiplication, considering that H is a very sparse matrix, the vector-matrix operation is reduced to $O(n)$ computational effort;
- The iterative method used is a linear stationary process performed applying the power method to matrix H; and
- H could be seen as a stochastic transition probability for a *Markov* chain, where the dangling nodes of the graph create 0 rows in the matrix and the other rows belong to the non-dangling node keeping stochastic values.

Consequently, H is considered a sub-stochastic matrix. In this work, the PageRank algorithm described above is adapted to rank the nodes in a network to-

pology using the Propagation Delay (PD) of the links to weight the transition probability between nodes. A heuristic based on this ranking process is described next.

4.2.2 Ranking nodes to create Communities

To avoid the potential issue concerning the overload of the nodes close to the gateways and the gateways themselves, an alternative is to create *communities* that can share the load of the gateways while keeping the applications deployed in their proximity, as explored in other works [Skarlat et al., 2017a; Lera et al., 2019a]. However, only selecting neighboring nodes to share the load could be restrictive since there is a chance that a non-neighbor is better qualified than the neighbors (e.g., has lower latency connectivity). Thus, ranking the nodes according to the probability in which they would communicate (e.g., be chosen in the shortest path) represents a better selection criterion to build these sharing groups or communities. In this work, the ranking process described above is used to create those communities.

The heuristic takes advantage of the low latency levels in the Fog by deploying the modules of the applications through the Fog infrastructure, and near to the final users whenever possible. When the Fog runs out of resources, deployment in the Cloud will be carried out. The ranking of the nodes is created according to the procedure described in Subsection 4.2.1. For a weighted graph, the probability of moving from one node to the next will not only depend on the presence of a link but on a metric based on the weights of the edges. In this work, the Propagation Delay is used as a metric of the link latency. Thus, nodes connected through a link with a lower PD will have a higher probability to connect since they will be part of the shortest path towards the destination node.

The community-building process is described in Algorithm 4.1. The first step is to get the topology graph (line 1), and to remove the Cloud node since it will not be part of any community (i.e., only being used when there are no resources available in the Fog), and also transforming it as a complete directed graph. Following this, the nodes are ranked using the process described in Subsection 4.2.1. The PD is used to weight the transition probability from one node to another in the graph (line 2).

The communities are built by selecting all the nodes with a transition probability higher than a threshold (line 7). For this work, the threshold was defined in 0.1, to create bigger communities, but the value could be adapted following a different criterion. After this, a community is created by combining this initial community with all the communities of its initial members (line 14).

All the Fog nodes that were not assigned to any community during this process are grouped in the final step, as shown in line 16. The results of this process include (see line 17):

- A structure with the communities for all nodes;

Algorithm 4.1: Build Communities

```
   Result: Communities for all nodes in the infrastructure
1  topology ← getTopology()
2  prank ← PageRank(topology, weight=PD)
3  communities ← ∅
4  non_communities ← ∅
5  foreach node in topology do
6  |   foreach element in prank do
7  |   |   if prank[element] ⩾ threshold then
8  |   |   |   Add element to communities[node];
9  |   |   end
10 |   end
11 end
12 avg_size ← getCommunitySize(communities)
13 foreach node in topology do
14 |   mergeCommunities(node, communities);
15 end
16 non_communities← prank − communities;
17 return communities, non_communities, avg_size
```

- A structure with the nodes that do not belong to any community; and
- The average size of the initial communities.

Once the communities are built, the deployment process begins, as explained in the following subsection.

4.2.3 Popularity Ranked Placement

The heuristic proposed, called Popularity Ranked Placement, uses the communities described in the previous subsection for the placement process, and is presented in Algorithm 4.2. After creating the communities (line 3), the applications to deploy are ranked according to their requests (i.e., to prioritize popular applications, as in the ILP model described in Section 4.1), as shown in line 5. Also, the nodes in each community are ranked according to the probabilities calculated in the previous step.

The first option is to deploy the modules of the applications within the community of the Gateway (GW) from which the request to said application was launched. The search space inside the community is limited by an expanding window, which size is set according to the average size of the initial communities (line 6). The expanding window technique allows to minimize the internal fragmentation of resources inside of the nodes, while allowing to place a less demanding module in the already explored nodes of the community. The searching process within the windows follows a Round Robin (RR) approach until no more

Algorithm 4.2: Popularity Ranked Placement

```
   Result: Placement of applications' modules
1  placement_matrix ← ∅
2  topology ← getTopology();
3  community, non_community ← Build Communities()
4  reqs ← getRequests();
5  apps ← rankApps();
6  expWin ← getAvgCommunitySize();
7  foreach req in reqs do
8      while req > max(WindowNodeCapacity) ∧
9      ¬ ReachCommunitySize do
10         Expand expWin;
11     end
12     if req ≤ max(WindowNodeCapacity) then
13         deploy(placement_matrix, community);
14     else
15         if req ≤ max(NonComNodeCapacity) then
16             deploy(placement_matrix, non_community);
17         else
18             deploy(placement_matrix, Cloud);
19         end
20     end
21 end
22 return placement_matrix
```

resources are available to host the module. In the case that the deployment succeeds, the RR pointer is updated accordingly.

When the community window runs out of resources to deploy the module of the application, the size of the expanding window is augmented by the average size of the initial communities, until the size of the community is reached, and a new search process to find the hosting node is carried out (lines 8 - 11). The process is also illustrated in Figure 4.2. The community for node A is shown in Figure 4.2(a). The initial window size in this example is 3 (i.e., the average size of the initial communities). Once the subset contained inside the expanded window runs out of resources (there is no node that can host the application's module to deploy), the window is augmented by its original size, 3 in this case, as depicted in Figure 4.2(b). This process is repeated until finding a node with enough resources to fit the application's module to deploy, or until the community size is reached, as seen in Figure 4.2(c).

If the expanding window reaches its maximum size (i.e., the community size) and there are not enough resources to satisfy the request, a new search process is initiated in the non-community nodes (also a resulting structure from the previous step, see Subsection 4.2.2) following a First Fit approach (lines 15 - 16). Thus, Fog nodes are prioritized for placement before trying to deploy the

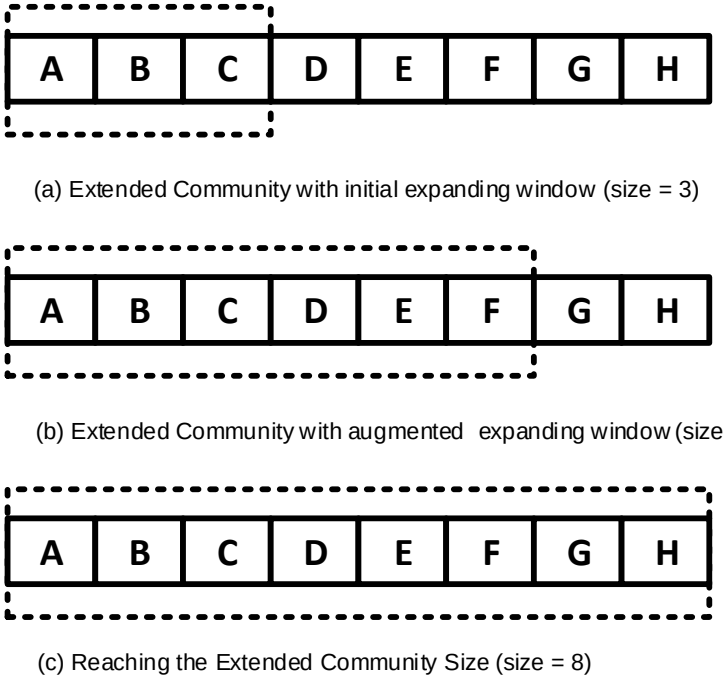


Figure 4.2: Expanding Window

application's modules in the Cloud. If all previous attempts fail, the module is deployed in the Cloud (line 18).

The validation of the mechanisms presented in this chapter is carried out via simulation experiments, as it was seen in Chapter 2 that it was commonly used to evaluate the performance of this type of mechanisms. The evaluation setup is described in the next section.

4.3 Experimental Evaluation

The workflow for the experimental evaluation is depicted in Figure 4.3. Configuration parameters are used to generate three files: one for the definition of the network topology, one for the definition of the applications, and one final file for the user definition, indicating the gateways from which the requests are originated. This information is used to create the orchestration catalog, that is going to contain information regarding the network, applications, and users (e.g., requests by application or popularity, requirements of the services, resources of the nodes). With the catalog ready, the mechanisms are executed to determine the locations where the services are going to be placed and use this information for the simulations using YAFS. The raw data generated by YAFS is used to plot the results that are presented in the following section.

The main goal of the service placement mechanisms was to minimize the latency of the entire system, while prioritizing popular applications. Thus, the experiments were designed to measure the latency of the system, and to observe with a finer granularity level the performance of popular applications regarding

latency.

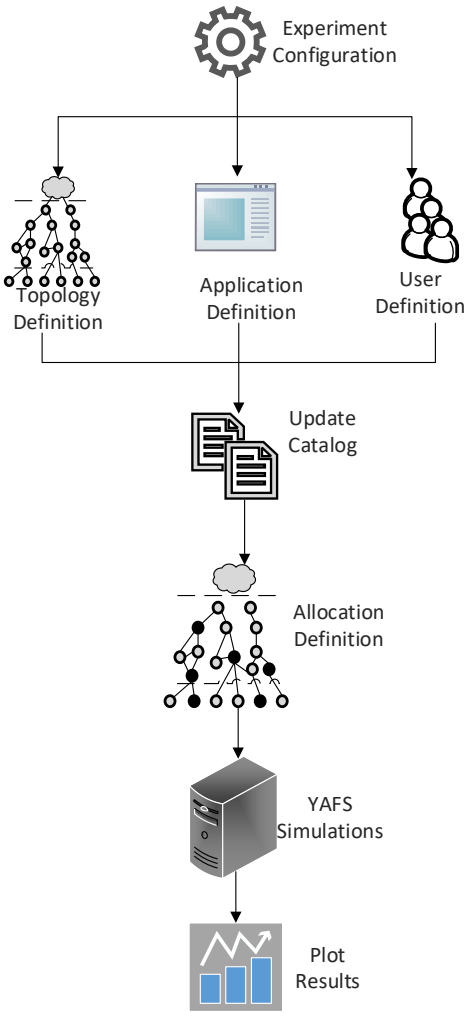


Figure 4.3: Evaluation Workflow

The validation was performed using YAFS [Lera et al., 2019b] because of its strong support of Fog critical features and high granularity of reported results [Perez Abreu et al., 2020]. The experiments were conducted on a PC with 32GB 2400MHz DDR4 RAM and 2.80GHz Intel Core i7-7700HQ with 4 cores and 8 threads (2 threads per core) processor. The PC was running Microsoft Windows 10 Pro (Build 18363) operating system. Python 2.7.16 was used for YAFS. For the ILP model, the IBM CPLEX Optimizer version 12.9 was used [IBM, 2019].

Regarding the network topology, a graph was generated according to the complex network theory, following a random Barabasi-Albert network [Jalili and Perc, 2017]. 50 Fog nodes comprise the network, and an additional node was added to represent the centralized Cloud. This node is connected to the Fog nodes with the highest betweenness centrality in the graph. The nodes with lowest betweenness centrality were appointed as GW, representing the nodes at the edge of the network.

Applications were randomly generated following a *Growing Network* graph structure [Yao et al., 2014]. This means a vertex is added one at a time with an edge to the last added vertex. Two types of applications were modeled. The first type is depicted in Figure 4.4, where the communication flows in a single direction. This allows modelling typical IoT applications, such as sensing applications, commonly used in Smart City scenarios. We called this type of applications *Fire&Forget*, based on their messaging pattern [Davies et al., 2008; Gutierrez, 2017].

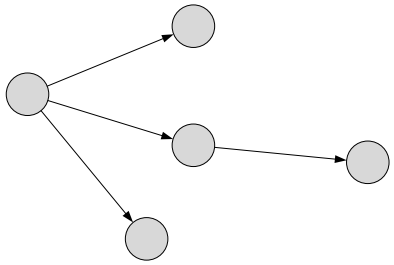


Figure 4.4: Fire&Forget Applications

A second type of applications, shown in Figure 4.5, was used to model a broader spectrum of services. Applying the same approach as before and after the graph is complete, two vertices are randomly selected (excluding the source) to generate an information flow to the source vertex. This allows simulating delay-sensitive applications that can be deployed at Cloud/Fog environments. For instance, an augmented reality application that can capture location-related information, process it, and send a reply to the user; or an eHealth application where some medical values are captured from the patient, processed and/or stored for later analysis, and sent to the health specialist for evaluation, or to regulate a patient's medication. Based on their messaging pattern [Gutierrez, 2017], we called these applications *Asynchronous Response*.

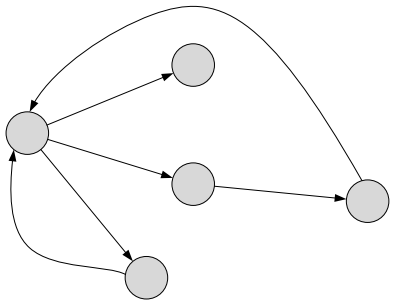


Figure 4.5: Asynchronous Response Applications

The network load was varied to evaluate the performance of the proposals under different conditions. Four scenarios were defined: (1) *tiny*: 5 different applications, (2) *small*: 10 applications, (3) *medium*: 15 applications, and (4) *large*: 20 applications. Each of the applications has at least one request. To simulate the popularity of the applications, the number of requests was determined using

a uniform distribution. All the scenario setup, as well as the source code, is available via a GitLab repository [Velasquez et al., 2019].

The parameters were set according to the values displayed in Table 4.3, for each network link, Fog node, GW, and application. Similar values have been previously used in related work [Lera et al., 2019a]. The application demands are measured using the YAFS' resources unit, defined as a vector containing the capacity of different computational resources (e.g., number of cores for CPU, GB for memory, or TB for the hard disk).

Table 4.3: Parameters Values for Experiments

	Parameter	Value (min - max)
Network	Propagation Delay (ms)	2 - 10
	Bandwidth (bytes/ms)	75000
Fog	Resources (units)	10 - 25
	Speed (instr/ms)	500 - 1000
GW	Request rate (1/ms)	1/1000 - 1/200
	Popularity (prob)	0.25
Application	Services (number)	2 - 8
	Resources (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000

The ILP solution and the proposed heuristic (PRP) are compared with the well known FF algorithm, as it was used in other works for evaluation purposes [Skarlat et al., 2017a,b]. For the FF algorithm, the nodes were organized according to their resources, from lowest to highest, to prioritize nodes with less resources that usually are deployed at the edge of the network, closer to the users. All solutions were executed once before the simulations, i.e., a static service placement is considered, and 30 simulations were executed using these static placements to mitigate the statistical error, including 95% confidence intervals in the plots.

4.4 Results and Analysis

The performance of each placement method (ILP model, PRP, and FF) is presented in this section. The first metric to evaluate is the latency (for this case, the OWD as defined in Chapter 2 is reported). The latency is calculated as the sum of the transmission times among the application's modules [Lera et al., 2019b]. The results are grouped by scenario and displayed in milliseconds.

Figure 4.6 depicts the results for the Fire&Forget applications while Figure 4.7 shows the performance of the Asynchronous Response applications. It is possible to note that in both cases, the best results correspond to the ILP optimization model, as expected. As the load grows, the latency increases, which is also expected.

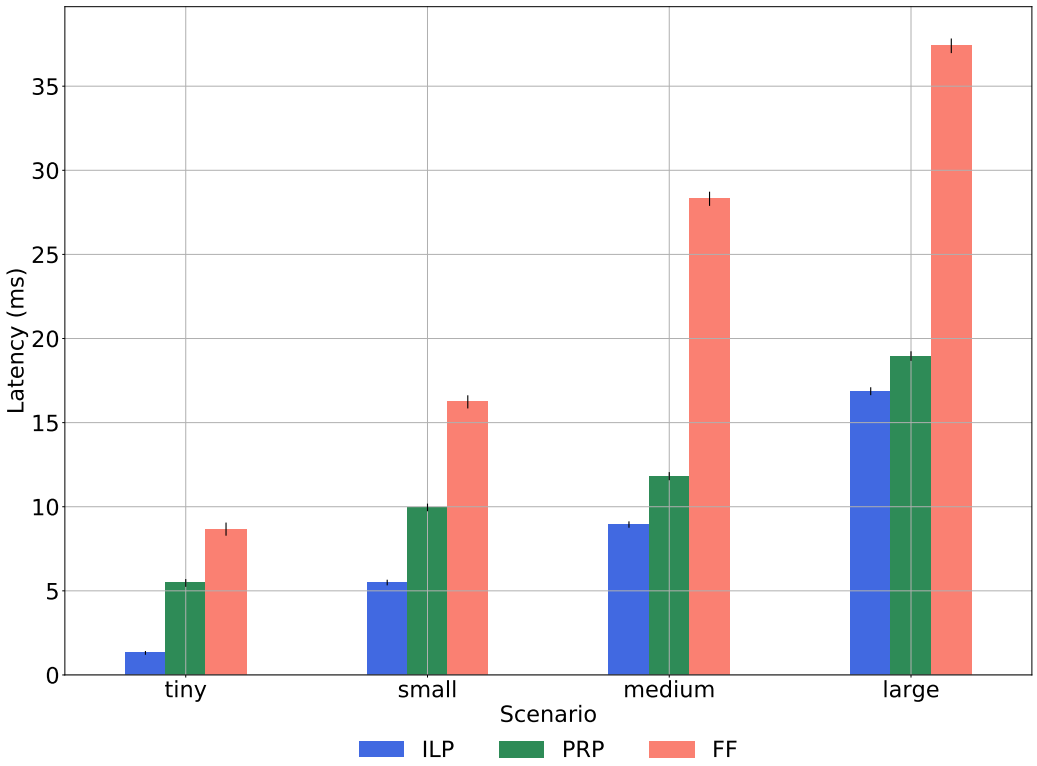


Figure 4.6: Total Latency by Scenarios - Fire&Forget Applications

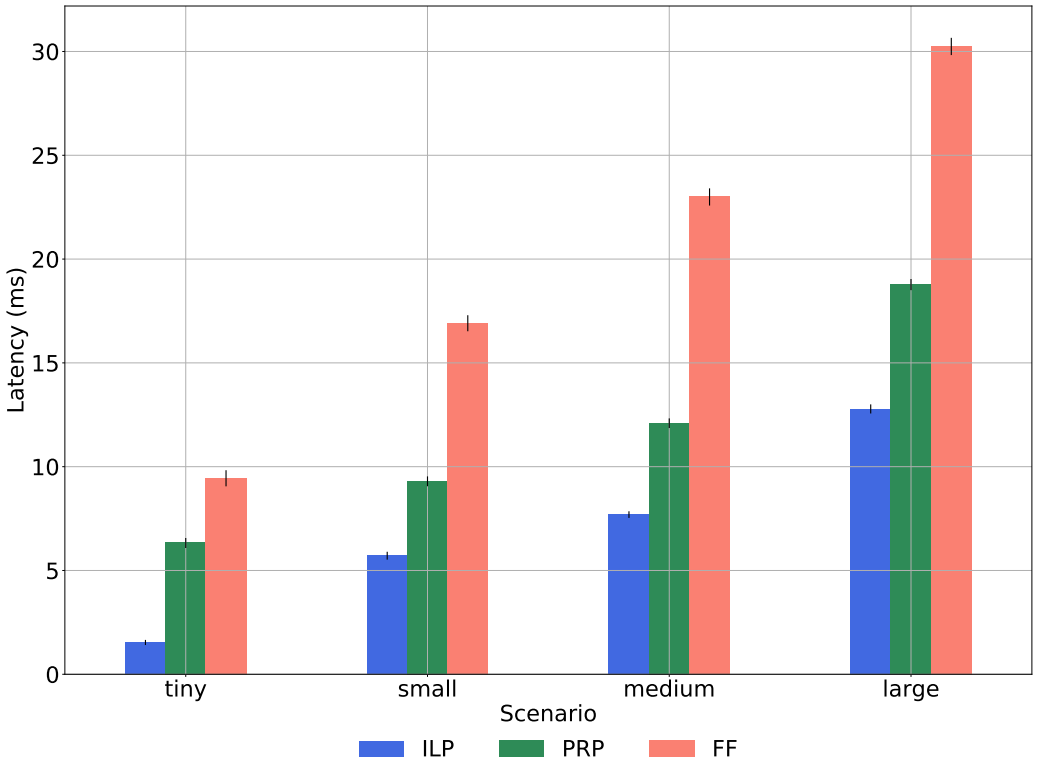


Figure 4.7: Total Latency by Scenarios - Asynchronous Response Applications

For the smallest load (i.e., *tiny* scenario) in both application types, PRP shows an exceeding latency of about 3 times the values obtained with the ILP placement, while FF shows an extra of about 5 times regarding the ILP latency. These discrepancies are reduced in higher load scenarios. Nevertheless, while the breach is reduced between ILP and PRP, it is more prominent for FF, especially in the Fire&Forget applications. For instance, in the large scenario for the Fire&Forget applications, PRP showed a slight increment (about 1.1 times) over ILP; while FF had a significantly larger increment of around 2 times over ILP. Simultaneously, for the Asynchronous Response applications in the *medium* scenario, FF shows an increase of 3 times respecting the ILP approach, while PRP only shows 1.5 times more latency.

Grouping the nodes in communities (PRP) showed better results than performing a linear search (FF). As the scenario got more complex, the improvement is more noticeable; in general, the results of PRP are closer to the optimal than FF. It is also noticeable that the Asynchronous Response applications (i.e., there is a response from two services to the source), shown in Figure 4.7 have a similar trend than the Fire&Forget applications. For the Fire&Forget applications, PRP shows results closer to ILP than for the Asynchronous Response applications, which might be caused by the extra messages sent to the source.

The following experiments were focused on the network transmission, including the application messages forwarded and the average network buffer occupancy (i.e., the average amount of messages kept in node's network buffers waiting for link availability). Figure 4.8 depicts the results for the Fire&Forget applications and Figure 4.9 does the same for the Asynchronous Response applications.

From these results, it is important to point out that the ILP method forwarded the lowest amount of messages, thus generating less traffic and, ultimately, less congestion. PRP values remained close to the theoretical optimum (ILP), especially for the larger scenarios in the Fire&Forget applications. As the load increases, so does the number of messages forwarded. Also, for the Asynchronous Response applications, there is a larger number of messages exchanged, which is caused by the two extra edges added to the application graph for this type of applications.

FF showed to be the least efficient method; the nodes with the lower amount of resources are selected first, saturating the network buffers as the load increases. For the application messages, since FF does not take into consideration the interaction between nodes and the nodes are selected in a RR fashion, modules from the same application can be placed in different nodes; thus, the amount of messages forwarded is larger. Furthermore, these different nodes can even be distant (i.e., several hops, higher propagation delay), since they are only organized by their resources; hence, the latency incurred by this mechanism is higher.

The Asynchronous Response applications also showed more messages transmitted, as expected since they have two services sending extra messages to the source. The trends in the results remained similar among the different types of applications for all the evaluated scenarios, which indicates that there is no

significant impact on varying the type of applications in the performance of the placement mechanisms regarding the network metrics.

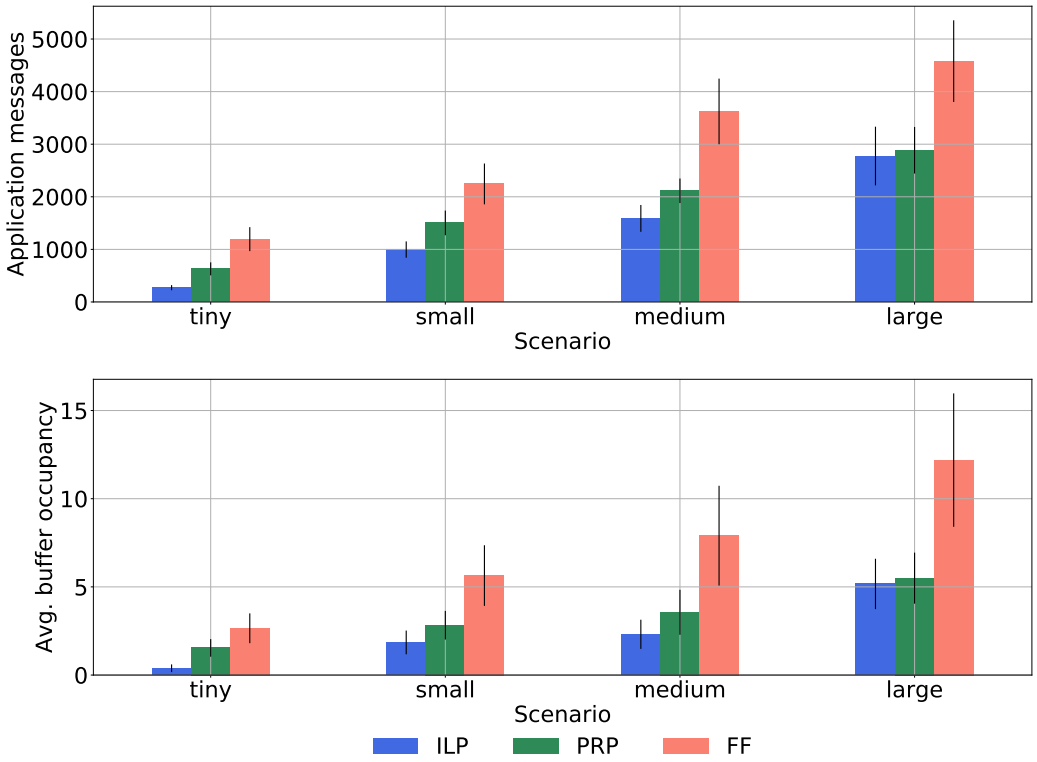


Figure 4.8: Network Transmission - Fire&Forget Applications

The next experiment was aimed at the resource usage. Figure 4.10 illustrates the results for the Fire&Forget applications and Figure 4.11 for the Asynchronous Response applications.

The first metric is the number of nodes used by each method; this means the number of nodes where the modules of the applications were placed. The second metric is the number of modules placed on the busiest node; this is the node where more modules were deployed. Since the placement was statically performed before the simulations, there are no variations in the results and thus no confidence intervals shown.

The ILP model led to the concentration of the placement of modules in the nodes closer to the request points (i.e., the GW), saturating these nodes, and therefore creating a new issue in the form of overloading of the nodes and the links connected to said nodes. As the load increases, the advantages of the optimization are missed by the overload created. On the other hand, PRP uses more nodes for the two smallest scenarios, thus having impact on the energy consumption. For the two larger scenarios, the number of nodes used tends to stabilize in all the placement approaches, and for the two types of applications, given that the increase of the load begins to saturate the network resources.

The next experiment depicts a comparison of the latency obtained by each application considering the amount of requests it has (i.e., popularity). The results

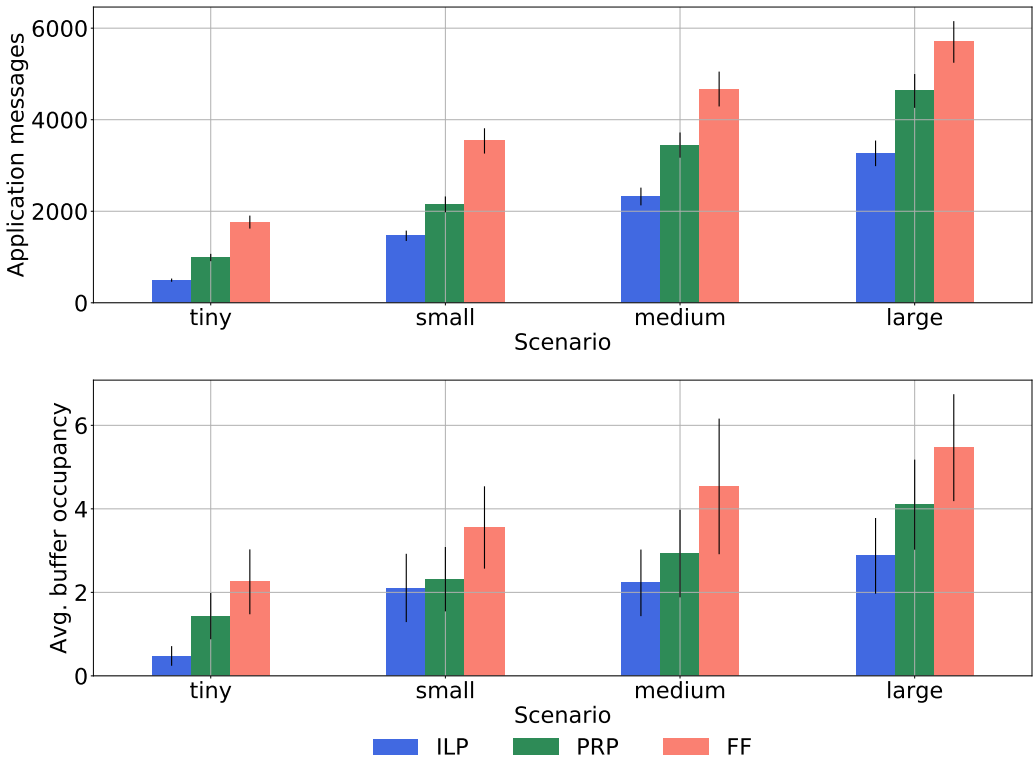


Figure 4.9: Network Transmission - Asynchronous Response Applications

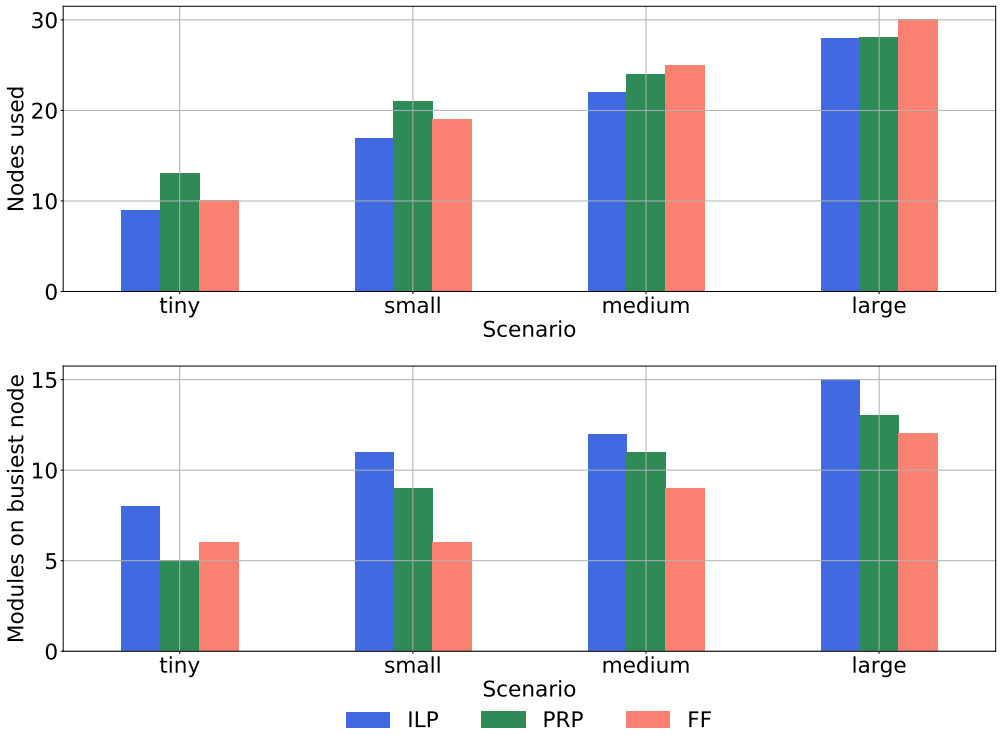


Figure 4.10: Module Placement Metrics - Fire&Forget Applications

are similar for the different scenarios, and to ease the readability only the results related to the large scenario are presented in this section, in Figure 4.12 for

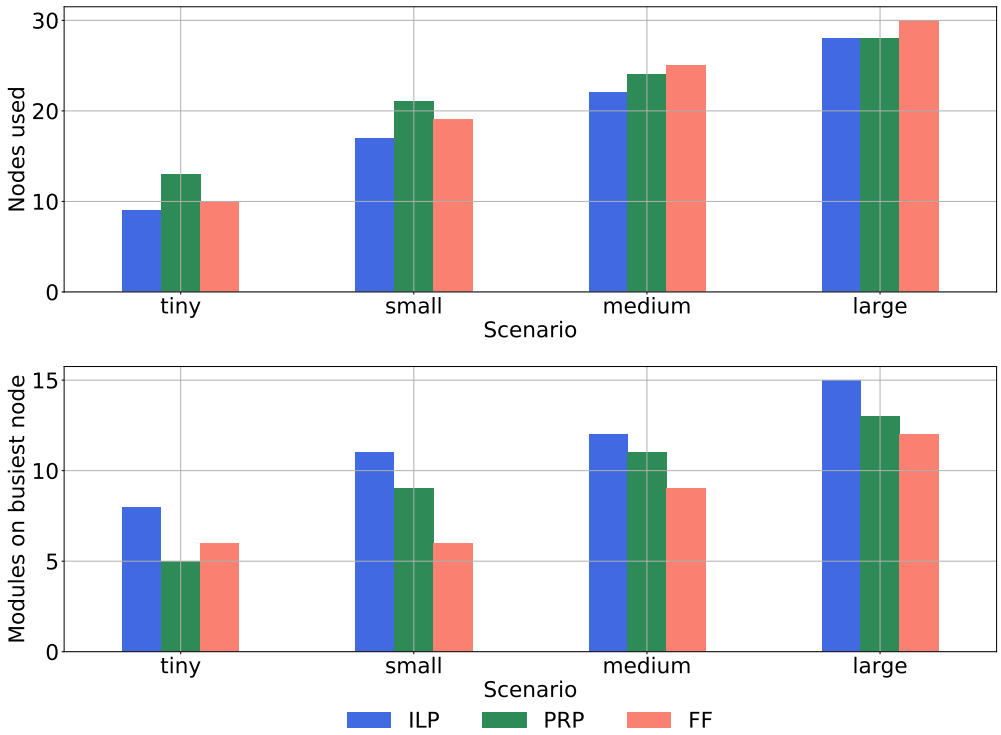


Figure 4.11: Module Placement Metrics - Asynchronous Response Applications

the Fire&Forget applications and in Figure 4.13 for the Asynchronous Response applications, respectively. Results for the rest of the scenarios are presented in Appendix A.

Overall, the ILP model obtained the best results, while FF got the worst. It is noticeable that as the application has more requests (i.e., more popularity), it shows lower latency for ILP and PRP. Furthermore, ILP and PRP results are relatively close, with ILP showing the lowest latency values. It is clear that there are three tiers regarding the latency response per application, being the lowest the one corresponding to the ILP model, the following to PRP, and the highest to FF. It is also important to point out that the most popular applications showed lower latency levels than less popular applications. The trend is maintained for all the scenarios and for both application types. These results confirm that while the overall latency was reduced, popular applications received a better treatment by the Orchestrator, thus the categorization of applications proved to be effective.

Finally, Table 4.4 shows the average execution time, in seconds, for the placement methods, by scenario. Since FF has the most straightforward logic, it has also the lowest execution times, followed closely by PRP. The times for ILP are significantly higher since this method evaluates all possible solutions in order to find the optimal result. The times obtained by the ILP model could not be suited for more complex and dense realistic scenarios.

In general, the ILP approach got the best results regarding latency, but the worst on execution time and in node overload. The latency values obtained with PRP

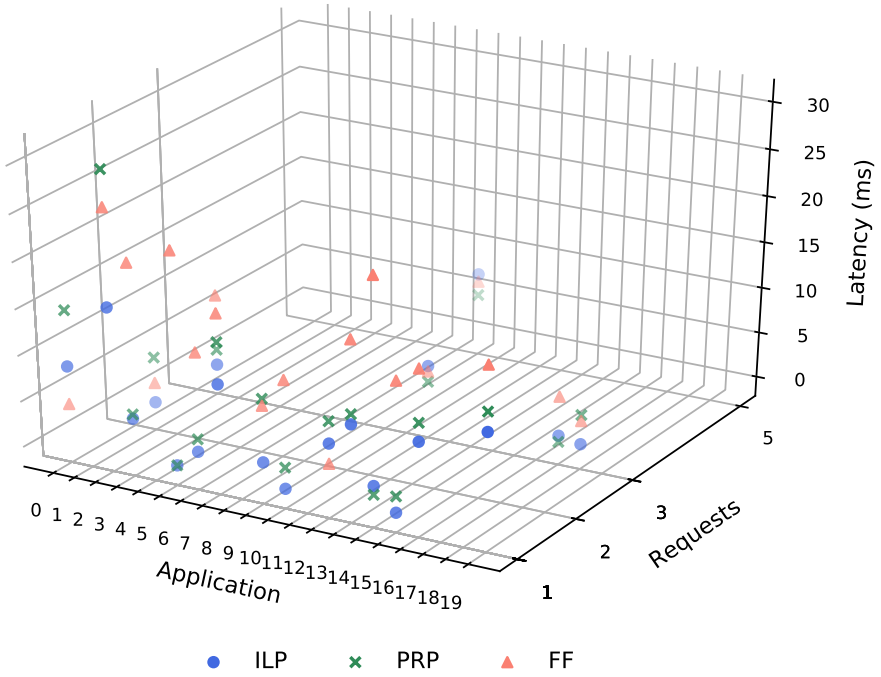


Figure 4.12: Latency by Application - Fire&Forget Applications - Large Scenario

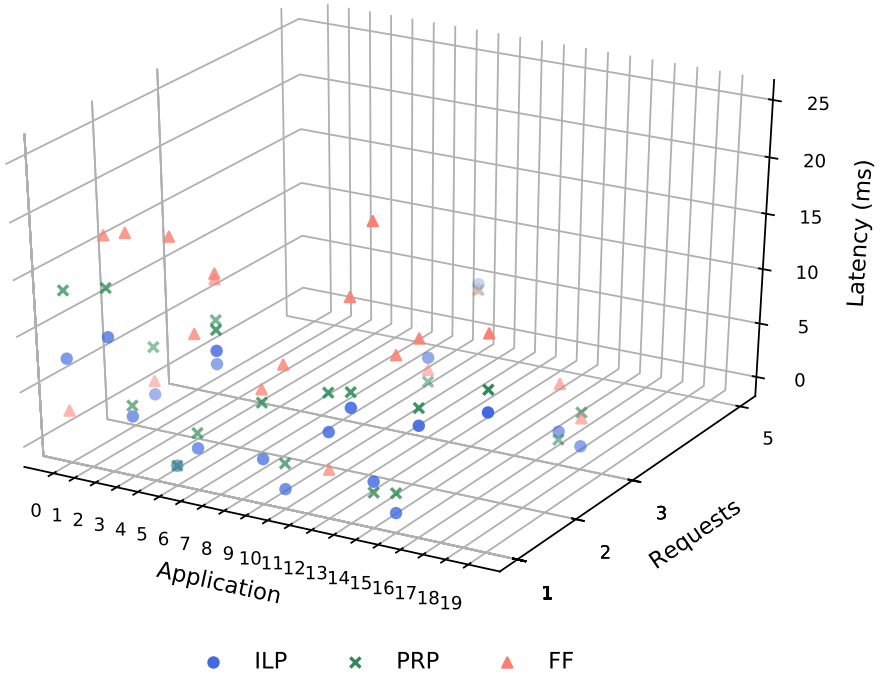


Figure 4.13: Latency by Application - Asynchronous Response Applications - Large Scenario

Table 4.4: Execution time (in seconds)

Scenario	**ILP**	**PRP**	**FF**
Tiny	19.191	0.017	0.005
Small	47.382	0.019	0.007
Medium	104.983	0.022	0.011
Large	313.588	0.031	0.0019

are close to the ones reported by ILP while getting significantly lower execution times. Furthermore, since the PageRank can be calculated dynamically, it is possible to apply this solution in case of variations in the network infrastructure and user requests. Finally, by creating communities, the load is spread among different nodes, which could lead to other issues such as higher energy consumption, that are not being considered in this work. This spread also means lower congestion levels in both the Fog nodes and communication links for PRP. The different types of applications did not show any significant difference among them, depicting similar behavior in the evaluations performed.

4.5 Chapter Summary

An ILP model for service placement, aimed at maximizing the placement of popular applications, while minimizing their latency is proposed. Moreover, a heuristic based on the PageRank algorithm, called Popularity Ranked Placement, is also introduced. PRP ranks the applications according to their requests as a measurement of their popularity, and also ranks the nodes in the network topology to create communities with the nodes with the highest transition probability; this way, the placement load is divided among the nodes within the community, avoiding congestion while also maintaining low latency levels. An expanding window controls the placement among the nodes in the community.

The performance of the ILP model and the PRP solution were tested using YAFS, and are also compared with the well known FF approach. Simulation results show that while the ILP model had the lowest latency for all the scenarios, it also had the highest concentration of placement in the same nodes, thus generating congestion in the processing nodes and for the communication links. PRP kept latency at low levels, close to the ILP results, but the load was balanced between the nodes in the communities. The use of popularity proved to be effective in the results, prioritizing applications with higher request numbers for both ILP and PRP. PRP showed significantly lower execution times than the ILP model, which makes it more suitable for more complex and dense scenarios, and is the option that should be deployed in practical realistic situations. The results did not show significant variations regarding the application types (i.e., Fire&Forget vs. Asynchronous Response applications).

The mechanisms presented in this chapter could be integrated in the *Service Or-*

chestrator architecture (depicted in Section 3.2.3) presented in Chapter 3, thus combining the different contributions from this work. The proposals presented in this chapter are partially reported in the following publications.

Description of the ILP model and the heuristic based on the PageRank algorithm, called PRP:

> Velasquez, K., Perez Abreu, D., Paquete, L., Curado, M., and Monteiro, E. (2020). A Rank-based Mechanism for Service Placement in the Fog. In *2020 IFIP Networking Conference (Networking)*, pages 64-72, Paris, France. IEEE.

Analysis of different Fog simulators to select the proper simulation tool for the validation process:

> Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2020). A comparative analysis of simulators for the Cloud to Fog continuum. *Simulation Modelling Practice and Theory*, 101(1):102029. Elsevier.

Chapter 5

Service Placement via Application Profiling

Contents

The mechanisms presented in the previous chapter have a static behaviour, in the sense that the placement is carried out once and no new requests or placements are performed. The ILP model provides a theoretical threshold of minimized latency, but this type of solutions is not always well-suited for dynamic, more realistic scenarios given their time and resource consumption to reach a solution [Lee and El-Sharkawi, 2008]. However, the PRP heuristic could be adapted for dynamic scenarios to further evaluate its performance regarding latency reduction.

One possibility to add dynamism to the simulated scenario is to apply a time window approach [Wang et al., 2017; Zhang et al., 2012], in which the time is divided into *slots* and for each time slot the requests are evaluated and placed according to the updated network conditions; this approach is the same as described in Chapter 3 for random start and prediction start. The Service Orchestrator collects statistics from the *Information Collection* module about the status of the network, the amount of requests and their locations, and based on this information performs the placement decisions for each time window.

By using time windows it is possible to model changing conditions in the applications, for instance those related to their popularity. The popularity of the applications could vary over time, increasing or decreasing. This leads to the possibility to categorize the applications according to their popularity over time, and using said categorization for the placement process. This chapter proposes a profiling system based on application popularity for service placement in dynamic scenarios.

The already introduced PRP is compared with an additional heuristic, based on a Genetic Algorithm. The GA proposed combines two objectives using weighted sums for the fitness function. Validation of both solutions under dynamic conditions via simulations is provided.

5.1 Profiling Applications according to their Popularity

Placement in Cloud/Fog environments can have a crucial impact on reducing the latency. Using characteristics from the applications has proven to be useful for the placement process, prioritizing the applications according to a given criterion such as popularity, as seen on the previous chapter, since it might favor a majority of users.

The Service Orchestrator has to monitor the deployment infrastructure (i.e., nodes, links, services, and users) to make smart informed decisions that will affect the system behavior. One of the modules of the Orchestrator, presented in Chapter 3, is dedicated to the execution of planning and placement mechanisms in charge of the selection of optimal location to deploy the application components. To improve the performance of the placement mechanisms, it is essential to provide them with knowledge not only about the substrate network, but also of the entire system including users and applications. The use of a

behavior profiling approach for this goal would provide to the Orchestrator the possibility to respond to the changing conditions in the environment.

The devices in the Fog are notoriously resource constrained in comparison with the Cloud [Skarlat et al., 2017a]. However, in these tiers final users will benefit from lower latency. Thus, it is critical to determine the proper location in which to deploy the services. For this particular scenario, having knowledge of the behavior of application components would benefit the decision making for the Orchestrator. This knowledge has to cover past behavior and current conditions to be able to react to dynamic situations such as traffic load variations.

Although the use of profiling has been explored in the context of Cloud, the adoption of the microservices architecture and the constraints inherent to the Fog bring new challenges. To the best of our knowledge, the use of behavior profiling in Cloud to IoT scenarios has not been successfully exploited so far from the academic point of view because of the complexity of the landscape and the lack of datasets available for study to apply data analytics techniques. Nevertheless, there have been some recent efforts to gather real-time data in this kind of environment that are still in an early stage [Khan et al., 2018]. There have also been efforts in the categorization of applications in Fog environments by using their requirements (i.e., latency, bandwidth, processing, and memory) to group them [Ding and Janssen, 2018; Siddiqi et al., 2019]. Such a categorization is presented in Table 5.1, listing applications typically used in Fog environments and their resource requirements, as established for 5G environments. Applications are grouped by domains, displaying their resource demands estimate.

Table 5.1: Application Profiling by Domain

Domain	**Application**	**Latency**	**Bandwidth**	**Processing**	**Memory**
Optimized broadband	Video UHD/3D	Low	Ultra high	High	High
	Virtual reality	Low	Ultra high	High	High
	Augmented reality	Low	Ultra high	High	High
	Cloud games	Low	Ultra high	High	High
Ultra-reliable communications	Industrial automation	Low/Ultra-low	High	Medium/Low	Low
	Critical mission	Low/Ultra-low	High	Medium/Low	Low
	Driver-less car	Low/Ultra-low	High	Medium/Low	Low
	Health care	Low/Ultra-low	Medium/High	Medium/Low	Low
Massive machine communications	Smart homes	Medium/High	Low	Low	Low
	Smart offices	Medium/High	Low	Low	Low
	Sensor networks	Medium/High	Low	Low	Low

Given the lack of datasets available regarding the behavior of Fog applications, a possibility not yet explored is to profile the applications according to their popularity measured by their requests, as listed in Table 5.2. Using the popularity of the applications to prioritize their placement has proven to be effective, as seen on the previous chapter.

For this work, in each time window the amount of requests and their originating locations (i.e., source gateway) were modified to model the variations in the request rate at different times. Four patterns, listed in Table 5.2, were modelled for the request rate. These changes will influence not only the amount of application instances to place, but also the resource consumption in the network,

Table 5.2: Application Profiling by Popularity

Profile	**Description**
Fixed	The same amount of requests is maintained
Mixed	The amount of requests oscillates, increasing or decreasing in each time window according to a rate
Up	The amount of requests is increased according to a rate
Down	The amount of requests is decreased according to a rate

including the node resources needed to place the services and the amount of traffic that affects the communication among the services. The different popularity patterns will allow to model a more realistic environment in which the request rates change. Again, as the application shows a higher popularity value, it should be benefited with a placement that favors it by lowering its perceived latency.

Figure 5.1 describes the overall placement process proposed in this work, including the use of information about application behavior.

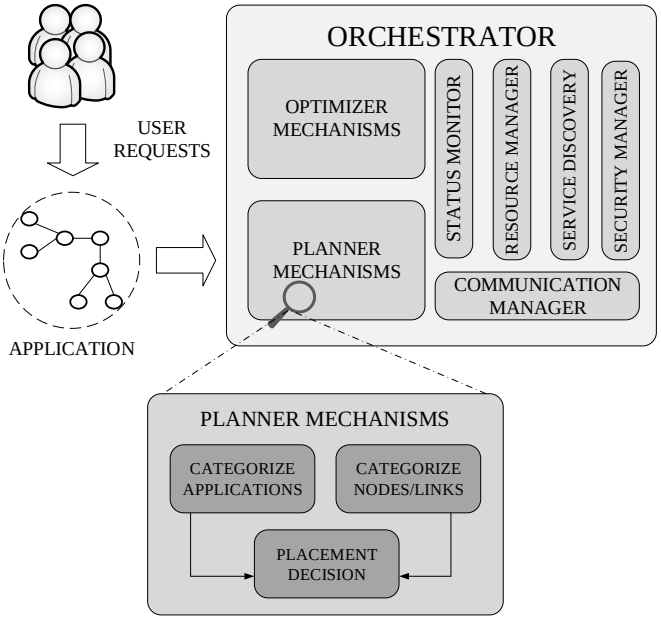


Figure 5.1: Planner Mechanisms in the Service Orchestrator

Users place their requests on different applications. The Orchestrator (based on the architecture presented in Chapter 3) collects this information and uses it to generate smart informed decisions. Particularly, the *Planner Mechanisms*, in charge of determining the location to deploy the applications, apply a categorization based on the popularity of the applications, and also a categorization of the substrate components of the deployment infrastructure (i.e., node resources and propagation delay) to make a placement decision. The Orchestrator will re-evaluate the placement decisions over time using a time window to update the values regarding the popularity of the applications, as well as the revised information about the deployment infrastructure.

To determine if the profiling based on application popularity impacts the final performance of the applications, particularly the latency perceived by end-users, this chapter uses the heuristic presented in the previous chapter (PRP) adapted for dynamic environments, and an additional heuristic based on Genetic Algorithms. The newly proposed heuristic is described in the following section.

5.2 A Heuristic based on Genetic Algorithms

Genetic Algorithms are a family of computational models inspired by evolution. These algorithms model the problem solution as a *chromosome*-like data structure and apply them simple combinatory operators to try to keep critical information while enhancing the final solution [Whitley, 1994]. The main objective is to move forward a *chromosome* from one *generation* to the next using genetics-inspired operators of crossover, mutations, and inversion [Mitchell, 1998]. Each chromosome consists of *genes* (i.e., bits), the selection operator chooses the chromosomes in the population that will be reproduced in the next generation, with the goal of producing offspring from the fittest chromosomes. The *fitness* of the chromosomes is defined by a *fitness function*, which is based on the objectives to optimize and defines the quality of the solution. Common selection operators include *mutation* (i.e., random change in one gene or group of genes) and *crossover* (i.e., merge of two individuals). Solutions in GA are usually represented in n-dimensional arrays that represent the chromosome, where each element of the array is called *gene* [Gen and Cheng, 2000].

Genetic Algorithms are often viewed as function optimizers and have previously been used to solve service placement problems [Moens et al., 2014; Skarlat et al., 2017a; Khebbache et al., 2018; Guerrero et al., 2019b]. However, popularity has not been used as a metric in GAs for the placement of applications. From the different variants of GAs and evolutionary algorithms used for service placement in the Fog, the Weighted Sum Genetic Algorithm (WSGA) showed a good compromise regarding the execution time and fitness values obtained, while also exhibiting lower convergence times [Guerrero et al., 2019b] which led to the selection of this algorithm. The WSGA is described in the following section.

5.2.1 Weighted Sum Genetic Algorithm

The Weighted Sum Genetic Algorithm consists on a transformation that normalizes the values of multiple objective functions to the unit interval, and weights them to obtain a final result. The calculation is shown in Equation 5.1, where ω_i is the scaling factor, θ_i is the weight, and X_i is the value of the objective function.

$$\sum_{i \in numObj} \omega_i \times \theta_i \times X_i \tag{5.1}$$

Table 5.3 lists the parameters and variables used by the GA-based heuristic implemented in this work. The resource unit is used for the variables ω_s and Ω_n, where a resource unit reflects the resources (i.e., CPU, memory, storage) of node $n \in N$. The cost matrix contains the cost (in terms of latency) to reach a node from a gateway. $C_{n,gw}$ equals the propagation delay of the shortest path that connects $n \in N$ to $gw \in GW$. The instance matrix identifies the services that compose an application. $I_{a,s}$ equals 1 if service $s \in S$ belongs to application $a \in A$, and 0 otherwise. Finally, the placement matrix relates the request per application and the node that is selected as the location for the services. $P_{s,n}^{a,r} = 1$ if service $s \in S$ is located in node $n \in N$ to satisfy request $r \in R$ for application $a \in A$, and 0 otherwise.

Table 5.3: Parameters and Variables for the GA Heuristic

Parameters	
Parameter	**Description**
S	Set of services to be placed
N	Set of nodes where the service can be executed
GW	Set of gateways
A	Set of applications. An application is composed by a set of services
R	Set of requests for all the applications
Q_a	Sum of requests for $a \in A$
Ω_n	Resource capacity for $n \in N$
ω_s	Resource requirement for $s \in S$
C	Cost matrix. An $\|N\| \times \|GW\|$ matrix
I	Instance matrix. An $\|A\| \times \|S\|$ matrix
Variables	
Variable	**Description**
P	Placement matrix. An $\|A\| \times \|R\| \times \|S\| \times \|N\|$ matrix

For the GA implemented in this work, *individuals* are modeled using the placement matrix P. For a solution to be valid, the *genes* or bits in P must reflect valid requests according to R, the proper services $s \in S$ that compose $a \in A$ according to the information from the instance matrix I, and must comply with feasibility restrictions of node capacities imposed by Ω.

The mutation is carried out by keeping fixed the information regarding the requests ($r \in R$), applications ($a \in A$), and services ($s \in S$), and altering the information of the nodes $n \in N$. This way, the same services placed in a previous solution are moved to a different location generating a new solution. The selection operator used is a *binary tournament*, using two parent solutions to mix. A random amount of services from the first parent and their locations are combined with the rest of the services from the second parent and their locations. If a solution is invalid, a fitness value of *infinite* is assigned, so that

it is not passed to the next generation.

Algorithm 5.1 shows the basic procedure of this GA. It starts at line 1 by randomly generating *popSize* (population size) solutions to create the first generation of solutions. The objective function values are calculated for the first generation (line 2) and the fitness value is calculated (line 3). Then, at line 4, for each generation, it initializes the offspring population as the empty set, as seen on line 5.

Algorithm 5.1: Weighted Sum Genetic Algorithm

```
   Result: solution
1  P_t ← generateRandomPopulation(popSize)
2  objValues ← getObjValues(Pt)
3  fitness ← ws(objValues, ω, θ)
4  foreach i in generations do
5  |  P_off ← ∅
6  |  foreach j in popSize do
7  |  |  parent1 ← selectParent(P_t, fitness)
8  |  |  parent2 ← selectParent(P_t, fitness)
9  |  |  child1,child2 ← crossover(parent1, parent2)
10 |  |  if random() ≤ mutationProb then
11 |  |  |  mutate(child1, child2, numGenes)
12 |  |  end
13 |  |  P_off ← P_off ∪ child1, child2
14 |  end
15 |  objValues ← getObjValues(P_off)
16 |  fitnessOff ← ws(objValues, ω, θ)
17 |  fitness ← fitness ∪ fitnessOff
18 |  P_off ← P_off ∪ P_t
19 |  P_off ← order(P_off, fitness)
20 |  P_t ← P_off[1..popSize]
21 end
22 solution ← min (P_t, fitness)
23 return solution
```

On each generation, and for each individual in the population, the parents are selected from a binary tournament selection operator (line 7), this is, choosing the individual with the best fitness from a subset of the population. The process is repeated for the second parent (line 8). After selecting the parents, the children are created by applying a crossover operator (line 9), and then mutating them with a probability of *mutationProb*, as seen in line 11. The crossover consists on mixing a portion from the first solution (i.e., parent1) with the remaining portion of the second solution (i.e., parent2). The children solutions are mutated with an uniformily distributed probability (25% in this case), as shown in line 10. The mutation selects a node from the solution and changes it for a different node (i.e., for a given service of a given application and a given

request, the location is updated). The amount of nodes to update (i.e., mutate) in the children solutions represents 10% of the total of services in the solution, indicated by *numGenes* parameter in line 11.

In line 13, the newly created children join the new population (offspring). The fitness value is calculated in line 15, as described by Equation 5.2 and Equation 5.3, illustrated in the following section. Both objectives are combined with Equation 5.1 in line 16. All the fitness values are combined in line 17, and all the population members are joined in line 18. All the elements of the population (including the newly created children) are sorted by their fitness value, in line 19. Only the best *popSize* elements of the population will survive for the next generation (line 20). After iterating for *generation* times, the returned solution will be the one with best fitness value, shown in line 23. The fitness function used in this algorithm is described in the following section.

5.2.2 Calculating the Fitness Value

Latency and resource usage are the two different objectives that are combined in the fitness function, using Equation 5.1. Latency is calculated using the propagation delay of the links in the network topology (as a latency source impacting placement decisions, see Section 2.1.1), and the resource usage is calculated using the YAFS resource unit [Lera et al., 2019b]. The goal is minimizing both objectives prioritizing the latency, using the weight factor (i.e., θ) for this purpose. Since the main goal in this work is to minimize the latency, this objective value received a weight of 0.9, and the resource usage only 0.1. The resource usage aims at minimizing the amount of free resources on the network, thus maximizing the amount of accepted applications, as with the ILP model discussed in the previous chapter.

To evaluate the latency, information about the propagation delay (i.e., matrix C) is added for the services that belong to application $a \in A$ (i.e., matrix I) of the placed applications (i.e., matrix P). This value is weighted according to the popularity of the application, as stated by Equation 5.2; where Q_a is the amount of requests for application $a \in A$ measuring its popularity.

$$\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} \sum_{n \in N} \sum_{gw \in GW} \frac{1}{Q_a} \times \left[P_{s,n}^{a,r} \times I_{a,s} \times C_{n,gw} \right] \tag{5.2}$$

Regarding the second objective function, resource usage, it is calculated by adding the resources used by each node, as shown in Equation 5.3, where ω_s is the amount of resources required by service $s \in S$, and Ω_n denotes the amount of resources of node $n \in N$. Thus, Equation 5.3 depicts the free resources of nodes $n \in N$.

$$1 - \frac{\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} \sum_{n \in N} \left[P_{s,n}^{a,r} \times I_{a,s} \times \omega_s \right]}{\sum_{n \in N} \Omega_n} \tag{5.3}$$

Both objectives are combined in Equation 5.1, and minimized in Algorithm 5.1 in line 22.

Simulation experiments were carried out to validate this proposal; the experimental setup is described in the following section.

5.3 Experimental Evaluation

The evaluation was performed via simulation using YAFS, given its strong support for Fog features [Perez Abreu et al., 2020]. The experiments were conducted using a PC with 32GB 2400MHz DDR4 RAM and 2.80GHz Intel Core i7-7700HQ with 4 cores and 8 threads (2 threads per core) processor. The operating system used was Windows 10 Pro (Build 18363). The version of Python used for YAFS was 2.7.16.

The entire evaluation workflow is depicted in Figure 5.2. Once the configuration parameters are defined, three files are created: one for the definition of the topology, one for the definition of the applications (generated using the *Growing Network* approach), and one for the definition of the users, (i.e., network load), which represents the number of requests for each applications and the originating gateways.

These files are combined in the catalog, describing service requirements, application popularity, and other information needed for the simulation. With the catalog ready, the mechanisms are executed, indicating the nodes where the services are going to be placed, and this information is provided to YAFS so the simulations can take place. The process is repeated for each time window, updating the catalog in each iteration, thus generating new placement decisions. With the raw data reported by YAFS, the final plots can be created.

A graph was generated for the network topology using the complex network theory, following a Barabasi-Albert network model [Jalili and Perc, 2017]. One hundred (100) Fog nodes were created, and an additional node represents the Cloud, for a total of one hundred one (101) nodes in the topology. The Cloud node is the one connected to the Fog nodes with the highest betweenness centrality in the graph. In contrast, the Fog nodes with lowest betweenness centrality in the graph were designated as GWs, representing the nodes at the edge of the network.

The applications were randomly generated following the *Growing Network* graph structure, in which the vertices are added one by one with an edge to the last added vertex [Yao et al., 2014]. Finally, two vertices (except the source) are randomly selected to generate an information flow towards the source vertex. This allows modeling applications that collect data and send an automated

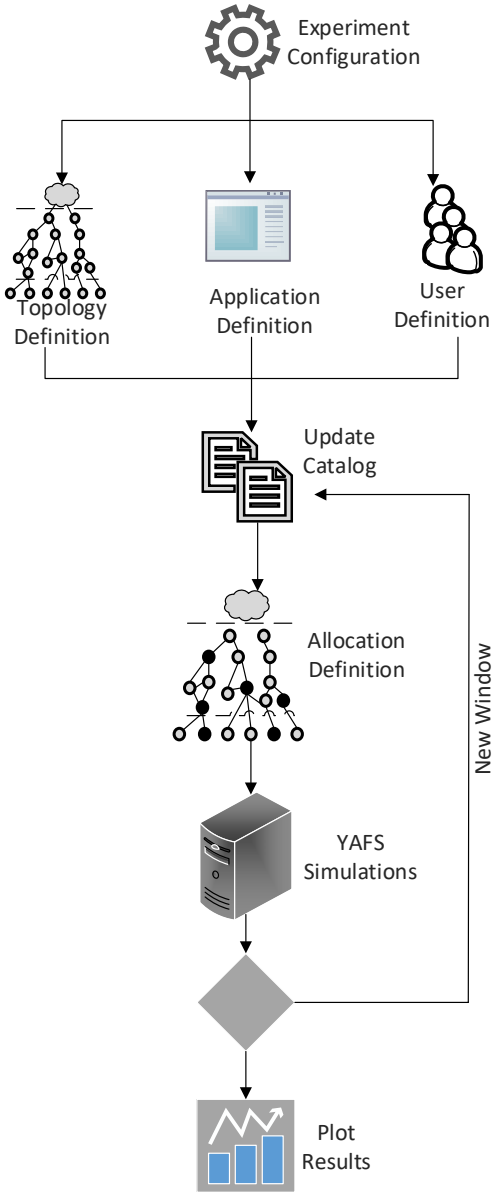


Figure 5.2: Evaluation Workflow for Dynamic Scenarios

asynchronous answer (e.g., eHealth, augmented reality).

Regarding the network load, three different scenarios were modelled, in order to evaluate the performance of the placement mechanisms with varying conditions: (1) *small*: 5 different applications; (2) *medium*: 10 different applications, and; (3) *large*: 15 different applications. Each application has at least one request, and follows one of the four application profiles described in Section 5.1. The number of requests was determined using a uniform distribution. Ten (10) time windows were used for each simulation, with a duration of 10000 time units. All the scenario setup, the source code, and additional material (i.e., plots, charts) are available via a GitLab repository [Velasquez et al., 2020].

The three different network loads were used to determine the amount of gen-

erations needed to converge to a solution. Figures 5.3, 5.4, and 5.5 show the evolution of the fitness value for the three different loads.

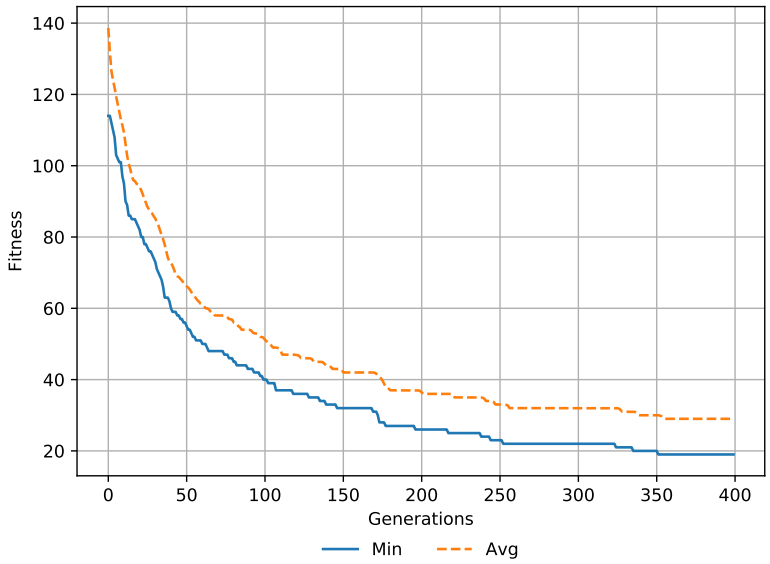


Figure 5.3: Fitness - Small Scenario

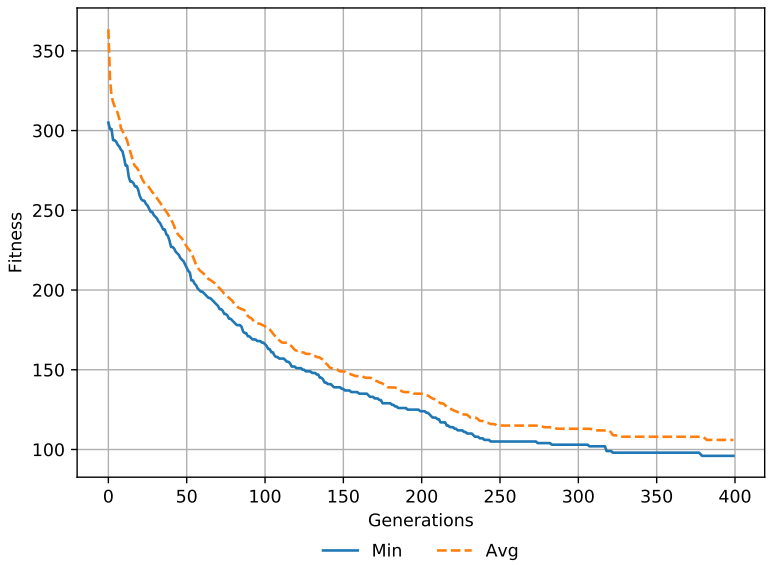


Figure 5.4: Fitness - Medium Scenario

It is noticeable that for all cases, around 250 generations are needed to converge at a relatively stable fitness value, with a slight improvement until reaching generation 400. The plots also show that as the load in the scenario grows, so does the fitness values obtained. This is an expected behavior since that by increasing the load so does the amount of resources used and the traffic in the network, which directly impacts the latency.

The configuration parameters for the experiments are summarized in Table 5.4, for the network links, Fog nodes, GWs, applications and services.

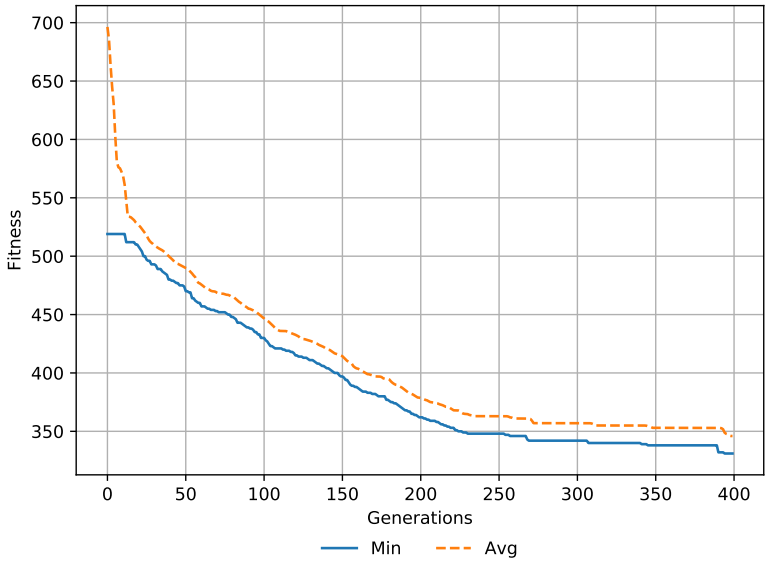


Figure 5.5: Fitness - Large Scenario

Table 5.4: Parameters Values for Dynamic Experiments

	Parameter	Value (min - max)
Network	Propagation Delay (ms)	2 - 10
	Bandwidth (bytes/ms)	75000
Fog	Resources (units)	10 - 25
	Speed (instr/ms)	500 - 1000
GW	Request rate (1/ms)	1/1000 - 1/200
	Popularity (prob)	0.25
Application	Services (number)	2 - 8
	Resources (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000
Genetic algorithm	Population size	100
	Generations	400
	Mutation probability	0.25
	Num. indiv. to mutate	10% total services
	Tournament size	2

Values similar to these were used in previous work ([Lera et al., 2019a; Guerrero et al., 2019b]) and in the previous chapter. The requirements for each application service are measured using the YAFS resource unit [Lera et al., 2019b], which is a vector that contains the capacity of different computational elements (e.g., memory, CPU, hard disk).

The two proposed mechanisms, PRP and GA, are compared with the well known FF algorithm, as it was done in similar works for evaluation purposes [Skarlat et al., 2017a,b], and also in the previous chapter. In the case of the FF al-

gorithm, the nodes were organized from lowest to highest according to their available resources. This way the nodes with less resources are prioritized, these nodes usually correspond with the nodes deployed at the edge of the network. 30 simulations were executed to mitigate the statistical error, including 95% confidence intervals in the plots. The simulation results are presented in the next section.

5.4 Results and Analysis

The performance of PRP and the genetic approach, GA, are evaluated in this section. Figure 5.6 shows the total latency by scenarios. PRP always reported the lowest latency, followed by GA and FF.

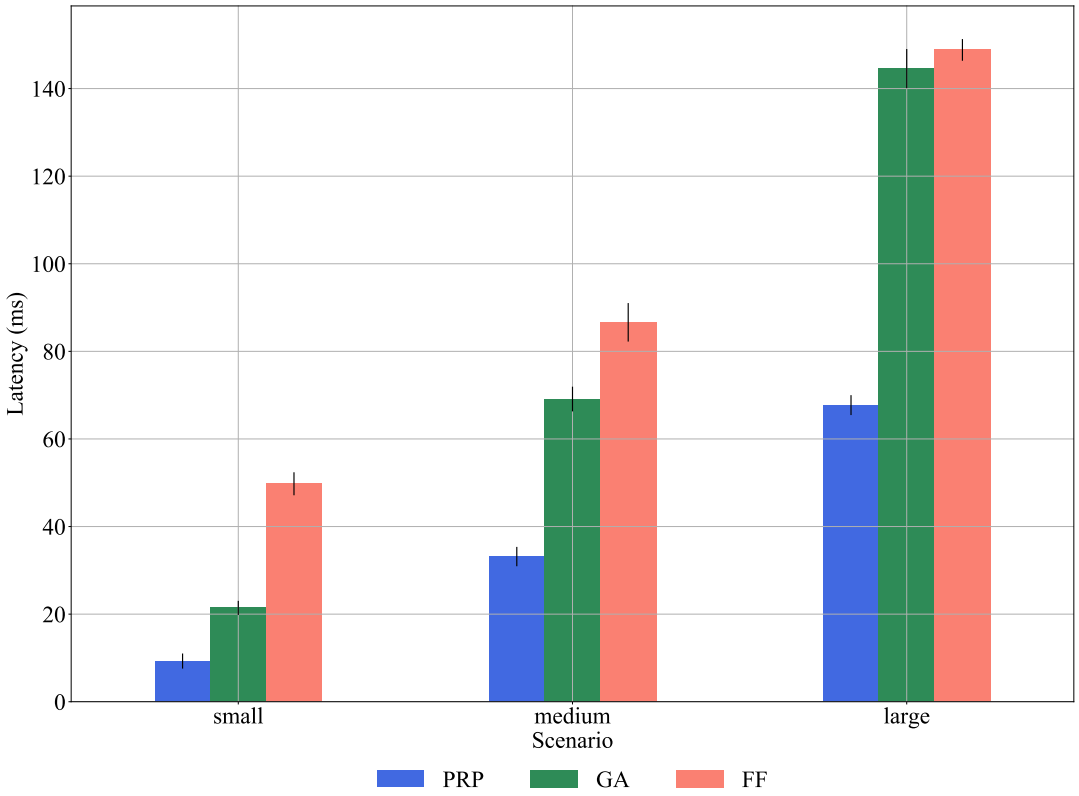


Figure 5.6: Total Latency by Scenarios

As the load grows, so does the latency for all the mechanisms, as expected. The traffic increases, saturating the nodes and communication links, influencing the overall latency of the system. For the smallest load (i.e., *small* scenario), GA showed an exceeding latency of twice the values reported by PRP, while FF showed a surmount of around 5 times over PRP. This breach shrinks as the load grows, as seen in the *large* scenario, where GA is only slightly better than FF. This is because as the load grows, the feasibility condition to validate solutions generated by the GA (i.e., not exceeding the capacities of the nodes) was more difficult to reach via the mutations introduced by the algorithm, generating solutions with elevated fitness values that were discarded for the following

generations, thus evolving slower. More generations are going to be needed as the load grows to attain better results, increasing also the execution time.

The following plots also show the latency, but with a finer granularity level. The latency is shown by mechanism, by scenario, and by application. The mean value of the boxplot determines the average latency of the application, while the boxplot itself reflects the variation of the latency values, i.e., the jitter experienced by the application, that can be calculated due to the dynamism in the scenario. Different colors are used to code the profile to which the application belongs to (see Section 5.1), and finally the dashed line shows the average popularity (i.e., amount of requests) of the application during the entire simulation (i.e., among all the time windows).

Figures 5.7, 5.8, and 5.9 show the results for the small scenario for PRP, GA, and FF, respectively.

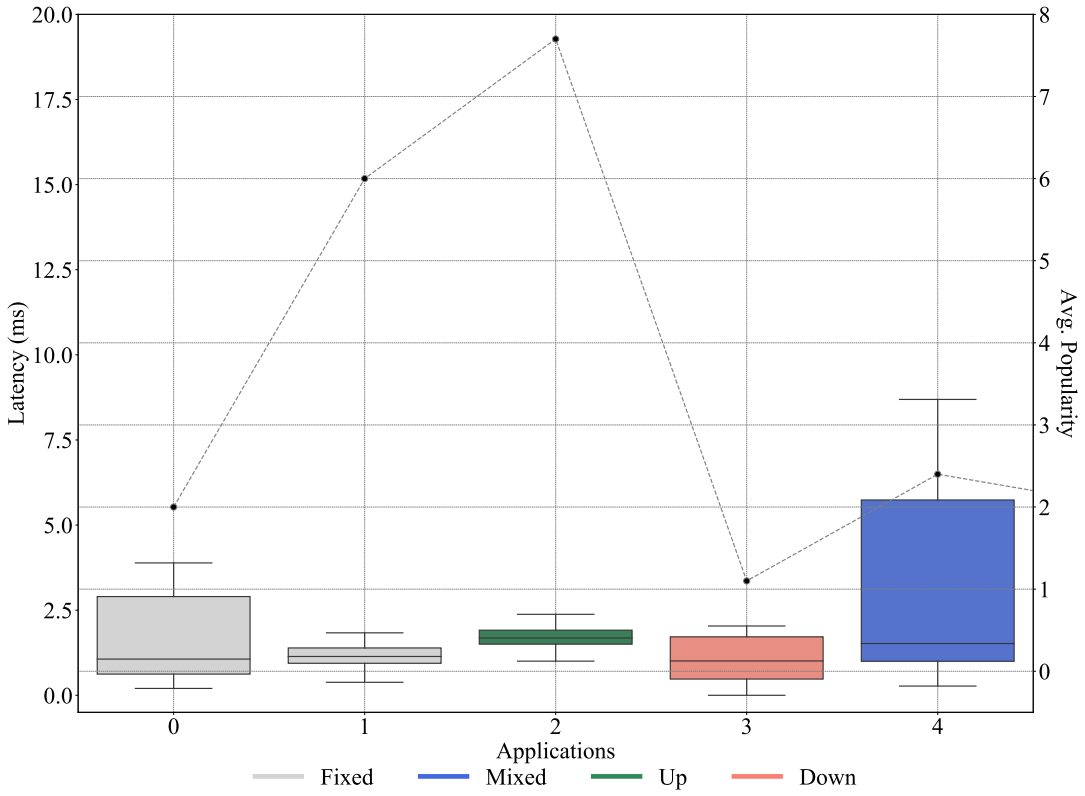


Figure 5.7: Latency and Jitter per Application - PRP - Small Scenario

The first observation that arises is that PRP showed the lowest latency values, followed by GA and FF, as it was depicted in Figure 5.6. On the other hand, the smallest jitter was shown by GA, followed closely by PRP and then by FF. However, for PRP, it is noticeable that with higher popularity, the latency variation, i.e., jitter, is lower, giving a clearer advantage to the most popular applications. This means that PRP showed a better treatment of the applications according to their profile. In the case of FF, not only the latency and jitter are very superior to the two heuristics, but also there is no differentiation in the treatment of the applications regarding their popularity.

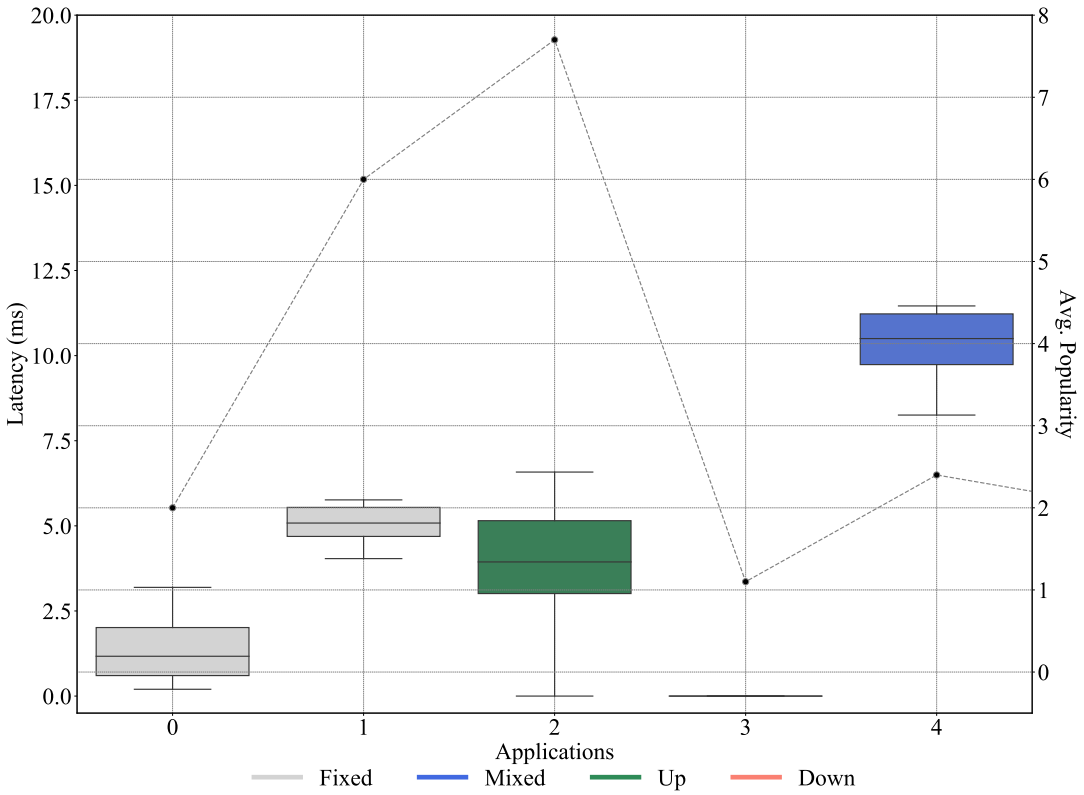


Figure 5.8: Latency and Jitter per Application - GA - Small Scenario

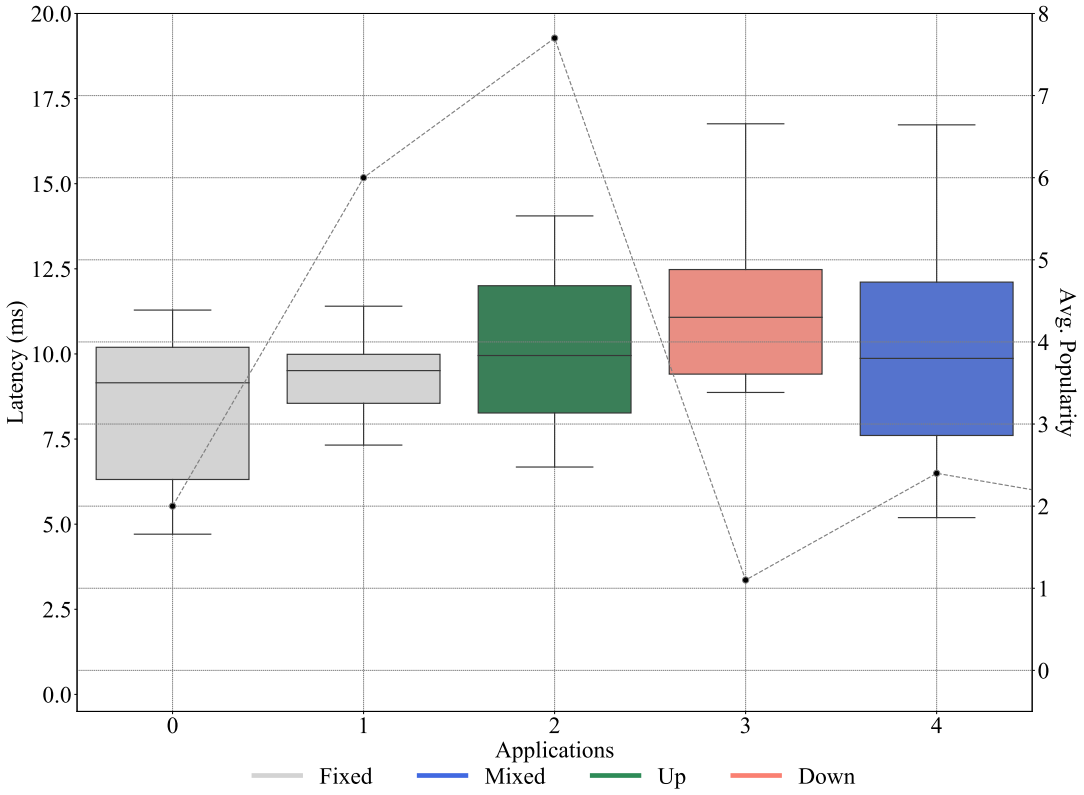


Figure 5.9: Latency and Jitter per Application - FF - Small Scenario

Figures 5.10, 5.11, and 5.12 show the results for the medium scenario for PRP, GA, and FF respectively. For PRP, it is noticeable that applications with a fixed popularity value have different performance when their popularity is lower (see Application 0) or higher (see Application 1), but for applications with an increasing popularity profile (see Applications 2 and 7) both the latency and jitter are lower.

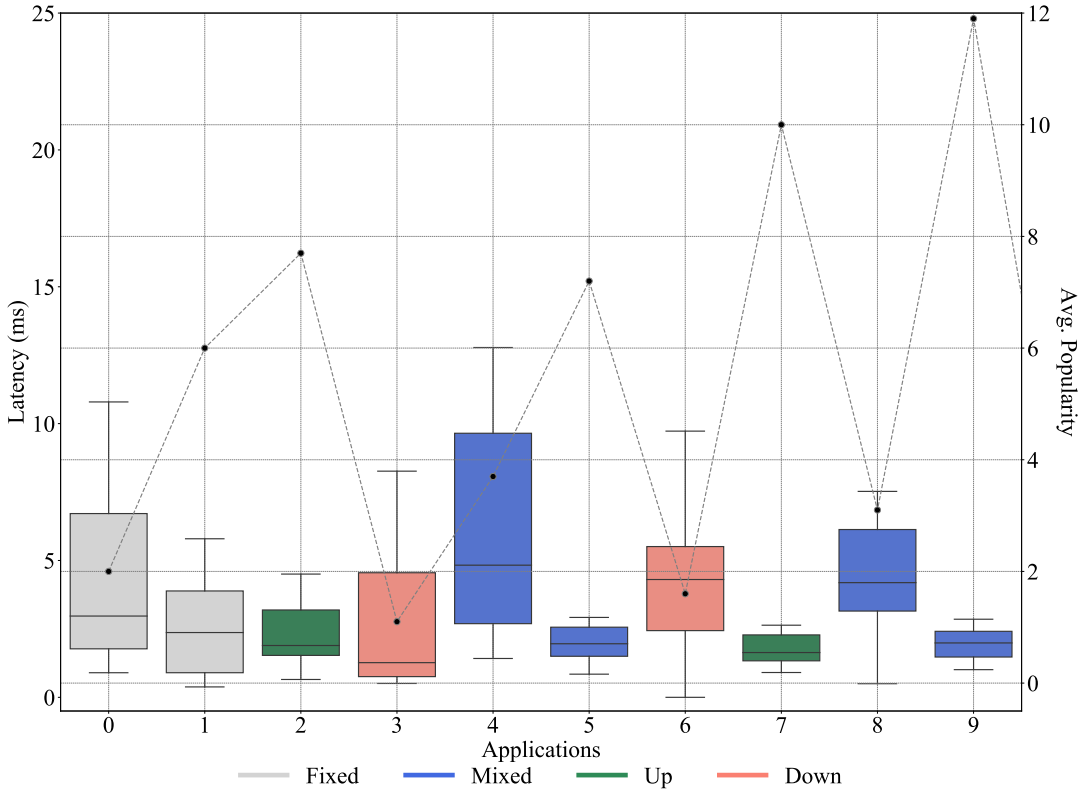


Figure 5.10: Latency and Jitter per Application - PRP - Medium Scenario

For the applications that have oscillating popularity values the jitter is more erratic, since for some time windows they were gaining popularity and for others they were losing popularity, thus having less stable results. Finally, for applications that constantly lost popularity (see Applications 3 and 6) the jitter is higher. The overall latency was also the lowest among all the mechanisms evaluated.

The advantages of having a lower jitter as the application has more popularity are less evident with the GA, although the mixed behavior for oscillating applications is also present. In the case of applications constantly losing popularity (see Application 3 and Application 6) the mean latency values are lower, this might be caused by a high popularity in early time windows (which favours them in the beginning of the simulation) that was constantly decreasing. This behaviour affects the jitter of these applications, as seen in the boxplots.

GA outperformed FF in both latency and jitter. FF showed the highest latency and jitter values, as well as less discrepancies among applications regarding their popularity levels and profiles.

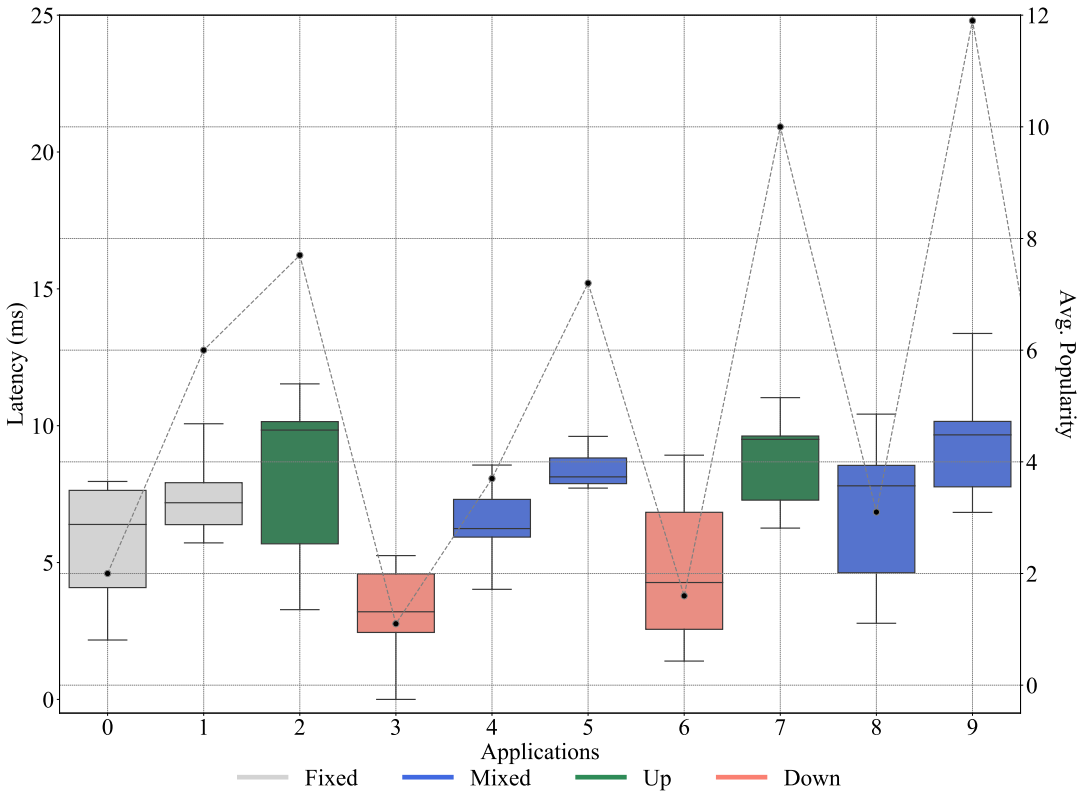


Figure 5.11: Latency and Jitter per Application - GA - Medium Scenario

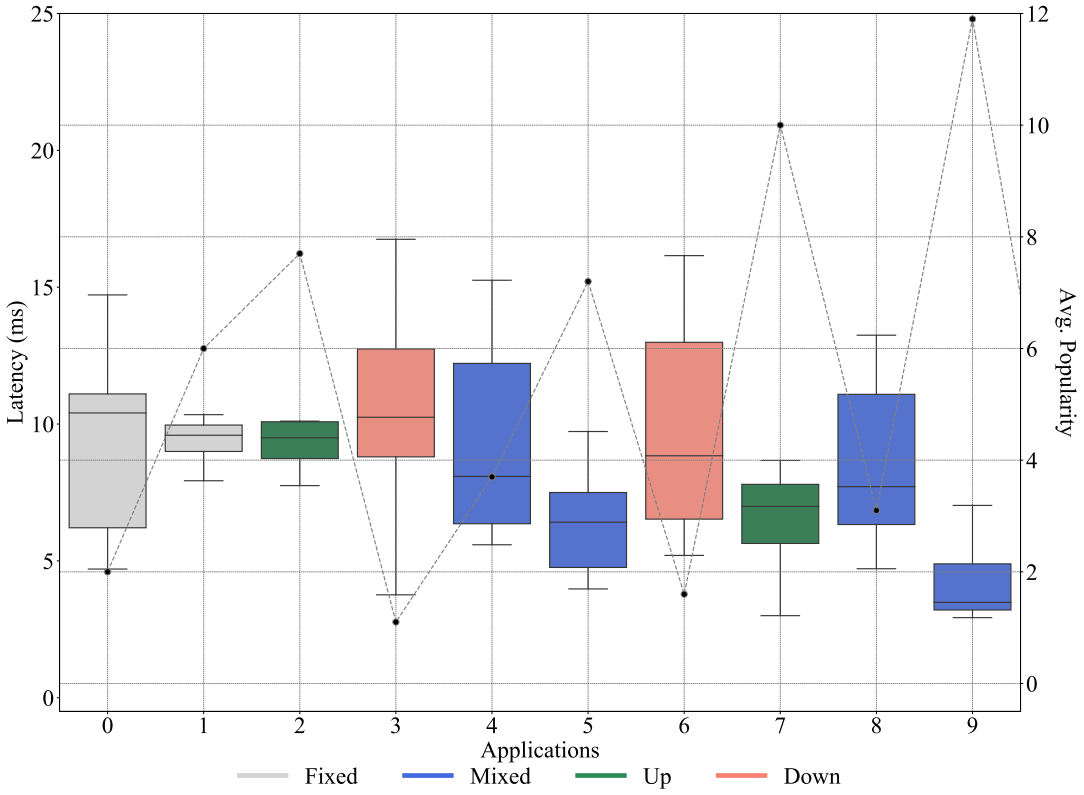


Figure 5.12: Latency and Jitter per Application - FF - Medium Scenario

Figures 5.13, 5.14, and 5.15 show the results for the large scenario for PRP, GA, and FF respectively.

PRP showed the best results regarding latency. The most popular applications were benefited in comparison with less popular applications, for latency as well as jitter. The profiles that were benefited the most by this mechanism were *Up* and *Fixed*, particularly regarding the jitter.

For GA, the applications with the highest jitter were the least popular as well (*Down* and *Mixed* profiles); although in general, this mechanism showed higher levels of latency than PRP. FF showed to be the least efficient mechanism in the evaluation, displaying the highest observed latency and jitter, as well as no evident differentiation in the treatment of applications according to their profile.

As in the previous scenarios, PRP displayed a clear advantage for the applications according to their profiles, followed by GA. FF did not show any difference in the treatment of the applications according to their profile.

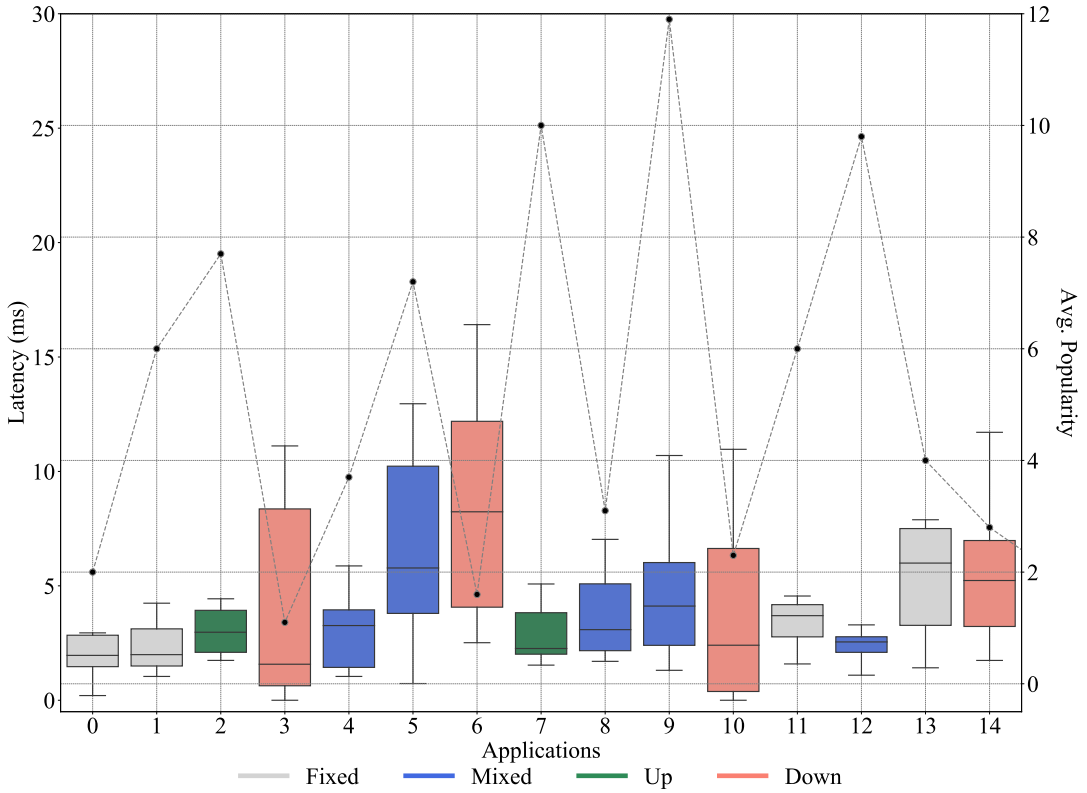


Figure 5.13: Latency and Jitter per Application - PRP - Large Scenario

For all the scenarios, with PRP and GA it is noticeable that as the popularity grows or remains the same, the latency tends to remain stable, with smaller jitter values. On the other hand, as the popularity decreases or oscillates, the latency shows significant variations, increasing the jitter. FF showed less differentiating behavior regarding the popularity of the applications, as well as displaying the highest latency and jitter values. The jitter can be particularly damaging for

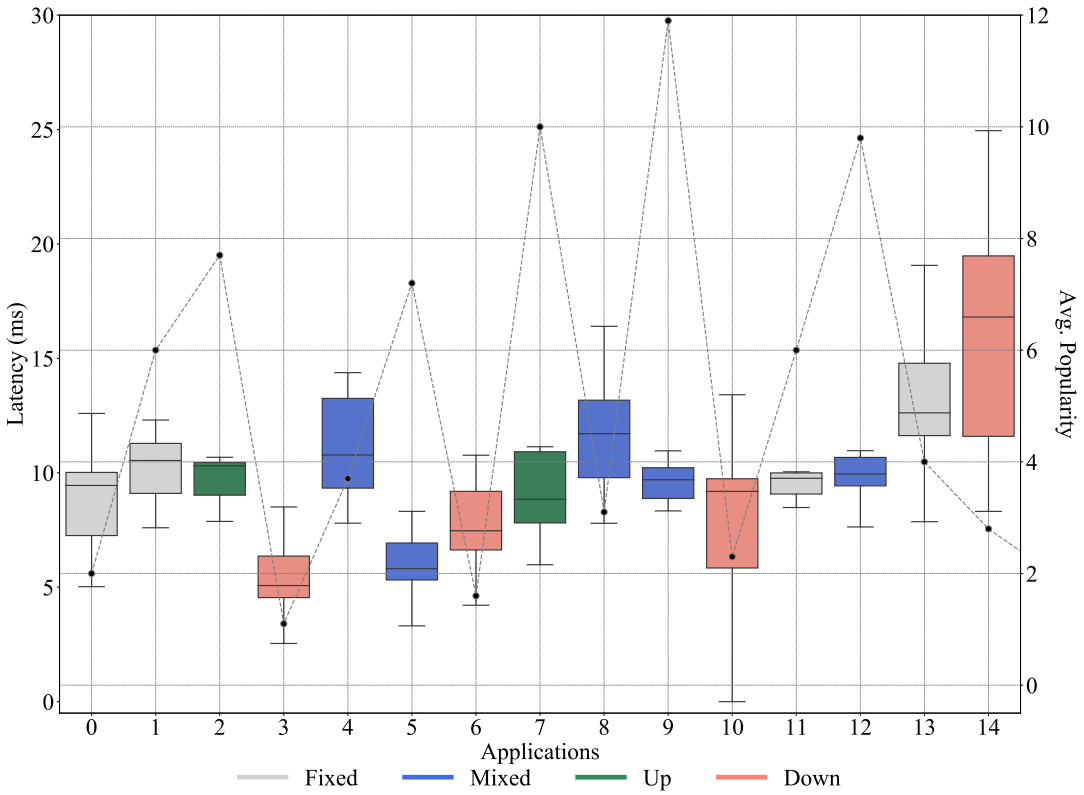


Figure 5.14: Latency and Jitter per Application - GA - Large Scenario

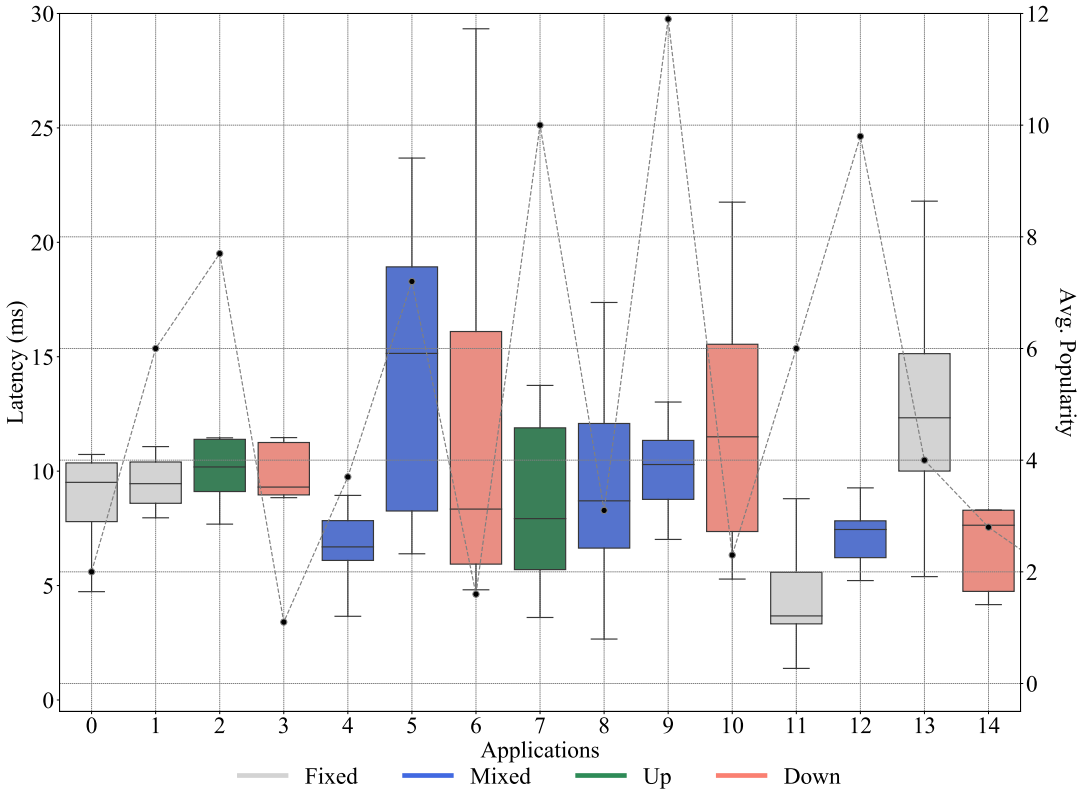


Figure 5.15: Latency and Jitter per Application - FF - Large Scenario

certain types of applications, such as real-time communications, virtual/augmented reality, and Cloud games. The mechanisms shown in this chapter (especially PRP) could be adapted to use a different profiling system, not based on the popularity of the applications but in other relevant factors, for instance how sensitive they are to jitter. This type of profiling system could prioritize jitter-sensitive applications, offering them lower jitter values while maintaining low latency levels for the overall system.

5.5 Chapter Summary

The PRP mechanism presented in the previous chapter was evaluated under dynamic conditions. An additional heuristic based on Genetic Algorithms was also introduced. Both mechanisms take into consideration the popularity of the applications as well as the propagation delay as metrics to guide the placement process. An application profiling system based on changing popularity was also presented.

The simulations were conducted using YAFS, and FF was used as benchmark for evaluation purposes. The experimental evaluation showed that PRP outperformed the GA in every scenario, which in turn also outperformed FF. The advantages of using the GA are diminished as the load increases, and more generations are needed to reach better results, which implies higher execution times. An additional advantage of using the profiling system proposed was observed in the results, as more popular applications (growing and fixed popularity) suffered lower jitter than less popular applications (oscillating or decreasing popularity). This profiling system prioritizes popular applications, but a profiling system that organizes applications from more sensitive to less sensitive to jitter could be used instead of the popularity to benefit this type of applications.

As it was with the previous chapter, the mechanisms presented here for dynamic scenarios could also be integrated in the *Service Orchestrator* (depicted in Section 3.2.3) presented in Chapter 3. The proposals presented in this chapter are partially reported in the following publication.

Description of the Genetic Algorithm-based heuristic, and the modifications on the PRP heuristic for dynamic scenarios, and the profiling of applications based on their popularity:

> Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2021). Service Placement for Latency Reduction in the Fog via Application Profiling. Submitted for publication to *IEEE Access*, pages 1-15. IEEE.

Chapter 6

Conclusions and Future Work

Contents

The development and deployment of IoT devices enable the proliferation of new services and applications. Among them, augmented reality and virtual reality, as well as video over IP, constitute the more significant portion of IP traffic for the upcoming years. These types of applications have special requirements, including low latency and jitter. In many cases, the requirements established by the applications cannot be met by the IoT devices. The Cloud paradigm offers a solution for some of these applications, but other applications and services are not particularly fit for the Cloud. The Fog extends the idea of the Cloud bringing resources to the edge of the network, closer to the final user, enabling lower latency levels, location awareness, and mobility support among other advantages.

The Fog encompasses a highly complex scenario with a huge amount of different devices that must cooperate with each other. This requires effective orchestration mechanisms to guarantee the smooth performance of applications and services. However, mechanisms typically applied to the Cloud can not directly be migrated to the Fog given its particular characteristics. This calls for the design and development of new orchestration mechanisms for the Fog, particularly to handle the demands of low latency.

6.1 Synthesis of the Thesis

In this thesis, a hybrid approach to manage the Fog is proposed, using the combination of orchestration and choreography styles of managing for different regions of the infrastructure. The solution outlined enables a global view which allows general optimizations, and also automated dynamic reactions at the lower levels. A framework for smart orchestration and an architecture for the Service Orchestrator are introduced. The Service Orchestrator architecture is detailed enough that it incorporates the description of its modules and how those modules could interact.

The proposed architecture for the Service Orchestrator allows the use of multiple instances so that different tenants could adapt it to their particular requirements. By using different instances of the orchestrator, it is possible to apply different optimization goals according to the requirements of the applications.

For the module *Planner Mechanisms* in the Service Orchestrator, different service placement solutions were proposed, using mixed metrics that consider the network infrastructure and the applications simultaneously. An ILP model for service placement, aimed at maximizing the placement of popular applications, while minimizing their latency is described. Moreover, a heuristic based on the PageRank algorithm, called Popularity Ranked Placement, is also introduced. PRP ranks the applications according to their requests as a measure of their popularity, and also ranks the nodes in the network topology to create communities with the nodes with the highest transition probability; this way, the placement load is divided among the nodes within the community, avoiding congestion while also maintaining low latency levels. An expanding window controls the placement among the nodes in the community.

The performance of both the ILP model and the PRP solution were tested using YAFS, and are also compared with the FF approach. Simulation results show that while the ILP model had the lowest latency for all the scenarios, it also had the highest concentration of placement in the same nodes, thus generating congestion in the processing nodes and for the communication links. PRP kept the latency at low levels, close to the ILP results, but the load was balanced between the nodes in the communities. Both mechanisms were able to favour popular applications, prioritizing them for the latency reduction. Moreover, PRP showed significantly lower execution times than the ILP model, which makes it more suitable for more complex and dense scenarios, and is the option that should be deployed in practical realistic situations.

Once that it was determined that using the popularity of the applications resulted in differentiating the perceived latency of the applications, a profiling system for the applications based on their modifying popularity was suggested. This profiling system was evaluated with the use of PRP and an additional heuristic, based on Genetic Algorithms. A time window approach was applied to modify the traffic conditions in the network and creating a dynamic scenario by altering the current request rate, i.e., popularity, of the applications by following the different profiles in the system. The profiling system proved to have an impact in the final placement of the applications, benefiting those applications with growing popularity levels, providing them not only lower latency but also lower jitter.

The experiments were also conducted using YAFS, and showed that PRP maintained the lowest overall latency in comparison with GA, both mechanisms differentiated the perceived latency by the applications according to their popularity profile. Applications that constantly gained popularity over time or that maintained their popularity value, obtained lower latency and lower jitter over time, while applications that lowered or oscillated their popularity level suffered from higher latency and jitter. A different profiling system could be adapted to benefit time and jitter-sensitive applications, while maintaining the rest of the mechanisms as described in this work.

6.2 Contributions

The following contributions were achieved during this research:

- A proposal of a hybrid approach for orchestration in Fog environments, combining orchestration in the upper layers and choreography for the lower layers;
- A smart orchestration framework, following the proposed hybrid approach;
- An architecture for a Service Orchestrator that can be instantiated with different optimization goals, including the characterization of its modules and their interaction;
- A mathematical model based on Integer Linear Programming, to optimize the service placement based on the popularity of the applications, aimed

at reducing the latency;

- A heuristic based on graph partition, that prioritizes popular applications obtaining latency levels close to the optimal solution;
- A proposal of application profiling based on the popularity patterns of the applications; and
- A multi-objective heuristic based on a Genetic Algorithm that weights the applications according to their popularity, to minimize latency.

All the mechanisms proposed were evaluated using YAFS. Simulations showed that PRP had results close to the ones obtained with the optimization model, but with a considerable reduction in node congestion and execution time. Also, PRP and GA were able to improve application performance providing them lower latency and lower jitter, according to their popularity profiles in dynamic scenarios. These proposals, as well as the literature review performed, produced four international conference papers (three as first author) and four journal articles (three as first author). Furthermore, cooperative work connected with this research resulted in additional publications, namely, two journal articles and six international conference papers.

6.3 Future Work

Possible research paths to follow up this work include taking in consideration mobile users in the placement process, to automate the decision about migrating the services according to the final user movement. The migration could be predicted using machine learning techniques [Anagnostopoulos et al., 2011; Tang et al., 2019].

The placement mechanisms could also be enhanced by modifying their main optimization goal and including additional metrics in the placement decision, such as node availability, so in the case of failures the application remains active. It would also be interesting to determine the impact in the resource usage of the communication infrastructure by altering the optimization goals of the placement mechanisms.

Another possibility is designing additional mechanisms for the remaining modules in the service orchestrator architecture, and not only focusing on placement mechanisms for the Planner and Optimizer modules. For instance, for the Security Manager, mechanisms regarding authentication and privacy are needed. Furthermore, the Communication Manager should be further explored, since it allows the interaction of different orchestrator instances. This module also enables the communication and cooperation with other Management and Orchestration (MANO) instances such as the ones described for 5G and Network Function Virtualization (NFV) architectures [Yousaf et al., 2019; Mechtri et al., 2017].

Regarding the profiling analysis for the applications, it would be interesting to use a different approach and instead of using the popularity of the applications

use their domains. This is, if they need to have a reliable services, optimized bandwidth or delay.

Finally, another approach could be exploring machine learning techniques with alternative heuristics for the service placement [Rahbari and Nickray, 2020]. For this, it is necessary to have access to datasets that contain enough information for the learning process.

These extensions will strengthen the orchestration tasks, completing a fuller function. Some limitations should be taken into consideration before delving into the development of these works, such as the need of proper datasets required for the machine learning techniques.

Bibliography

Aazam, M. and Huh, E.-N. (2014). Fog Computing and Smart Gateway Based Communication for Cloud of Things. In *2014 International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 464–470, Catalonia, Spain. IEEE.

Aazam, M. and Huh, E.-N. (2015a). Dynamic resource provisioning through Fog micro datacenter. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 105–110, St. Louis, MO, USA. IEEE.

Aazam, M. and Huh, E.-N. (2015b). Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 687–694, Gwangiu, South Korea. IEEE.

Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N., and Chen, Y. (2018). Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, 56(9):118–123.

Alimi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and Woundy, R. (2014). Application-Layer Traffic Optimization (ALTO) Protocol. Technical Report RFC7285, IETF.

Anagnostopoulos, T., Anagnostopoulos, C., and Hadjiefthymiades, S. (2011). Mobility Prediction Based on Machine Learning. In *2011 IEEE 12th International Conference on Mobile Data Management*, pages 27–30, Lulea, Sweden. IEEE.

Apple Technology Company (2020). HealthKit. `https://developer.apple.com/healthkit/`. Accessed 01 Oct 2020.

Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C., and Rana, O. (2018). The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet of Things*, 3-4(1):134 – 155.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the*

MCC Workshop on Mobile Cloud Computing, pages 13–16, Helsinki, Finland. ACM.

Borgia, E. (2014). The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54(C):1–31.

Borujeni, E. M., Rahbari, D., and Nickray, M. (2018). Fog-based energy-efficient routing protocol for wireless sensor networks. *The Journal of Supercomputing*, 74(12):6831–6858.

Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56:684–700.

Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, I.-J., Gjessing, S., Fairhurst, G., Griwodz, C., and Welzl, M. (2014). Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Communications Surveys Tutorials*, 18(3):2149 – 2196.

Brogi, A. and Forti, S. (2017). QoS-Aware Deployment of IoT Applications Through the Fog. *IEEE Internet of Things Journal*, 4(5):1185–1192.

Chen, Y.-C., Lim, Y.-s., Gibbens, R. J., Nahum, E. M., Khalili, R., and Towsley, D. (2013). A Measurement-based Study of MultiPath TCP Performance over Wireless Networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, pages 455–468, Barcelona, Spain. ACM.

Chiang, M. and Zhang, T. (2016). Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, 3(6):854–864.

Cisco (2015). Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are - White Paper. `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`. Accessed 01 Oct 2020.

Cisco (2016). The Zettabyte Era: Trends and Analysis - White Paper. `http://www.netmode.ntua.gr/courses/postgraduate/video_communications/2014/VNI_Hyperconnectivity_WP.pdf`. Accessed 01 Oct 2020.

Cisco (2019). Cisco Visual Networking Index: Forecast and Trends. Technical Report 2017–2022, Cisco.

Cisco (2020). Cisco Visual Networking Index: Forecast and Trends. Technical Report 2018–2023, Cisco.

Dastjerdi, A. V. and Buyya, R. (2016). Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer Magazine*, 49(8):112–116.

Davies, J., Schorow, D., Ray, S., and Rieber, D. (2008). Asynchronous Messaging. In *The Definitive Guide to SOA: Oracle Service Bus*, pages 119–142. Apress.

de Brito, M. S., Hoque, S., Magedanz, T., Steinke, R., Willner, A., Nehls, D., Keil, O., and Schreiner, F. (2017). A service orchestration architecture for Fog-enabled infrastructures. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 127–132, Valencia, Spain. IEEE.

Deng, R., Lu, R., Lai, C., Luan, T. H., and Liang, H. (2016). Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181.

Ding, A. Y. and Janssen, M. (2018). 5G Applications: Requirements, Challenges, and Outlook. In *Seventh International Conference on Telecommunications and Remote Sensing*, pages 27–34, Barcelona, Spain. ACM.

Dolui, K. and Datta, S. K. (2017). Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, Geneva, Switzerland. IEEE.

Dukkipati, N. and McKeown, N. (2006). Why Flow-Completion Time is the Right Metric for Congestion Control. *SIGCOMM Computer Communication Review*, 36(1):59–62.

Espling, D., Larsson, L., Li, W., Tordsson, J., and Elmroth, E. (2016). Modeling and Placement of Cloud Services with Internal Structure. *IEEE Transactions on Cloud Computing*, 4(4):429–439.

ETSI GS MEC (2016). Mobile Edge Computing (MEC); Framework and Reference Architecture. Technical Report ETSI GS MEC 003. v1.1.1, European Telecommunications Standards Institute.

ETSI GS NFV-MAN (2014). Network Functions Virtualisation (NFV); Management and Orchestration. Technical Report ETSI GS NFV-MAN 001. v1.1.1, European Telecommunications Standards Institute.

Faigl, Z., Szabó, Z., and Schulcz, R. (2014). Application-Layer Traffic Optimization in Software-Defined Mobile Networks: a Proof-of-Concept Implementation. In *Proceedings of the 16th International Telecommunications Network Strategy and Planning Symposium (Networks)*, pages 1–6, Funchal, Portugal. IEEE.

Fitbit, Inc. (2020). Fitbit. `https://www.fitbit.com`. Accessed 01 Oct 2020.

Ford, M. (2014). Workshop Report: Reducing Internet Latency, 2013. *ACM SIGCOMM Computer Communication Review*, 44(2):80–86.

Gen, M. and Cheng, R. (2000). *Genetic algorithms and engineering optimization*. John Wiley & Sons, 1 edition.

Ghaznavi, M., Khan, A., Shahriar, N., Alsubhi, K., Ahmed, R., and Boutaba, R. (2015). Elastic Virtual Network Function Placement (EVNFP). In *4th IEEE Conference on Cloud Networking (CloudNet) 2015*, pages 255–260, Niagara Falls, ON, Canada. IEEE.

Guerrero, C., Lera, I., and Juiz, C. (2019a). A lightweight decentralized service placement policy for performance optimization in fog computing. *Journal of Ambient Intelligence and Humanized Computing*, 10(1):2435–2452.

Guerrero, C., Lera, I., and Juiz, C. (2019b). Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures. *Future Generation Computer Systems*, 97(1):131 – 144.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296.

Gutierrez, F. (2017). Messaging. In *Spring Boot Messaging*, pages 1–5. Apress.

Hao, Z., Novak, E., Yi, S., and Li, Q. (2017). Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, 21(2):44–53.

He, J., Wei, J., Chen, K., Tang, Z., Zhou, Y., and Zhang, Y. (2018). Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities. *IEEE Internet of Things Journal*, 5(2):677–686.

Hoque, S., De Brito, M. S., Willner, A., Keil, O., and Magedanz, T. (2017). Towards Container Orchestration in Fog Computing Infrastructures. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, pages 294–299, Turin, Italy. IEEE.

Hung, S. C., Hsu, H., Lien, S. Y., and Chen, K. C. (2015). Architecture Harmonization Between Cloud Radio Access Networks and Fog Networks. *IEEE Access*, 3(1):3019–3034.

IBM (2019). CPLEX Optimizer. `https://www.ibm.com/analytics/cplex-optimizer`. Accessed: 2019-12-20.

ITU-T (2013). Roadmap for the Quality of Service of Interconnected Networks that use the Internet Protocol. Technical Report Y.1545, International Telecommunication Union - Telecommunication Standardization Sector.

Jalili, M. and Perc, M. (2017). Information cascades in complex networks. *Journal of Complex Networks*, 5(5):665–693.

Jazdi, N. (2014). Cyber physical systems in the context of Industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4, Cluj-Napoca, Romania. IEEE.

Jiang, Y., Huang, Z., and Tsang, D. H. K. (2018). Challenges and Solutions in Fog Computing Orchestration. *IEEE Network*, 32(3):122–129.

Kadhim, A. J. and Seno, S. A. H. (2019). Energy-efficient multicast routing protocol based on SDN and fog computing for vehicular networks. *Ad Hoc Networks*, 84(1):68 – 81.

Khan, M. G., Taheri, J., Kassler, A., and Darula, M. (2018). Automated analysis and profiling of virtual network functions: the nfv-inspector approach. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–2, Verona, Italy. IEEE.

Khebbache, S., Hadji, M., and Zeghlache, D. (2018). A multi-objective non-dominated sorting genetic algorithm for VNF chains placement. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–4, Las Vegas, NV, USA. IEEE.

Kim, N. Y., Ryu, J. H., Kwon, B. W., Pan, Y., and Park, J. H. (2018). CF-CloudOrch: container fog node-based cloud orchestration for IoT networks. *The Journal of Supercomputing*, 74(1):7024–7045.

Krishnamurthy, B., Wills, C., and Zhang, Y. (2001). On the Use and Performance of Content Distribution Networks. In *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop - IMW '01*, pages 169–182, San Francisco, California, USA. ACM.

Langville, A. N. and Meyer, C. D. (2012). *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press.

Lee, K. Y. and El-Sharkawi, M. A. (2008). *Modern heuristic optimization techniques: theory and applications to power systems*. John Wiley & Sons, Ltd, 1st edition.

Leite, L. A. F., Ansaldi Oliva, G., Nogueira, G. M., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2013). A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, 7(3):199–216.

Lera, I., Guerrero, C., and Juiz, C. (2019a). Availability-Aware Service Placement Policy in Fog Computing Based on Graph Partitions. *IEEE Internet of Things Journal*, 6(2):3641–3651.

Lera, I., Guerrero, C., and Juiz, C. (2019b). YAFS: A Simulator for IoT Scenarios in Fog Computing. *IEEE Access*, 7(1):91745–91758.

Liu, L., Chang, Z., Guo, X., Mao, S., and Ristaniemi, T. (2018). Multiobjective Optimization for Computation Offloading in Fog Computing. *IEEE Internet of Things Journal*, 5(1):283–294.

Liu, X. and Murata, T. (2010). Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*, 389(7):1493 – 1500.

Lu, T., Chang, S., and Li, W. (2018). Fog computing enabling geographic routing for urban area vehicular network. *Peer-to-Peer Networking and Applications*, 11(4):749–755.

Luan, T. H., Gao, L., Li, Z., Xiang, Y., Wei, G., and Sun, L. (2016). Fog Computing: Focusing on Mobile Users at the Edge. `https://arxiv.org/pdf/1502.01815.pdf`. Accessed 01 Oct 2020.

Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. (2019). Quality of Experience (QoE)-aware placement of applications in Fog computing environments. *Journal of Parallel and Distributed Computing*, 132(1):190 – 203.

Manasrah, A. M., Aldomi, A., and Gupta, B. B. (2019). An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment. *Cluster Computing*, 22(1):1639–1653.

Mechtri, M., Ghribi, C., Soualah, O., and Zeghlache, D. (2017). NFV Orchestration Framework Addressing SFC Challenges. *IEEE Communications Magazine*, 55(6):16–23.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, 1 edition.

Moens, H., Hanssens, B., Dhoedt, B., and Turck, F. D. (2014). Hierarchical network-aware placement of service oriented applications in Clouds. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, Krakow, Poland. IEEE.

Naas, M. I., Lemarchand, L., Boukhobza, J., and Raipin, P. (2018). A Graph Partitioning-Based Heuristic for Runtime IoT Data Placement Strategies in a Fog Infrastructure. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, page 767–774, Pau, France. ACM.

Naranjo, P. G. V., Shojafar, M., Mostafaei, H., Pooranian, Z., and Baccarelli, E. (2017). P-SEP: a prolong stable election routing algorithm for energy-limited heterogeneous fog-supported wireless sensor networks. *The Journal of Supercomputing*, 73(2):733–755.

Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:1–15.

OASIS Standards Group (2013). TOSCA. Topology and Orchestration Specification for Cloud Applications. Technical Report TOSCA-v1.0-os, OASIS.

Okay, F. Y. and Ozdemir, S. (2018). Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution. *IEEE Internet of Things Journal*, 5(6):4871–4889.

OpenFog Consortium (2017). OpenFog Reference Architecture for Fog Computing. Technical Report OPFRA001.020817, OpenFog Consortium.

Osanaiye, O., Chen, S., Yan, Z., Lu, R., Choo, K. K. R., and Dlodlo, M. (2017). From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access*, 5(1):8284–8300.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab.

Parés, F., Gasulla, D. G., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., Cortés, U., and Suzumura, T. (2018). Fluid communities: A competitive, scalable and diverse community detection algorithm. In *Complex Networks & Their Applications VI*, pages 229–240, Lyon, France. Springer.

Paul Martin, J., Kandasamy, A., and Chandrasekaran, K. (2020). CREW: Cost and Reliability aware Eagle-Whale optimiser for service placement in Fog. *Software: Practice and Experience*, 50(12):2337–2360.

Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2020). A comparative analysis of simulators for the Cloud to Fog continuum. *Simulation Modelling Practice and Theory*, 101(1):1020291–10202927.

Peterson, L. L. and Davie, B. S. (2011). *Computer Networks, Fifth Edition: A Systems Approach*. Elsevier, 5th edition.

Pu, C. (2018). Jamming-Resilient Multipath Routing Protocol for Flying Ad Hoc Networks. *IEEE Access*, 6(1):68472–68486.

Qiuli, C., Wei, X., Fei, D., and Ming, H. (2019). A reliable routing protocol against hotspots and burst for UASN-based fog systems. *Journal of Ambient Intelligence and Humanized Computing*, 10(8):3109–3121.

Rahbari, D. and Nickray, M. (2020). Task offloading in mobile fog computing by classification and regression tree. *Peer-to-Peer Networking and Applications*, 13(1):104–122.

Rotsos, C., Farshad, A., Hart, N., Aguado, A., Bidkar, S., Sideris, K., King, D., Fawcett, L., Bird, J., Mauthe, A., Race, N., and Hutchison, D. (2016). Baguette: Towards End-to-End Service Orchestration in Heterogeneous Networks. In *2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS)*, pages 196–203, Granada, Spain. IEEE.

Rotsos, C., King, D., Farshad, A., Bird, J., Fawcett, L., Georgalas, N., Gunkel, M., Shiomoto, K., Wang, A., Mauthe, A., Race, N., and Hutchison, D. (2017). Network service orchestration standardization: A technology survey. *Computer Standards & Interfaces*, 54(4):203 – 215.

Rumble, S. M., Ongaro, D., Stutsman, R., Rosenblum, M., and Ousterhout, J. K. (2011). It's Time for Low Latency. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, pages 11–11, Napa, California. USENIX Association.

Sadfi, C., Penz, B., Rapine, C., Błażewicz, J., and Formanowicz, P. (2005). An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. *European Journal of Operational Research*, 161(1):3–10.

Saini, M., Alam, K. M., Guo, H., Alelaiwi, A., and El Saddik, A. (2017). InCloud: a cloud-based middleware for vehicular infotainment systems. *Multimedia Tools and Applications*, 76(9):11621–11649.

Santoro, D., Zozin, D., Pizzolli, D., De Pellegrini, F., and Cretti, S. (2017). Foggy: A Platform for Workload Orchestration in a Fog Computing Environment. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 231–234, Hong Kong, China. IEEE.

Santos, H., Alencar, D., Meneguette, R., Rosário, D., Nobre, J., Both, C., Cerqueira, E., and Braun, T. (2020). A multi-tier fog content orchestrator mechanism with quality of experience support. *Computer Networks*, 177(1):107288.

Santos, J., Wauters, T., Volckaert, B., and Turck, F. D. (2017). Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks. *Entropy*, 20(4):1–26.

Saraiva de Sousa, N. F., Lachos Perez, D. A., Rosa, R. V., Santos, M. A., and Esteve Rothenberg, C. (2019). Network Service Orchestration: A survey. *Computer Communications*, 142-143(1):69 – 94.

Seedorf, J. and Burger, E. (2009). Application-Layer Traffic Optimization (ALTO) Problem Statement. Technical Report RFC5693, IETF.

Sheng, Z., Wang, H., Yin, C., Hu, X., Yang, S., and Leung, V. C. M. (2015). Lightweight Management of Resource-Constrained Sensor Devices in Internet of Things. *IEEE Internet of Things Journal*, 2(5):402–411.

Siddiqi, M. A., Yu, H., and Joung, J. (2019). 5G Ultra-Reliable Low-Latency Communication Implementation Challenges and Operational Issues with IoT Devices. *Electronics*, 8(9):981.

Siltanen, R. (2013). Designing low latency networks - Whitepaper sponsored by Nokia. `http://blog.networks.nokia.com/mobile-networks/2013/03/27/designing-low-latency-networks/`. Accessed 01 Oct 2020.

SINTEF, S. (2013). Collaborating Smart Solar-powered Micro-grids. `https://cordis.europa.eu/project/id/608806`. Accessed 01 Dec 2020.

Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., and Leitner, P. (2017a). Optimized IoT service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443.

Skarlat, O., Nardelli, M., Schulte, S., and Dustdar, S. (2017b). Towards QoS-Aware Fog Service Placement. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 89–96, Madrid, Spain. IEEE.

Souza, V., Masip-Bruin, X., Marín-Tordera, E., Sànchez-López, S., Garcia, J., Ren, G., Jukan, A., and Ferrer, A. J. (2018). Towards a proper service placement in combined Fog-to-Cloud (F2C) architectures. *Future Generation Computer Systems*, 87(1):1–15.

Steiner, M., Gaglianello, B. G., Gurbani, V., Hilt, V., Roome, W., Scharf, M., and Voith, T. (2012). Network-aware Service Placement in a Distributed

Cloud Environment. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 73–74, Helsinki, Finland. ACM.

Stojmenovic, I. (2014). Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)*, pages 117–122, Southbank, VIC, Australia. IEEE.

Stojmenovic, I. and Wen, S. (2014). The Fog computing paradigm: Scenarios and security issues. In *2014 Federated Conference on Computer Science and Information Systems*, pages 1–8, Warsaw, Poland. IEEE.

Taherizadeh, S., Stankovski, V., and Grobelnik, M. (2018). A Capillary Computing Architecture for Dynamic Internet of Things: Orchestration of Microservices from Edge Devices to Fog and Cloud Providers. *Sensors*, 18(9):2938.

Taneja, M. and Davy, A. (2017). Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228, Lisbon, Portugal. IEEE.

Tang, Y., Cheng, N., Wu, W., Wang, M., Dai, Y., and Shen, X. (2019). Delay-Minimization Routing for Heterogeneous VANETs With Machine Learning Based Mobility Prediction. *IEEE Transactions on Vehicular Technology*, 68(4):3967–3979.

Tong, W. and Zhu, P. (2014). 5G: A Technology Vision - Whitepaper sponsored by Huawei. `https://www.huawei.com/mediafiles/CORPORATE/PDF/Magazine/WinWin/HW_329327.pdf`. Accessed 01 Oct 2020.

Ullah, A., Yao, X., Shaheen, S., and Ning, H. (2020). Advances in Position Based Routing Towards ITS Enabled FoG-Oriented VANET–A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 21(2):828–840.

Vaquero, L. M. and Rodero-Merino, L. (2014). Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32.

Varshney, P. and Simmhan, Y. (2017). Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 115–124, Madrid, Spain. IEEE.

Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2020). Service Placement via Application Profiling. `https://git.dei.uc.pt/kcastro/appProfiling.git`. Accessed: 2020-10-20.

Velasquez, K., Perez Abreu, D., Paquete, L., Curado, M., and Monteiro, E. (2019). Rank-based Service Placement - GitLab Repository. `https://git.dei.uc.pt/kcastro/rankPlacement.git`. Accessed: 2019-12-20.

Venticinque, S. and Amato, A. (2019). A methodology for deployment of IoT application in fog. *Journal of Ambient Intelligence and Humanized Computing*, 10(1):1955–1976.

Viejo, A. and Sanchez, D. (2019). Secure and privacy-preserving orchestration and delivery of fog-enabled IoT services. *Ad Hoc Networks*, 82(1):113 – 125.

Vulimiri, A., Godfrey, P. B., Mittal, R., Sherry, J., Ratnasamy, S., and Shenker, S. (2013). Low Latency via Redundancy. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, pages 283–294, Santa Barbara, California, USA. ACM.

Vulimiri, A., Michel, O., Godfrey, P. B., and Shenker, S. (2012). More is Less: Reducing Latency via Redundancy. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 13–18, Redmond, Washington. ACM.

Wang, J., Zhou, M., and Li, Y. (2004). Survey on the End-to-End Internet Delay Measurements. In *High Speed Networks and Multimedia Communications*, pages 155–166, Berlin, Heidelberg. Springer.

Wang, S., Urgaonkar, R., He, T., Chan, K., Zafer, M., and Leung, K. K. (2017). Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):1002–1016.

Wang, T., Zeng, J., Lai, Y., Cai, Y., Tian, H., Chen, Y., and Wang, B. (2020). Data collection from WSNs to the cloud based on mobile Fog elements. *Future Generation Computer Systems*, 105(1):864 – 872.

Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., and Rovatsos, M. (2017). Fog Orchestration for IoT Services: Issues, Challenges and Directions. *IEEE Internet Computing*, 21(2):16–24.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85.

Xiong, G., Hu, Y.-x., Tian, L., Lan, J.-l., Li, J.-f., and Zhou, Q. (2016). A virtual service placement approach based on improved quantum genetic algorithm. *Frontiers of Information Technology & Electronic Engineering*, 17(7):661–671.

Yao, B., Liu, X., Zhang, W., Chen, X., and Yao, M. (2014). Nested growing network models for researching the Internet of Things. In *2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference*, pages 450–454, Chongqing, China. IEEE.

Yi, S., Hao, Z., Qin, Z., and Li, Q. (2015a). Fog Computing: Platform and Applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78, Washington, DC, USA. IEEE.

Yi, S., Li, C., and Li, Q. (2015b). A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Hangzhou, China. ACM.

Yousaf, F. Z., Sciancalepore, V., Liebsch, M., and Costa-Perez, X. (2019). MANOaaS: A Multi-Tenant NFV MANO for 5G Network Slices. *IEEE Communications Magazine*, 57(5):103–109.

Zhang, Q., Zhu, Q., Zhani, M. F., and Boutaba, R. (2012). Dynamic Service Placement in Geographically Distributed Clouds. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 526–535, Macau, China. IEEE.

Zhou, Y. T., Deng, M. L., Ji, F. Z., He, X. G., and Tang, Q. J. (2015). Discovery Algorithm for Network Topology Based on SNMP. In *International Conference on Automation, Mechanical Control and Computational Engineering*, pages 1623–1628. Atlantis Press.

Appendix

A Results - Latency by Application

This appendix presents the results for the experiments to measure the latency of each application considering its popularity in static scenarios. These plots complement the results presented in Chapter 4, Section 4.4, showing the same trends.

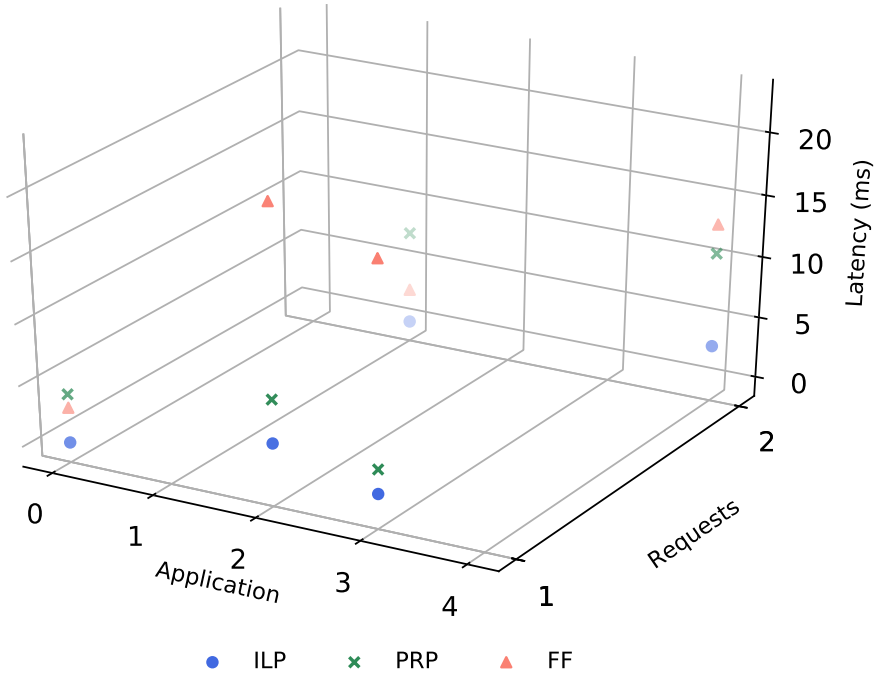


Figure A.1: Latency by Application - Fire&Forget Applications - Tiny Scenario

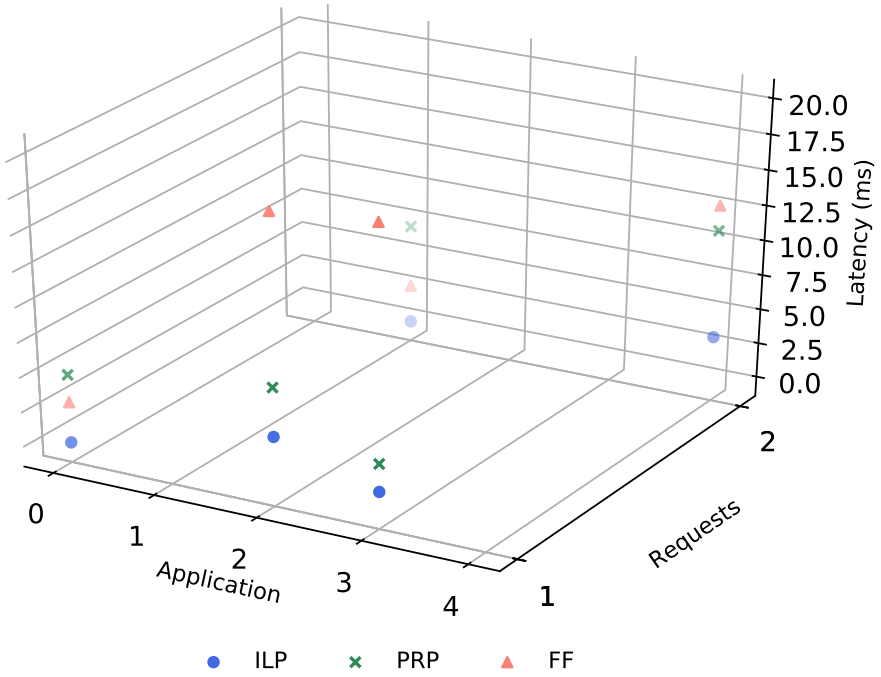


Figure A.2: Latency by Application - Asynchronous Response Applications - Tiny Scenario

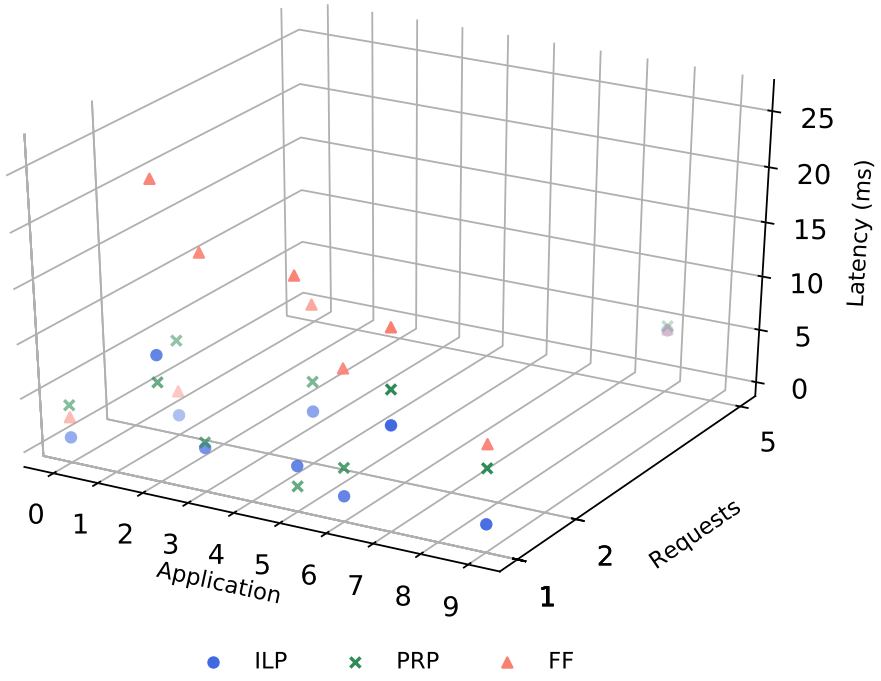


Figure A.3: Latency by Application - Fire&Forget Applications - Small Scenario

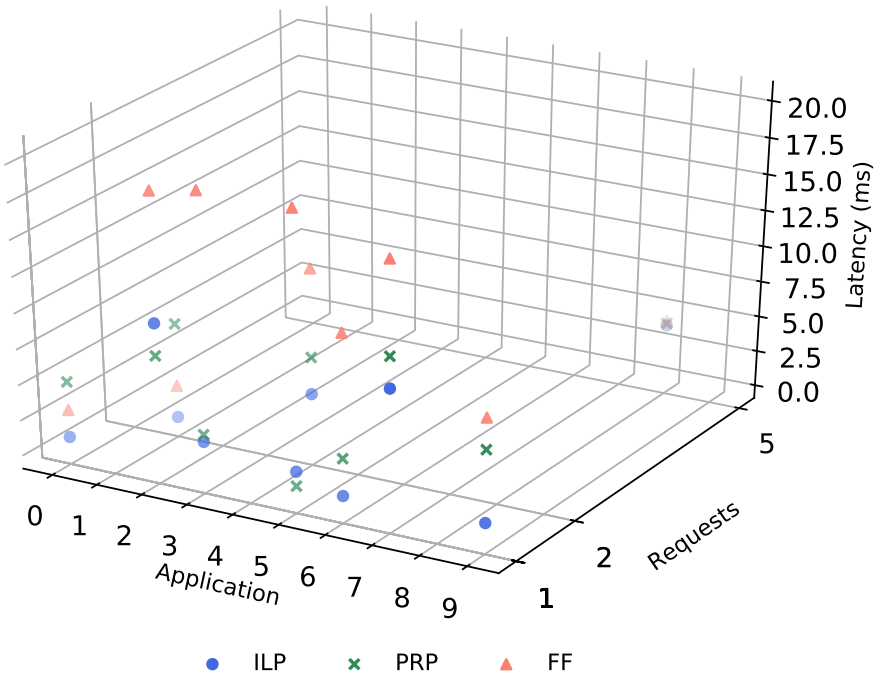


Figure A.4: Latency by Application - Asynchronous Response Applications - Small Scenario

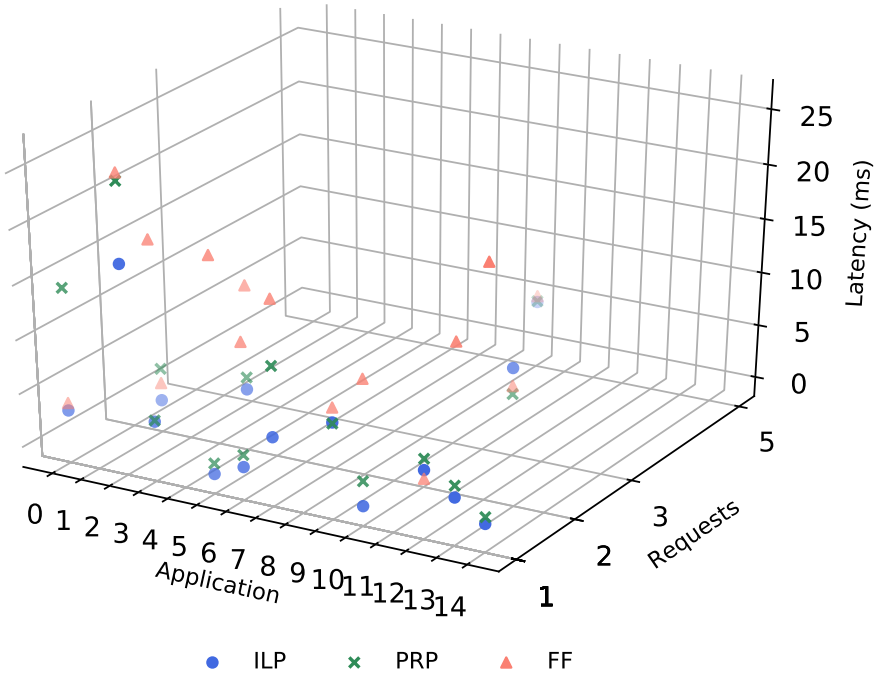


Figure A.5: Latency by Application - Fire&Forget Applications - Medium Scenario

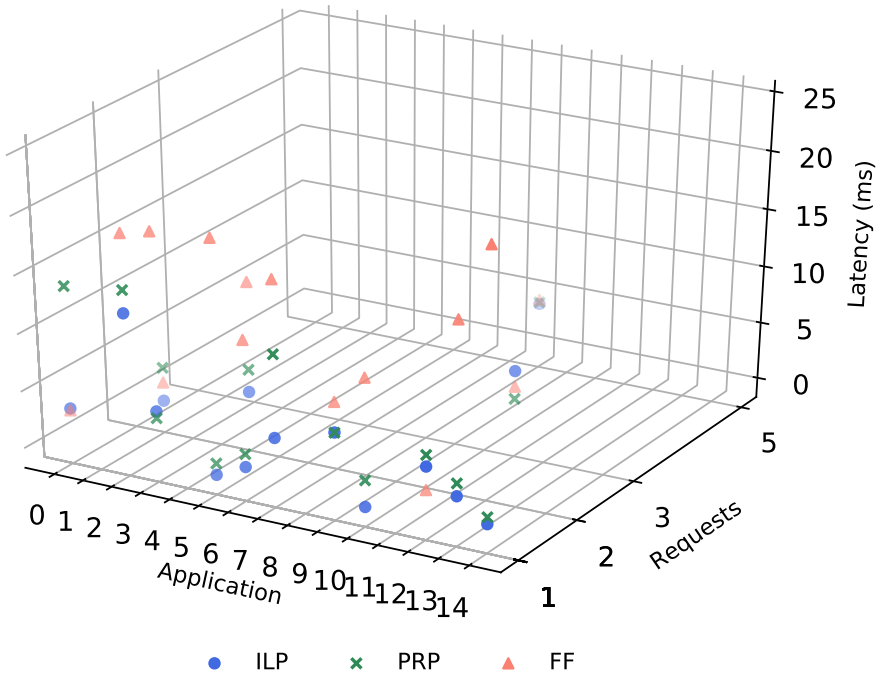


Figure A.6: Latency by Application - Asynchronous Response Applications - Medium Scenario