1 2 9 0

UNIVERSIDADE DE COIMBRA

David Alejandro Perez Abreu

Resilience in the Internet of Things for Smart City Applications

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

Dezembro de 2020

1 2 UNIVERSIDADE D COIMBRA 9 0

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

Resilience in the Internet of Things for Smart City Applications

David Alejandro Perez Abreu

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

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

December, 2020

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

Resiliência na Internet das Coisas para aplicações de Cidades Inteligentes

David Alejandro Perez Abreu

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

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

Dezembro, 2020

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) under the projects grants MITP-TB/CS/0026/2013, FCT/13263/4/8/2015/S; as well as under the PhD grant SFRH/BD/117538/2016. Additionally by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under the project grant CAPES-FCT/8572/14-3.

Cofinanciado por:

Acknowledgments

By the end of this journey, I look back and I realize that many people, in one way or another, encouraged me to move forward in the research and academic path, and I could not miss this opportunity to thank them formally.

I would like to thank my esteemed advisors Prof. Edmundo Monteiro and Prof. Marilia Curado for their invaluable supervision, support, and advice during the course of my PhD degree. Both of you are not only excellent researchers but also splendid human beings. I am delighted to have chosen you both as my advisors and even more glad to consider you friends.

My gratitude extends to all the staff of the Center for Informatics and Systems of the University of Coimbra, which were always available to help me with academic or administrative stuff. From the academic perspective, I thank Prof. Luís Paquete for his time to discuss methods and approaches that were out of my knowledge scope at the beginning of my PhD. Regarding the administrative perspective, I have to thank Jorge Avila, who supported the paperwork for publications.

I want to thank my friends, lab mates, colleagues, and research team for a cherished time spent together in the lab and social setting. Moreover, special recognition to the sixth-floor LCT posse (Ricardo, Mariana, David Lima, Marcelo, Duarte, Vitor, Rui, Proença, Ngombo, Paulo, Tina) for making me feel accepted and part of a group; thanks a lot dudes.

To my grandparents, parents, and brothers for their unconditional support and understanding.

Karima, there are not enough words to describe the things we have spent together. Thank you for being there to help and support me. We make an incredible team.

Abstract

NOWADAYS companies and governments are using Information and Communication Technologies as tools to deploy their services and make them accessible to citizens; in order to expand urban resource efficiency with a low environmental impact and contributing with the development of the economy. This trending is known as the Smart City paradigm and it has taken advantage of the Cloud to Internet of Things continuum to provide communication and collect a massive amount of data. This scenario permits that the data can be processed and analyzed by applications to support smart services, enabling heavy calculations that run inside powerful data centers in the Cloud.

The Cloud to Internet of Things not only offers power to process data, but also makes it possible to use virtualization technologies to enhance how the object heterogeneity could be managed. Services like eHealth, smart traffic control, and smart home applications are composed of different functions that can be virtualized over physical hardware components within the network landscape. These Virtual Functions are grouped in a structure called Service Chains that fulfill particular smart service requirements, enabling a new broader set of smart end-user applications. The Cloud to Internet of Things continuum infrastructure provides communication, processing, and storage support for these applications. However, this complex, heterogeneous, and distributed landscape requires orchestration and management mechanisms in order to guarantee their proper functioning, especially in the face of failures.

One particular factor to manage is the resilience to provide service availability even in the event of failures. Automated proactive solutions to enhance the survivability of Service Chains when failures occur have to be considered within the orchestration solution for the Cloud to Internet of Things continuum. This research proposes an architecture and a set of mechanisms to orchestrate, formalize, and embed a collection of service requests for chaining Virtual Functions jointly to fulfill specific requirements of applications while enhancing their resilience.

The first contribution of this work is a resilience architecture for the deployment of Internet of Things services and applications in Smart Cities. The architecture proposed takes advantage of virtualization techniques to deal with the heterogeneity in the Cloud to Internet of Things continuum, as well as to provide a set of components focused on improving the resilience of applications. Specifically, the Resilience Manager module in the architecture implements a set of functions to enhance the availability of applications in case of failures. The architecture proposed worked as an inspiration for the design and implementation of an ontology to describe the Internet of Things infrastructure, to standardize how the information of the underlying components is exchanged.

The second set of contributions of this work are focused on the Resilience Manager module of the proposed architecture. In detail, a framework to address the composition and embedding of Service Chains in the Cloud to Internet of Things continuum with support of replicas to increase the availability of applications was designed and implemented.

The composition element in the framework is tackled via a formal grammar that enables the description of customized applications, allowing the definition of replicas for their components. In a second step, once the applications were specified, a Pareto analysis is used to optimize the selection of the components of the applications according to a given set of goals.

Regarding the embedding of Virtual Functions in the substrate infrastructure, three contributions are proposed: (1) an Integer Linear Programming model that prioritizes the use of nodes with higher availability; (2) a genetic algorithm that uses a fitness function that combines node availability, use of disjoint nodes, and the tiers to which the nodes belong to; and (3) a heuristic to handle more complex scenarios by taking advantage of the multi-tier scenario comprising the Cloud-Fog-Mist-Internet of Things.

The embedding mechanisms proposed in this work were evaluated via simulation. The assessment included measurements of failure rate, node utilization, and response time of the Service Chains embedded in the subtracted infrastructure. Simulation results show that it is possible to increase the resilience of chained Virtual Functions, while balancing the load of the infrastructure nodes.

Keywords: Internet of Things, Cloud, Smart City, Resilience, Service Chains, Embedding.

Resumo

ATUALMENTE, governos e empresas estão a usar Tecnologias de Informação e Comunicação como ferramentas para disponibilizar os seus serviços e torná-los acessíveis aos cidadãos; de forma a expandir a eficácia dos seus recursos urbanos, com pouco impacto ambiental e contribuindo para o desenvolvimento da economia. Esta tendência é conhecida como o paradigma Cidades Inteligentes e tem tirado partido do continuum Nuvem-Internet das Coisas para proporcionar comunicações e recolher dados em larga escala. Este cenário permite que os dados sejam processados e analisados em aplicações que suportam os Serviços Inteligentes, tornando possível correr cálculos pesados dentro de grandes centros de dados na Nuvem.

A Nuvem não somente oferece poder computacional para processar dados da, e para a, Internet das Coisas, como também torna possível o uso de tecnologias de virtualização que melhoram a forma como a heterogeneidade dos dispositivos pode ser gerida. Serviços como saúde eletrónica, controlo de trânsito inteligente, e aplicações para casas inteligentes são compostos por diferentes funcionalidades que podem ser virtualizados em componentes de hardware físico, dentro do ambiente da rede. Estas Funções Virtuais são agrupadas em estruturas denominadas Cadeias de Serviço. As Cadeias de Serviço são responsáveis por cumprir requisitos específicos de serviços inteligentes, possibilitando um novo conjunto de aplicações inteligentes para os utilizadores finais. A infraestrutura do continuum Nuvem-Internet das Coisas proporciona comunicações, poder computacional e armazenamento para suportar estas aplicações. Contudo, este ambiente distribuído, complexo e heterogéneo necessita de mecanismos de orquestração e gestão de forma a garantir o seu correto funcionamento, especialmente em caso de falhas.

Um fator particular a ter em consideração é a resiliência, de modo a que um serviço se mantenha disponível, incluso em caso de falhas. Soluções proativas e automáticas para melhorar a capacidade de sobrevivência de Cadeias de Serviço, quando ocorrem falhas, têm de ser consideradas na orquestração do continuum Nuvem-Internet das Coisas. Esta tese propõe uma arquitetura e um conjunto de mecanismos para orquestrar, formalizar e incorporar um conjunto de pedidos de serviços para agrupar Funções Virtuais, de forma a cumprir requisitos específicos das aplicações e ao mesmo tempo melhorar a sua resiliência.

A primeira contribuição deste trabalho é uma arquitetura para a implantação de serviços de Internet das Coisas e aplicações em Cidades Inteligentes. A arquitetura proposta tira partido das técnicas de virtualização para lidar com a heterogeneidade no continuum Nuvem-Internet das Coisas, e ainda providencia um conjunto de componentes focados em melhorar a resiliência das aplicações. Especificamente, o módulo Gestor de Resiliência da arquitetura implementa um conjunto de funções para melhorar a disponibilidade das aplicações na presença de falhas. A arquitetura proposta serviu ainda de inspiração para projetar e implementar uma ontologia para descrever a infraestrutura da Internet das Coisas, de modo a uniformizar a maneira como a informação é comunicada entre os componentes subjacentes.

O segundo conjunto de contribuições deste trabalho foca o módulo Gestor de Resiliência da proposta arquitetura. Nomeadamente, foi projetada e desenvolvida uma framework para abordar a composição e incorporação de Cadeias de Serviços no continuum Nuvem-Internet das Coisas com suporte de réplicas para aumentar a disponibilidade das aplicações.

Considerando a incorporação das Funções Virtuais no substrato da infraestrutura, são propostos três mecanismos: (1) um modelo de Programação Linear Inteira que prioriza a utilização de nós com alta disponibilidade; (2) um algoritmo genético que usa uma função de adequação que combina a disponibilidade dos nós, uso de nós não adjacentes e a camada a que os nós pertencem; (3) uma heurística para cenários mais complexos, que tira partido do ambiente multicamada Nuvem-Névoa-Neblina-Internet das Coisas.

Os mecanismos embebidos propostos neste trabalho foram avaliados através de simulações. A avaliação inclui medições da taxa de falha, utilização dos nós e o tempo de resposta das Cadeias de Serviço embebidas no substrato da infraestrutura. Os resultados das simulações demonstram que é possível aumentar a resiliência de cadeias de Funções Virtuais, ao mesmo tempo que se equilibra a carga da infraestrutura dos nós.

Palavras-chave: Internet das Coisas, Computação em Nuvem, Cidades Inteligentes, Resiliência, Cadeias de Serviço, Incorporação.

Foreword

THE work detailed in this thesis was accomplished at the Laboratory of Communications and Telematics (LCT) of the Center for Informatics and Systems of the University of Coimbra (CISUC), within the context of the following projects and grants:

SusCity - Sustainable Cities; financed by Foundation for Science and Technology within the scope of the project (MITP-TB/C S/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. The work performed in this project was related to the design of a trustworthy Information and Communication Technologies infrastructure for Smart City services, including the design of a resilient Internet of Things architecture for Smart Cities.

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 Cloud to Internet of Things continuum requirements, the design of the orchestrator architecture, and the design and development of the embedding framework for Service Chains and their Virtual Functions.

PhD grant - Foundation for Science and Technology (FCT) (SFRH/BD/117538/2016).

The outcome of the design, experiments, and assessments of several mechanisms on the course of this thesis resulted in the following publications:

Journal papers:

- Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2017b). A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, 72(1):19–30;
- 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; and
- 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.

Conference papers:

- Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2015). Resilience in iot infrastructure for smart cities. In *2015 Cloudification of the Internet of Things (CIoT)*, pages 1–4, Paris, France. IEEE;
- Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2017a). 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;
- Perez Abreu, D., Velasquez, K., Pinto, A. M., Curado, M., and Monteiro, E. (2017c). 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; and
- Perez Abreu, D., Velasquez, K., Miranda Assis, M. R., 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.

Book chapters:

- Curado, M., Madeira, H., da Cunha, P. R., Cabral, B., Perez Abreu, D., Barata, J., Roque, L., and Immich, R. (2019). Internet of Things. In: Cyber Resilience of Systems and Networks. In *Cyber Resilience of Systems and Networks*, chapter 2, pages 381–401. Springer International Publishing.

Co-advisor of MSc Thesis:

- Abade, B. (2018). *Context-Aware Improved Experiences in Smart Environment.* Msc thesis, University of Coimbra, DEEC.

Cooperation papers:

- Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2015). Towards latency mitigation in emergency scenarios. In *2015 Cloudification of the Internet of Things (CIoT)*, pages 1–4, Paris, France. IEEE;
- 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;
- Fernandes, J., Perez Abreu, D., Velasquez, K., Monteiro, E., and Martins, A. (2017b). An architecture to support affordable internet of things applications: The suscity project case study. In *8th Congresso Luso-Moçambicano de Engenharia V Congresso de Engenharia de Moçambique (CLME2017 - V CEM)*, pages 1055–1056, Maputo, Moçambique. INEGI/FEUP;
- Velasquez, K., Perez Abreu, D., Gonçalves, D., Bittencourt, L., 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;

- Fernandes, J., Perez Abreu, D., Velasquez, K., Mateus, M., ao Carrilho, J., Silva, M., Monteiro, E., and Martins, A. (2017a). Building a smart city iot platform - the suscity approach. In *48nd Spanish Congress on Acoustics and the Iberian Encounter on Acoustics (TECNIACUSTICA)*, pages 1–9, Coruña, Spain. Sociedad Española de Acústica (SEA);
- Abade, B., Perez Abreu, D., and Curado, M. (2018). A Non-Intrusive Approach for Indoor Occupancy Detection in Smart Environments. *MDPI - Sensors*, 18(11):1–18;
- Velasquez, K., Perez Abreu, D., Assis, M. R., Senna, C., Aranha, D. F., Bittencourt, L. F., Laranjeiro, N., Curado, M., Vieira, M., Monteiro, E., et al. (2018). Fog orchestration for the internet of everything: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 9(1):1–23;
- 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*, pages 64–72, Paris, France. IEEE; and
- 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.

In parallel with the execution of the tasks related to this thesis, participation in different projects during this PhD course led to discussions and exchange of ideas that resulted in the publications listed as cooperation papers, which are framed within the context of this research and enriched the work performed. For these outcomes, the participation was focused on sharing knowledge of the Cloud to Internet of Things continuum and the approaches to enhance the resilience in this scenario. For all the cooperative papers the participation involved, besides the discussion and conception of ideas, the design and implementation of the experiments, and the analysis of results.

Contents

List of Figures

List of Algorithms

List of Tables

Acronyms

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
BNF	Backus-Naur Form
BNG	Broadband Network Gateway
CC	Chain Composition
CD	Community Detection
CoAP	Constrained Application Protocol
CR	Cognitive Radio
DACOM	Data Compression
DAGG	Data Aggregator
DANA	Data Analytic
DB	Data Base
DES	Discrete-Event Simulation
DPI	Deep Packet Inspector
E2E	End-to-End
ETSI	European Telecommunications Standards Institute
FCT	Fluid Communities by Tiers
FluidC	Fluid Communities
FF	First Fit
FW	Firewall
GA	Genetic Algorithm
GENSEN	Generic Sensor
ICT	Information and Communication Technologies
IDPS	Intrusion Detection Prevention System
IDS	Intrusion Detection System
ICN	Information Centric Network
IETF	Internet Engineering Task Force
ILP	Integer Linear Programming

IP	Internet Protocol
IT	Information Technology
IoT	Internet of Things
M2M	Machine-to-Machine
MANO	Management and Orchestrator
MDT	Mean Down Time
MOEA/D	Multi-Objective Evolutionary Algorithm based on Decomposition
MQTT	Message Queuing Telemetry Transport
MTTF	Mean Time To Failure
MUT	Mean Up Time
NAT	Network Address Translation
NF	Network Function
NFV	Network Function Virtualization
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
QoR	Quality of Resilience
QoS	Quality of Service
RPL	Routing Protocol for Lossy networks
SC	Service Chain
SDN	Software-Defined Networking
SFC	Service Function Chaining
SIoT	Social Internet of Things
SPARQL	SPARQL Protocol and RDF Query Language
TM	Traffic Monitoring
VF	Virtual Function
VF-CC	Virtual Function - Chain Composition
VM	Virtual Machine
VN	Virtual Network
VNF	Virtual Network Function
VNE	Virtual Network Embedding
VOC	Video Optimization Controller
VS	Virtual Sensor
VSDN	Virtual Software Defined Network

WAN	Wide Area Network
WiFi	Wireless Fidelity
WOC	WAN Optimization Controller
WSGA	Weighted Sum Genetic Algorithm
WSN	Wireless Sensor Networks
YAFS	Yet Another Fog Simulator

Chapter 1

Introduction

Contents

NOWADAYS, technology is part of our lives that the dependency on its benefits is growing faster than ever. With the arrival of the paradigms of Smart Cities and the Internet of Things (IoT), citizens are able to improve their quality of life. Given that sensors and actuators deployed in Smart Cities usually have limited resources, today, it is a common practice to use Cloud computing to extend the scope and benefits of Smart Cities. Taking into consideration that communication between applications and devices is vital for a good performance of services in a Smart City, it is necessary to design new architectures and mechanisms to provide reliability in communications. A key aspect that has to be addressed by the new communications approaches is the possibility to recover the network and its services in case of faults, without human intervention. This chapter provides the background and motivation of this research which is aimed at improving the resilience of applications deployed into the Cloud to IoT, besides presenting a discussion about the objectives and contributions of this thesis.

1.1 Background and Motivation

The Smart City paradigm emerged to describe the use of new technologies in everyday urban life including Information and Communication Technologies (ICT), as well as incorporating other aspects present in urban scenarios such as modern transportation, education, and public safety. In these environments, the communication endpoints are often special devices, like sensors and actuators (Smart Objects), that need to exchange data to coordinate their operations. Specifically, the interaction between these Smart Objects and their services/applications define the IoT; which is the interconnection of embedded computing devices (e.g., sensors and actuators) using the Internet infrastructure and the services/applications offered to end-users in order to improve their daily lives.

The convergence between devices and services in Smart Cities is only possible with good end-to-end network and computational resources quality; this could be achieved with a Cloud computing infrastructure capable of processing resource-demanding applications and a Fog approach to deliver real-time services in the last-mile, even in the worst scenarios. The Cloud computing paradigm adoption by network operators and service providers has been massive, given its benefits in cost-savings, enhancement in work and management response, business agility, and Quality of Service (QoS) [Marston et al., 2011]. Despite the initial success of Cloud adoption, in recent years, there has been a tendency shift to bring computational resources and services towards the edge of the network to fulfill the requirements of emerging paradigms such as the IoT. In this scenario, there is a larger scale of heterogeneous devices and lower latency that could represent a significant challenge for the traditional Cloud environments [Velasquez et al., 2018].

The Fog emerges as a solution to improve Cloud-based services by offering a distributed and federated compute model to decentralize the deployment, management, and orchestration of services and applications across the entire network infrastructure. Thus, the Fog computing paradigm lays on a tiered model that enables ubiquitous-access scalable computing resources. This model aids the placement of context-aware services and applications in computational nodes, which are set between smart end-devices and centralized Cloud systems. As soon as the Fog computing paradigm was adopted, the use of geographically dispersed, low-latency computational resources increased the need for more specialized and dedicated nodes closer to the end-users; thus, the concept of Mist computing emerged. The Mist nodes are deployed in the Mist computing layer which resides in the peripheral of the network infrastructure, even closer to the end-users [Iorga et al., 2018].

In a Smart City scenario, many services are designed to improve the quality of life of the citizens, for example, urban traffic control, emergency health assistance, home energy monitoring, and evacuation routes in case of natural disasters; turning the access to these services into a critical aspect. The Fog can be used to enable a fresh breed of services within the scope of the Smart Cities, such as smart traffic lights and train maintenance, both of which require high levels of availability and low levels of latency. Even more, augmented reality applications for guided tours in cities, could take advantage of the computation power and lower latency provided by the Fog and Mist.

The Cloud-Fog-Mist-IoT scenario is composed of a set of hardware and software components organized in tiers, going from the Cloud in the top, through the Fog and Mist, to the IoT in the bottom, allowing the interconnection of the Smart City services. This landscape has been heavily influenced by virtualization technologies, such as Network Function Virtualization (NFV) [Yi et al., 2018], Software-Defined Networking (SDN) [Kreutz et al., 2015], and Service Function Chaining (SFC) [Gupta et al., 2018], that enable the design, development, management and deployment of network functions [Wright et al., 2015] using the available hardware and software resources in the infrastructure. The same approach can be used transversely in the *Cloud to IoT continuum* to provide End-to-End (E2E) services.

End-to-End services and applications in the Cloud to IoT continuum can be described by a VF Forwarding Graph that links the endpoints through a set of interconnected VFs, called Service Chains (SCs). The landscape where the SCs are deployed lies on an extremely diverse substrate infrastructure composed of information and communication devices (i.e., Cloud nodes, Fog nodes, Mist nodes, Sensors, Actuators) and links (i.e., wired and wireless channels), which have to be orchestrated to host a plethora of services and applications with different performance as well as functional requirements. In such a complex scenario, resilience becomes a key factor to orchestrate and guarantee the continuity of the services and applications even in the face of failures. One possibility to increase the resilience level of the services is to use replicas; such that, in the event of a node failure, the replica can be activated and the service will be able to maintain its availability [Schöller and Khan, 2015].

The reliability and availability of the E2E services/applications are based on the behavior of their functional blocks (i.e., VFs and their communication links) [Nakamura, 2016]. Therefore, when replicas are considered as a resilience mechanism for Service Chains, it is possible to apply two methods: (1) the replication is applied to the entire chain, or (2) the replication is applied to one or more VFs along the chain. Besides the method used during the replication phase, it is also essential to consider where to place the replicas (e.g., deploy the primary and backup VF in different nodes to avoid that in the case of a failure in said node, the replica can be activated) and how to formally represent the SC requests considering the possibility of having replicas.

The service infrastructure provider is responsible for the orchestration, composition, embedding, and management of these SCs so that they can be instantiated to satisfy end-user requirements. Consequently, there is the need to formalize the tasks required from the service infrastructure provider, including the guidelines for resilient support to the composition and embedding of SCs in the Cloud to IoT continuum to satisfy the Smart City services and applications requirements.

Taking into consideration the discussion presented so far, in the next section the objectives and resulting contributions to help with the current issues in this context are shown. For the rest of this thesis, when the term Cloud to IoT infrastructure is used, it encompasses computing and networking elements from the substrate infrastructure.

1.2 Objectives and Contributions

The main goal of this research is to improve the resilience of Cloud to IoT continuum by designing and developing new mechanisms that prevent failures in services and applications, avoiding the disruption of computational and communication elements. To achieve this goal, the following objectives have been established:

- Define an architecture to enable the Cloud to IoT continuum to reach high levels of resilience on the infrastructure and service levels;
- Define a group of strategies that can be applied to improve the resilience levels of the Cloud to IoT continuum infrastructure for smart services;
- Design a set of mechanisms that implement the strategies aimed at the enhancement of the resilience of the Cloud to IoT continuum;
- Determine a set of metrics that take into consideration the resilience of the Cloud to IoT continuum infrastructure that guide the proposed mechanisms; and
- Validate the mechanisms and strategies proposed.

These objectives were designed in an incremental way to answer the question: how to improve the resilience in the Cloud to IoT continuum? A thorough revision of the state of the art is performed on the topic to provide deep knowledge about the scenario where the mechanisms and strategies are implemented. Then, a set of resilience metrics is selected to determine the efficiency of the mechanisms and strategies. This scenario is inspired by the Smart City paradigm that provides the overall context for this research work. Smart communication infrastructure virtualization and replication mechanisms are designed and developed using the scenario previously identified.

The hypothesis for these mechanisms is based on the fact that using disjoint nodes for the embedding of replicas and the different tiers in the Cloud to IoT continuum could enhance the resilience of the SCs.

Taking into consideration the goals described above, this thesis has produced the following contributions:

- **Contribution 1, A Resilience Architecture for the Cloud to IoT Continuum.** To deal with the intricacy and heterogeneity of the Cloud to IoT continuum, an architecture aimed to enhance the resilience of services and applications is described in Chapter 3. From this architecture, it is important to point out two significant outcomes, the *Resilience Manager* module which implements the embedding mechanisms proposed and validated in Chapters 5 and 6 respectively, as well as the ontology designed and implemented to describe an IoT infrastructure;
- **Contribution 2, A Framework for the Composition and Embedding of SCs.** One of the main contributions of this thesis lays in a standard approach to efficiently handle requests for virtualized services while improving the availability of Cloud to IoT continuum services/applications. Thus, Chapter 4 presents a composition and embedding framework designed and implemented to fulfill this goal. The composition element of the framework uses a formal grammar to validate the requests before performing a Pareto analysis focused on optimizing the selection of application components according to a given set of goals (see Sections 4.2 to 4.4);
- **Contribution 3, A Conceptual and Technical Review of Simulators for the Cloud to IoT Continuum.** The selection of a proper tool for the validation and assessment of approaches and mechanisms is a key aspect of scientific research. For the Cloud to IoT continuum scenario, there is a set of emulators and simulators that could be used; however, which is the more adequate for a particular research? In Chapter 6, Section 6.1 an analysis of a set of simulators for Cloud/Fog environments is presented, describing their main characteristics and offering a comparison among them intending to enlighten the decision process for the selection of the proper simulation tool;
- **Contribution 4, A Mathematical Model for the Embedding of VFs to Enhance Resilience via Replication.** An optimal solution for the embedding of VFs in the Cloud to IoT continuum aimed to improve availability was the first step to studying this embedding problem defining an upper bound (see Section 5.1). The mechanism uses a bi-level

Integer Linear Programming (ILP) formulation. On the first level, the goal is to maximize the acceptance rate; and the second one is focused on maximizing survivability by placing the VFs in the most reliable nodes. The limitations found in this approach regarding the computational cost motivated the next two mechanisms;

- **Contribution 5, A VFs Embedding Mechanism based on a Genetic Algorithm (GA) aimed to Improve Resilience Using a Weighted Sum of Metrics.** The conceptual definition, implementation, and assessment of an embedding mechanism that uses a GA approach is another contribution of this thesis (see Section 5.2). This mechanism uses a fitness function that combines node availability, use of disjoint nodes, and the infrastructure tier to which the node belongs in the decision making; and
- **Contribution 6, A VFs Embedding Heuristic Mechanism based on Graph Partition to Increase Resilience.** The proposal, design, and evaluation of a heuristic-based mechanism to enhance the resilience of SCs and their VFs is one of the contributions of this thesis (see Section 5.3). The mechanism uses a Fluid Communitiess approach for graph partition; specifically, the VFs are to be embedded in communities performing a vertical search in the Cloud to IoT continuum to find the optimal node considering its availability, to improve response times for the SCs.

Having discussed the goals and contribution of this research, the next section shows the outline of the thesis.

1.3 Outline of the Thesis

The remainder of this thesis is organized into seven chapters. Chapter 2 offers the research context, describing the Cloud to IoT continuum and introducing the concept of resilience in this environment. The chapter also presents possible solutions to improve the resilience in the Cloud to IoT continuum and shows an analysis of the existing approaches to handle this issue.

Chapter 3 introduces an architecture to improve the resilience of smart services in the Cloud to IoT continuum outlining the functions of its different modules and their interaction. Furthermore, the chapter presents an ontology for the description of the IoT infrastructure.

Chapter 4 talks about the softwarization in the Cloud to IoT continuum and the need of a standard way to represent the composition of VFs in SCs. A solution for the composition of SCs is provided in the design of a formal grammar for the representation and verification of SCs that allows the specification of the tiers in which to embed the VFs as well as the number of desired replicas.

Chapter 5 presents three different embedding mechanisms for the SCs formally defined in Chapter 4. The mechanisms include an ILP based approach, a GA approach, and a heuristic based in graph partition.

Chapter 6 shows an analysis of different Cloud/Fog simulation tools that help in the selection of the proper tool, to then describe the evaluation setup for the embedding mechanisms. Results from simulations are also analyzed in this chapter, including the SCs failure ratio, the infrastructure node utilization, and SCs response time.

Chapter 7 concludes this document, offering a synthesis of the thesis and a projection on possible research paths for extending this work.

Chapter 2

Resilience in the Cloud to Internet of Things Continuum

Contents

In computing and communication infrastructures, it is very important to take into consideration how to deal with possible faults to keep alive the components that allow maintaining the continuity of the services and applications. With this in mind, it is not only necessary to pay attention to the normal infrastructure operation state, but also to failure situations. The different failure situations are specified by the availability status of the links and nodes in the substrate infrastructure, as well as by elevated volumes of traffic related to a particular service/application. Thus, management and orchestration mechanisms are required to guarantee the proper functioning of services and applications that lay under the substrate infrastructure of service providers.

Since the communication infrastructure in the Cloud to IoT continuum plays a crucial role in nowadays services/applications, the resilience of this infrastructure becomes critical to achieve higher levels of availability of services and applications. The term *resilience* has been included in different scientific fields as a possible solution to manage unexpected behaviors and challenging circumstances, and it is used to indicate how a system responds to changes from inside or outside itself. This chapter introduces the concept of resilience in the Cloud to IoT continuum and some techniques based on virtualization used to improve it. A review of the work done to enhance the resilience in the Cloud to IoT continuum is also performed to identify open issues in this field of study with a discussion focused on the context of IoT and Smart Cities.

2.1 Understanding the Cloud to IoT Continuum

To bring to reality the services and applications encompassed by the Smart City ecosystem, a strong ICT infrastructure must provide communication, storage, and computing support [Hou et al., 2016]. “A Smart City is a system that enhances human and social capital wisely using and interacting with natural and economic resources via technology-based solutions and innovation to address public issues and efficiently achieve sustainable development and a high quality of life on the basis of a multi-stakeholder, municipally based partnership” [Fernandez-Anez, 2016]. The IoT enables the interaction of common intelligent objects to collect and analyze massive amounts of data; on the other hand, the Cloud provides the large storage and high computational capabilities required to deal with this scenario.

The Cloud computing paradigm is adopted in order to enable ubiquity, convenient, and on-demand access to the services in a Smart City scenario. The Cloud provides a pool of resources (i.e., network, storage, compute) that can be provided via the orchestration of the resources via the service provider [Peter Mell, 2011]. The Cloud offers several characteristics, such as broad network access, elasticity, and resource pooling. However, large physical distances and extensive data to process and transport make the Cloud suffer from some drawbacks such as large end-to-end delay, traffic congestion, and communication costs [Mukherjee et al., 2018]. In more recent years, and with the advent of newer services and applications, especially with the emergence of the Smart City paradigm, these and other different challenges arise [Chiang and Zhang, 2016]. The technological developments have brought the above mentioned tasks (i.e., storage, computing, and management) closer to the end-user in order to provide lower latency, mobility support, and location awareness. Fog computing is the more recent paradigm proposed to complement the use of the Cloud and harbor these newer requirements [Dastjerdi and Buyya, 2016]. The Fog is meant to enable the use of computational resources near IoT sensors/actuators facilitating local storage and preliminary data processing, thus reducing network traffic and response times and, ultimately, decision making.

For more specialized and dedicated nodes with lower capacities than the Fog nodes emerges the paradigm of Mist computing. The Mist is also referred to as a *lightweight Fog*, and is located even closer to the peripheral of the network, bringing the Fog closer to the smart devices in the IoT [Iorga et al., 2018]. Thus, the resource pool decreases its capacity while going down in layered network infrastructure, from the Cloud, Fog, Mist, and ultimately the IoT layer. The IoT connects objects around end-users to provide seamless communication and contextual services provided by them. The main idea behind the IoT is to materialize the interaction and cooperation of smart objects to make services better and accessible anytime, anywhere [Lee et al., 2012].

Figure 2.1 depicts the scenario discussed above. On the top layer, large datacenters offer access to extensive resources (i.e., storage, network, compute) to provide ubiquity and elasticity to the Smart City services/applications. On the lower layer, the Fog nodes offer smaller resource capacity but with the advantage of pre-processing data to reduce the network congestion, as well as reducing the response time to end-users. The following layer, comprised of the Mist nodes, offers an even smaller resource pool in the immediate vicinity of the end-user, in the IoT layer, where a set of smart objects (i.e., sensors and actuators) provide smart services that include sensing applications to control lighting and temperature in a smart home or control of traffic lights in smart transportation services.

The impact of the Cloud to IoT continuum in the development of novel services and applications for Smart Cities will depend in large part on the efficiency of the orchestration mechanisms specifically designed to manage such a complex scenario [Barcelo et al., 2016]. Accordingly, it is necessary to design, develop, and implement procedures and mechanisms to exploit the characteristics of the edge of the communication infrastructure, and paying special attention to maximizing the availability of services and applications, even in case of failures.

2.1.1 Virtualization for the Cloud to IoT Continuum

For decades, services and communication infrastructures were operated directly over the bare metal of a set of devices designed to perform specific tasks (e.g., servers, routers, firewall); however this approach has shifted from hardware-

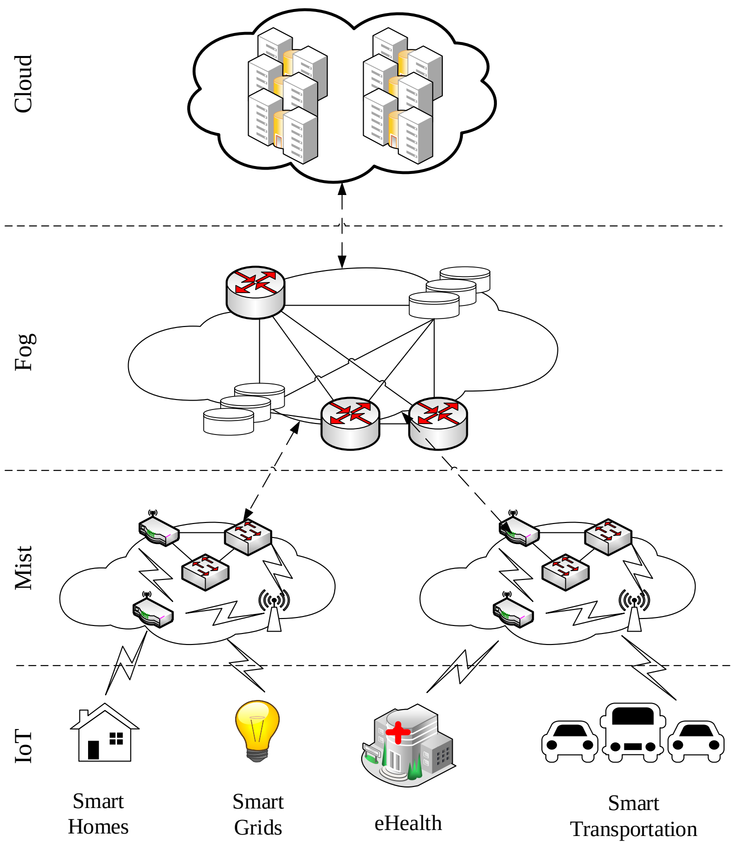


Figure 2.1: The Cloud to IoT Continuum.

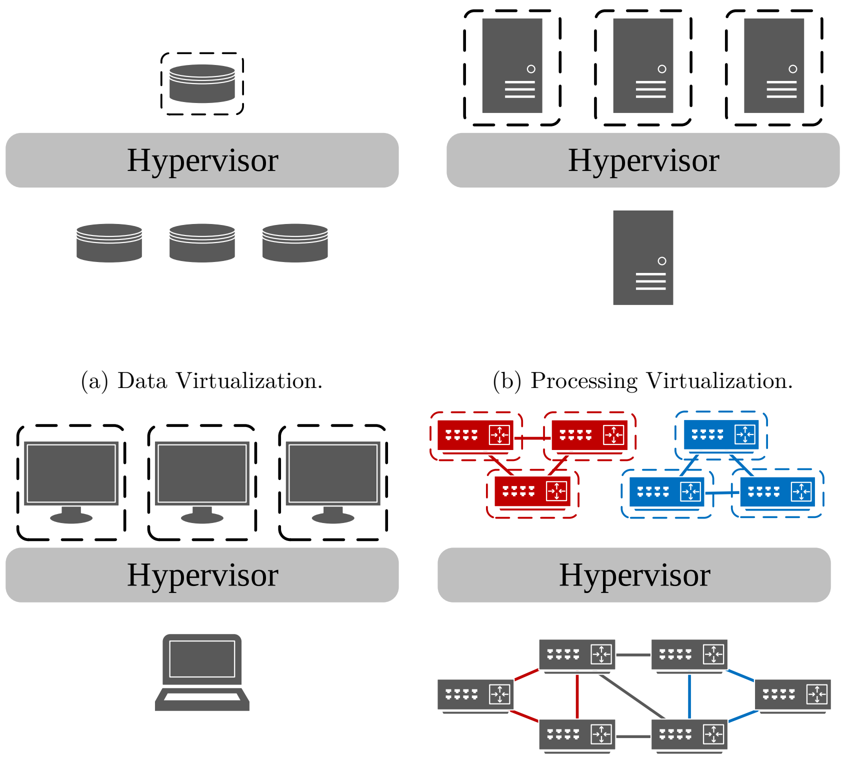
based to software-based mode thanks to virtualization technologies which paved the way to replace physical equipment by software capable to do the same functions [Pujolle, 2020]. Virtualization is a technology that allows the creation of software-based, or virtual, functions to fulfill particular tasks.The key component of the virtual approach lays in a layer of software separated from the physical resources usually called *Hypervisor*, which is in charge of managing and dividing the physical resources and exposing them to the virtual environments at upper layers [Uhlig et al., 2005].

Using virtualization technologies, resources are partitioned as required from the physical environment to the virtual ones. Users interact with and execute their tasks or functions in these isolated virtual environments. When an user, service or application issues an instruction or task that needs additional resources from the physical environment, the *Hypervisor* transfers the request to the physical system and manages the changes. The key properties of virtualization are [Uhlig et al., 2005; Pujolle, 2020]:

- Partitioning: enables running different functions over the same physical device, distributing the physical resources;
- Isolation: facilitates detachment of failures and security risks to a single virtual instance, preserving the performance of other virtual instances that use the same physical device;

- Encapsulation: allows to save the entire state of a virtual instance into a file that can be moved or copied to another physical device; and
- Hardware independence: allows the migration or instantiation of any virtual instance to any physical device.

Virtualization can be divided into four types, according to the functionality or resource that is being virtualized, and are depicted in Figure 2.2. Figure 2.2a shows the data virtualization. Data spread across the storage infrastructure can be seen as a consolidated single source, providing processing capabilities that can include new data sources to the pool and applying any data transformation required by the end-user.



(a) Data Virtualization. (b) Processing Virtualization.

(c) Operating System Virtualization. (d) Network Virtualization.

Figure 2.2: Types of Virtualization.

Figure 2.2b illustrate the processing of CPU virtualization. This process enables the use of a single central manager that deploys simulated processing environments, either desktop or server, with the additional advantage of allowing mass configuration, updates, and security checks. Particularly in the case of server virtualization, computers designed for more specific tasks (unlike the general purposes desktops) involves the partitioning of physical resources, so the components can be distributed to serve multiple functions. Figure 2.2c depicts the operating system virtualization. This virtualization type allows the use of different operating systems side-by-side over the same computer, virtualizing at the kernel level. As consequence, the security is increased, since all virtual instances can be monitored and isolated, also reducing the time needed from Information Technology (IT) teams for administrative tasks, such as system updates.

Figure 2.2d describes virtualization at the network level, also called NFV, which distributes key network functions (e.g., file sharing, IP configuration) along with the network environment. Since the software functions are independent of the physical devices where they are embedded, they can be packaged and assigned to different network environments. This approach reduces the number of physical devices (e.g., switches, firewalls, routers) needed to create multiple networks.

Next-generation communication infrastructures are being shaped by the softwarization of services and functions, as well as, the virtualization of physical resources. New approaches are being developed to manage the compute, storage, and networking resources; thus, application components are provisioned as virtual instances using different virtualization technologies in order to fulfill the requirements of the services and the capabilities of the communication infrastructure [Galis et al., 2014]. Technologies such as SDN [Kreutz et al., 2015], NFV [Yi et al., 2018], and SFC [Gupta et al., 2018] enable this landscape by complementing the Cloud to IoT continuum, facilitating elasticity of the services, migration when needed, and adaptability of general purpose hardware for specific functionalities required by applications such as Smart City services.

Virtualization also impacts the resilience levels of services, applications, and communication infrastructures. Service instances can be migrated, replicated, or instantiated online to recover from failures. A failing computing or network node can be instantiated in a different physical device recovering its tasks at the infrastructure. This opens a new door for resilience solutions for the Cloud to IoT continuum. The basic concepts of resilience and its metrics are presented in the following section.

2.2 Resilience in Communication Infrastructures

Despite the literature on network resilience being relatively broad, there is almost no consensus regarding resilience in the network management context. Some of the definitions available for resilience in this field include: (1) the ability to come back to normal conditions after the occurrence of a disruptive event, (2) the capability of a system to maintain its functions in the face of internal and external changes, (3) the ability of a system to absorb unpredictable change and still keep its essential functions, and (4) the aptitude of the system to withstand a significant disruption within admissible degradation parameters and to recover within an acceptable time and cost [Vugrin et al., 2010; Ahmadian et al., 2020]. For this reason, and before moving forward with the discussion, it is necessary to specify the resilience definition to adopt for this research.

Starting from the principle that the main aspect of resilience lays in the ability of an entity to recover from an external disruptive event, in this work, resilience in communication infrastructures is defined as a set of mechanisms to ensure service/application and networking robustness, by guaranteeing that resources and components are restored in case of failures. For the restoration, it is possible to use a proactive approach or protection (actions before a failure) and/or a reactive approach or restoration strategy (actions after a failure) to improve the availability of nodes and links of a communication infrastructure [Pioro and Medhi, 2004].

Reactive techniques do not pre-allocate resources for backup, instead they deal with failures once they take place. Using this approach might lead to slower reaction time for recovery, but would be less demanding with regards of resource consumption from the service provider perspective. Contrarily, proactive solutions pre-allocate backup resources to guarantee fast recovery of services/applications in case of failures of components in the communication infrastructure [da Fonseca and Boutaba, 2015].

From the viewpoint of the node failure, proactive approaches use backups or replicas of services that are allocated since the beginning of the lifetime of the application; while reactive approaches instantiate new replicas of services affected by failures after said failure occurs. For link failures, the resilience is focused on the path concept. A path is a logical communication provision identified by source and destination end-components, which is associated to a physical interface [Cholda et al., 2009]. Thus, the proactive approach or path protection is understood as reservation of resources at the time the flow on the path is set up. In path restoration (reactive approach), when a path is broken a Management and Orchestrator (MANO) module starts the restoration process by first calculating the backup path using the available resources and then re-establishing the flow.

Despite of the kind of failure recovery technique used, it is necessary to perform the following tasks [Papadimitriou and Mannie, 2006]: (1) *Fault Detection*, where an anomaly situation is identified; (2) *Fault Localization*, during this phase the point of failure is determined; (3) *Fault Notification*, here the failure is informed to the appropriate entity that has to triggering the recovery process; and (4) finally, the actions to manage the problems and restore the normal operation are executed in the *Recovery Switching* phase. The MANO module in the communication infrastructure is in charge to execute the previous tasks to trigger the proper actions to guarantee the resilience of services/application in case of any disruption meanwhile achieving the end-users requirements.

Taking into consideration the previous discussion, resilience (Res) denote a proportionality of the availability (Ava) and recovery (Rec) [Sousa, 2013], as per Equation 2.1.

$$Res = Ava * Rec \tag{2.1}$$

The main focus of resilience is to provide mechanisms to support continuity to systems and infrastructures (applications and network). However, when different resilience techniques are available, it is important to make a conscious decision about the proper technique regarding a particular set of requirements. In order to be able to make a choice in this context and also to compare the performance of the mechanisms to enhance resilience, it is necessary to define metrics to measure the behavior and results of the mechanisms that deal with resilience.

2.2.1 Resilience Metrics

To assess the best mechanisms to improve the resilience of services and application in the Cloud to IoT, resilience metrics are defined in this subsection. An important concept that has to be defined in this context is the Quality of Resilience (QoR). QoR is defined as the features of a communication infrastructure that affect the QoS perceived by users and are related to resilience [Cholda et al., 2007]. Considering this definition, it is possible to divide the metrics that measurement resilience according to their *reliability attributes*. A discussion about the *reliability attributes* is presented below [Cholda et al., 2007].

Continuity is the first reliability attribute and denotes the length of a period of time during which a service is not interrupted as a consequence of a failure. The Mean Time To Failure (MTTF) could be used to measure the continuity. MTTF is defined as the average duration of time from the time when a service request was received, assuming that the service was up at this time, until the service fails for the first time. In many systems, this metrics is estimated or approximated using the notion of Mean Up Time (MUT). Figure 2.3 shows a binary scenario of availability proposed by the ITU-T [ITU-T, 2009], where the system is considered available ($state = 1$) or unavailable ($state = 0$). Using this model, MUT corresponds to the moments where the system is in an available state, namely, where the model's function reaches the upper bound. Equation 2.2 represents a generic case of the MUT with n failures. In the absence of failures, Equation 2.3 is used since $tFail_n = 0$ and $tAvai_n = 1$.

$$MUT = (tFail_1 - tIni) + \sum_{i=2}^{n}(tFail_i - tAvai_{i-1}) + (tEnd - TAvai_n) \tag{2.2}$$

$$MUT = tEnd - tIni \tag{2.3}$$

Downtime is the second reliability attribute, and is related to the time period in which a service is inaccessible because of a failure in the communication infrastructure. The length of the interruption may be determined using Mean Down Time (MDT). MDT is the mean time duration from a service failure to the point when the service is recovered. In Figure 2.3, MDT is represented by the moment where the service is unavailable or reach the lower bound of the model's function. In a general case, MDT is determined by Equation 2.4. In absence of failures, the MDT is equal to zero ($MDT = 0$).

$$MDT = \sum_{i=1}^{n}(tAvai_i - tFail_i) \tag{2.4}$$

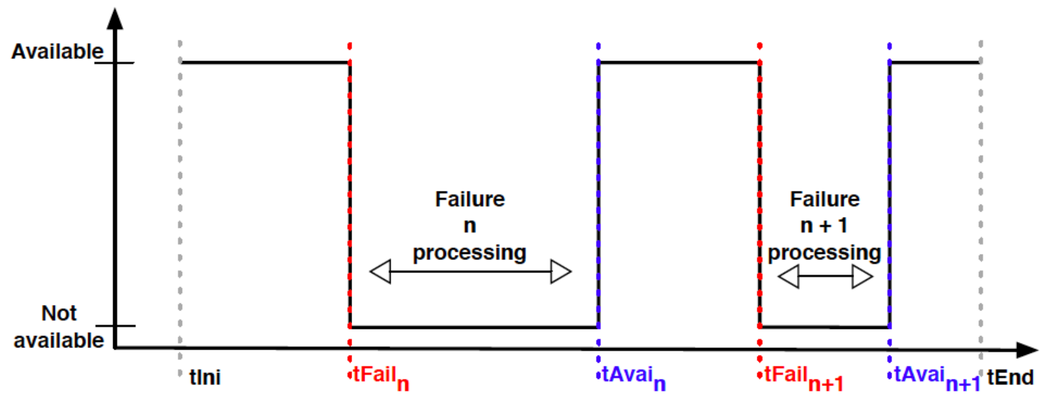


Figure 2.3: ITU-T E.800 Availability Model.
(source [ITU-T, 2009])

With the definitions of MUT and MDT, it is possible to describe the **availability** (Ava) of a system as a function of these two metrics. The ITU-T [ITU-T, 2009] proposes Equation 2.5 to relate the availability to the MUT and MDT. In the computing and communication environment, is most useful the concept of availability as steady-state, that may be understood as the probability of finding an item (i.e., object, device, network, and connection) in an operating state at any time that its service is required [Grover, 2003].

$$Ava = \frac{MUT}{MUT + MDT} \tag{2.5}$$

The definitions of resilience discussed so far are general enough to be applied with different methods and mechanisms with the respective adjustment. The metrics exposed (Ava, MUT, MDT) can be used to determine the performance of mechanisms aimed at improving the resilience of the Cloud to IoT continuum, which provide the communication infrastructure to end-user services/applications. Some resilience challenges in the Cloud to IoT continuum are described below.

2.2.2 Resilience Challenges in the Cloud to IoT Continuum

Service continuity is not only an end-user desire but often a regulatory requirement, as Cloud and communication providers are considered to be part of critical infrastructure that supports vital economic, government and citizens basic requirements; thus, there has to be a continuity in their services and applications. In the complex and diverse environment where the Cloud to IoT continuum acts, a seamless interaction between all the actors that build the infrastructure, from the physical (e.g., sensors, actuator, smart objects, links and nodes) to the logical perspective (e.g., service, applications, protocols), is a critical aspect. Even more, in this kind of scenario, the availability of the physical and logical devices and their services represent a key requirement, given that some critical applications such as assisted driving, augmented maps, and health monitoring require continuous availability while providing real-time feedback to end-users.

An improved connectivity between Cloud services and devices in the IoT is necessary to support the emerging applications that rely on this infrastructure. To deal with disruptions, it is required to have mechanisms that enhance the resilience both at infrastructure and service levels. Resilience has been defined as the ability of the communication infrastructure to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation [Sterbenz et al., 2013].

The following objectives from the resilience point of view have to be considered in this context:

1. It is necessary to ensure the availability of services and applications to achieve a successful end-to-end communication;
2. During the design and implementation of services and applications, it is necessary to define the expected availability of the constituent function blocks; and
3. A module or component (e.g., MANO instance) to detect and mitigate failures in the infrastructure is required.

To increase the resilience of smart objects that enable interaction with the physical world, replication and backup schemes must be implemented; however, how is it possible to efficiently adopt the aforementioned schemes? One traditional approach is using a primary and backup model, where devices and services are duplicated for robustness purposes. This would not be adequate, considering that this strategy could waste valuable resources in the already constrained Fog and Mist nodes in case no failure occurs. In this context, virtualization mechanisms have proven to be useful from the cost and operational perspectives.

Emerging virtualization paradigms like Containers [Pahl and Lee, 2015] and NFV [Matias et al., 2015] allow the improvement of the performance and availability of service and device components. Once mapped as a logical item, physical objects can be handled like any other piece of software, granting the possibility to apply migration, instantiation, and other well-known techniques over them. Thus, a failure concerning a service or device can be recovered by migrating or instantiating a logical object over a different physical device.

From the communication point of view, the traditional approach of distributed systems relies on trying to hide the distributed nature of the system to offer a perspective of a single machine. In Cloud/Fog environments, this "hiding" approach remains for the network layout considering that Cloud services just expose high-level information about their setup and distribution. On the other hand, in Fog scenarios it is essential to know about the network topology to take advantage of the geographical distribution which requires a more fine-grained topology abstraction. Thus, it is necessary to have an efficient and flexible way to control the route of the data and the topology of the communication infrastructure in the IoT.

Regarding the resilience at the communication infrastructure level, an approach in two phases could be applied using a detailed fine-grained topology abstraction at Cloud and Fog levels. In a first step, an offline mechanism to find disjoint paths between the components of the IoT could be executed to obtain backup paths that can be switched in case of failures. In a second stage, the detection of a failure and the migration of the data flows will be performed inline. To achieve this last task, the use of path-splitting and multipath routing strategies appears to be a feasible solution Perez Abreu et al. [2017b].

To guarantee a smooth work of the proposals mentioned above from the resilience perspective, an Orchestrator should be in charge of intelligent migration and instantiation of resources and services, providing a global view of the status of the Cloud to IoT continuum. Furthermore, the interaction between federative Clouds and services represents an additional challenge since the Orchestrator has to unify politics from different administrative entities smoothly. Some works have already been carried out to deal with resilience in the Cloud to IoT continuum. An analysis of said works is provided in the next section.

2.3 Addressing Resilience in the Cloud to IoT Continuum

The surplus and heterogeneity of devices in the Cloud to the IoT represent a new challenge for the mechanisms that manage the interaction and communication between smart objects. In order to satisfy these new demands, various research efforts have been carried out. This section shows a set of works related to the Cloud-Fog-Mist-IoT, which outcomes could be used to enhance the level of resilience in Cloud to IoT continuum. These works are grouped into three categories: (1) Managing the Cloud to IoT Continuum, focused on architectures to orchestrate the Cloud to IoT; (2) Connecting the Cloud to IoT Continuum, to address efforts regarding networking issues in this context; and (3) Embedding in the Cloud to IoT Continuum, concerning how to embed end-user requests in a substrate communication infrastructure.

2.3.1 Managing the Cloud to IoT Continuum

The Cloud to IoT continuum described in Section 2.1 brings challenges at many different levels. Looking from a broader perspective, one of the first challenging issues is the modeling of the managing and orchestration elements that need to be able to perform the deployment of the services and applications [Chen et al., 2015; Satyanarayanan et al., 2009] in the underlying communication infrastructure and handle tasks inside the environment, while at the same time guarantying good performance and service resilience for end-users. This subsection provides a look at some of the efforts already made in designing general solutions to manage the Cloud to IoT continuum and its components.

A reference architecture for the IoT in the context of the IoT-A European project [Internet of Things Architecture, 2010] is described by Bassi et al., [Bassi et al., 2013]. The idea behind this research is to provide a general IoT architecture that could be instantiated in more concrete use-cases using a domain model that allows representing the key concepts involved in the IoT. Pohls et al., [Pöhls et al., 2014] took some ideas of the IoT-A architecture to develop a framework in the context of the RERUM FP7 European Union project [RERUM, 2013] allowing that IoT applications for Smart Cities to add security and privacy mechanisms in early design phases.

Datta et al., [Datta et al., 2014] propose an IoT gateway centric architecture to support Machine-to-Machine (M2M) services, allowing real-time interaction between clients and IoT devices via a wireless gateway. Atzori et al. [Atzori et al., 2012] combine social networks and IoT, defining the Social Internet of Things (SIoT). In their work, they describe an architecture that enables the integration of things in a social network and analyze the characteristics of the proposed network structure using simulations. Hao et al., [Hao et al., 2014] propose an architecture for the future Internet, called *DataClouds*, based on Information Centric Networks (ICNs) to improve the accommodation of services. The architecture takes into consideration the characteristics of data-centric services under the IoT, using a logical and a physical layer that allows to group users in order to enhance how data is shared and disseminated.

Jin et al., [Jin et al., 2012] present four different architectures based on the IoT that enable various Smart City applications, including their QoS requirements. The proposed network architectures are (1) Autonomous, that supports networks not connected to the Internet; (2) Ubiquitous, where Smart Object networks are a part of the Internet; (3) Application-Layer Overlay, which uses NFV to reduce the stress and congestion among nodes; and (4) Service-Oriented, where special gateways deal with the intrinsic heterogeneity of the IoT environment.

Montori et al., [Montori et al., 2018] introduce an architecture focused on data that handles heterogeneous data sources from the IoT combined with data from crowdsensing campaigns. The unified data is used for IoT service composition enabling monitoring tasks in a Smart City environment by exploiting end-users devices. The proposed architecture is tested in a controlled environment via a set of IoT services connected among them in order to collect environmental data and display it to end-users via a dashboard. Qiu et al., [Qiu et al., 2020] present an architecture for the Industrial IoT including proposals for routing, task scheduling, and data storage and analytics. The architecture is comprised of the Cloud layer and the Edge layer. The authors also include a discussion on application scenarios including Smart Grids, Manufacturing, and Smart Logistics.

The service provider Orchestrator or Manager must have a global view of all the resources and services, from the edge of the infrastructure (i.e., IoT and Fog) to the computation and storage place on top in the Cloud. This global view will allow implementing smart mechanisms to optimize communication tasks, which are described in the following subsection. A comparative analysis of this set of works led to the identification of open issues that have to be addressed in order to improve resilience in the Cloud to IoT continuum. The discussion is presented in Section 2.4.

2.3.2 Connecting the Cloud to IoT Continuum

The evolution of the Cloud to IoT continuum to support new types of applications and services brings additional demands to the network infrastructure because of the presence of challenges such as the heterogeneity of the devices interconnected, the increasing number of end-users accessing the services, and the massive amount of data exchanged. In addition, the interconnection must consider the characteristics of entities that use the network, new services and applications provided over the Internet, and communication platforms such as wireless technologies and Cloud to IoT systems. A discussion about how to enhance communication among devices in Cloud-Fog-Mist-IoT environments is presented below.

The first group of works is related to the IoT infrastructure and proposes improvements directly aimed at the physical infrastructure that allows the communication between smart objects. Particularly, these works present mechanisms for topology control to grant fault tolerance by smart device placement or by taking advantage of the communication infrastructure providing alternative and simultaneous routing paths.

Han et al., [Han et al., 2010] present two algorithms for relay node placement to provide fault tolerance (full and partial) considering the heterogeneity of the wireless sensor network, particularly the different transmission radio footprint. Al-Turjman et al., [Al-Turjman et al., 2011] introduce a grid-based deployment for relay nodes to maximize the disjointed-sectors connectivity taking into consideration cost constraints.

Le et al., [Le et al., 2014] propose three multipath solutions based on Routing Protocol for Lossy networks (RPL). The first one consists in an energy-based load balancing strategy; the second one focuses on a fast local repair approach, and the last one is a combination of the previous two schemes. Pavković et al., [Pavković et al., 2011] present an adaptation of the IEEE 802.15.4 cluster-tree to enable the association of different parent nodes taking advantages of super-frames at the MAC layer. Using this modification, authors develop an opportunistic forwarding scheme that allows RPL to forward packets over multiple paths.

A set of mechanisms focused on a global view of the network infrastructure, from the IoT to the Cloud, Fog, and Mist environments, to allow smart decisions are proposed in the next group of works. The SDN approach [Kirkpatrick, 2013] is used in these researches providing a finer granularity of the network infrastructure and making possible to improve the resilience of the network (e.g., applying better forwarding and routing decisions).

Francisco et al., [Ros and Ruiz, 2014] propose a placement solution for SDN controllers using a *k-terminal-reliability* metric to increase the probability of having at least one operational path in the network. Beheshti and Zang [Beheshti and Zhang, 2012] introduce a set of algorithms to improve the resilience of the connection between control and data planes in SDN, based on resilience-aware controller placement and traffic control routing in the network.

Stephens et al., [Stephens et al., 2013] present an SDN resilience architecture to build networks with forwarding backups paths that, assuming large forwarding tables at data plane level, is resilient against *t* link failures. Reitblatt et al., [Reitblatt et al., 2013] propose a language for coding fault-tolerant network programs based on regular expressions. It allows specifying the paths of packets in the network along with the degree of fault tolerance required.

ElDefrawy and Kaczmarek [ElDefrawy and Kaczmarek, 2016] introduce a prototype for an SDN controller for *Byzantine* faults using a replication strategy. The performance of their proposal is assessed against other standard SDN controllers, and the results indicate that their proposal is not applicable for large scenarios.

Rehman et al., [Rehman et al., 2019] present a survey about SDN fault tolerance research efforts. The idea is identifying fault tolerance requirements for SDN environments and how it can be addressed. Despite a complete discussion regarding how to tackle faults at data and control planes in SDN is presented in this research, no particular solution or mechanism is proposed.

Malik et al., [Malik et al., 2020] propose an approach for fault management at the data plane aimed at eliminating the convergence time required to allocate new network paths to forward the traffic once a failure is detected. The availability of the services should be increased by reducing the service disruption time using a time windows approach to reconfigure the network before anticipated failures take place in the communication infrastructure.

These works handle fault tolerance to increase resilience by managing the communication infrastructure. Virtualization technologies have paved the way to build a reliable Cloud to IoT infrastructure not only from the communication point of view but also taking into consideration services and applications. An analysis of the limitations of this set of works is provided in Section 2.4. Some approaches based on smart mechanisms for embedding virtual services/applications in the substrate communication infrastructure are presented in the next subsection.

2.3.3 Embedding in the Cloud to IoT Continuum

Virtualization techniques, in the context of the Cloud to IoT, have been used to provide services and functions to reach end-user requirements, for example, when the NFV approach is used to map a virtual network in a substrate communication infrastructure (see Subsection 2.1.1). When this approach is implemented, a key aspect to consider lays in the embedding mechanisms adopted to perform the mapping of the VFs that belong to a SC in the best devices of the physical infrastructure to enhance the QoS of end-users, as well as, fulfill the metrics of the service providers.

There has been some work previously done in VFs embedding, and more specifically, concerning how to embed requested VFs in the substrate communication topology with the objective of increasing the survivability of the SCs. This subsection presents some works on embedding before introducing works focused on the area of resilient embedding.

Chowdary et al. [Chowdhury et al., 2009] propose two heuristics (called D-ViNE and R-ViNE). Their aim was to increase the acceptance ratio and revenue while decreasing the cost for the substrate network. The heuristics map Virtual Networks (VNs) requests into the substrate network based on previous requests.

Beck and Botero [Beck and Botero, 2017] tackle the NFV resource allocation problem by dividing it into two phases: service chain composition and service chain embedding. The embedding algorithm, called CoordVNF, tries to assign a Virtual Network Function (VNF) to a node in the substrate network and follows the flow of the service chain to assign the following VNF in a neighboring node. In case of finding a problem (e.g., not enough resources in neighboring nodes), the algorithm backtracks until finding a solution.

Three evolutionary algorithms for service embedding in the Fog are proposed by Guerrero et al. [Guerrero et al., 2019]: single-objective genetic algorithm with Weighted Sum Genetic Algorithm (WSGA); Non-dominated Sorting Genetic Algorithm-II (NSGA-II); and, a Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D). Three objectives were drawn: minimizing latency; minimizing free resources; and optimizing service spread (even distribution of services). The best results were obtained with NSGA-II overall, but MOEA/D showed lower latency. WSGA showed better convergence times in comparison with the other algorithms.

Bays et al. [Bays et al., 2016] uses knowledge of the substrate network to create a virtual infrastructure abstraction allowing the representation of VN requirements, while also proposing an embedding model ensuring physical feasibility. They use SDN and particularly OpenFlow to gather information about the substrate network. Embedding requests are processed by a privacy-aware compiler.

Mehraghdam et al. [Mehraghdam et al., 2014] present a model to specify network function chaining requests before introducing an embedding solution. The representation model proposed takes into consideration Network Functions (NFs) chaining with ordered and unordered NFs, and provides the possibility of splitting the communication flows through a set of NFs; however, the resilient component of the NFs, as well as, the tiered approach that drives the Cloud to IoT continuum are not considered in the model.

So far, the works analyzed focus on VFs embedding, without considering the survivability of the SCs. The following works are aimed at increasing the resilience of the SCs.

Khan et al. [Alam Khan et al., 2016] tackle the survivable VF embedding problem. They propose SiMPLE, a multi-path link embedding mechanism to maximize survivability while minimizing resource wastage with replicas. SiMPLE works in two stages: a proactive one to embed the VFs as they arrive, and a reactive stage in case of link failure. This solution relies on the existence of disjoint paths in the substrate network in order to guarantee its success.

Rahman and Boutaba [Rahman and Boutaba, 2013] also analyze the survivable virtual function embedding problem. They propose a hybrid heuristic as well as a baseline heuristic. The hybrid heuristic uses a re-routing strategy that previously reserves a backup quota on each physical link. The proposal is based on single substrate link failures and does not deal with node failures.

Proactive and reactive approaches are also studied by Souza et al. [Souza et al., 2017], who model them as the well-known multidimensional knapsack problem. Their proactive recovery strategy consists of pre-allocating backup resources for each primary resource that can become available as soon as the failure occurs; while the reactive recovery strategy has a pool of backup resources that is shared among primary resources to minimize resource utilization.

Aidi et al. [Aidi et al., 2018] propose the use of replicas to increase the survivability of SFCs. They also present two heuristics for more complex scenarios. The first heuristic analyzes each node to determine the maximum amount of VFs that it can backup. The second heuristic allows the host to backup VFs that are as spread as possible in different physical nodes.

Lera et al. [Lera et al., 2019a] introduce an embedding policy aimed at increasing the service availability and the QoS. They use communities to divide the network infrastructure and embed the SC inside said communities. The graph partition technique applied to create the communities among the Fog nodes is based on the work of Newman and Girvan [Newman and Girvan, 2004].

After analyzing these efforts, it is possible to establish research paths to enhance the resilience in the Cloud to IoT continuum. Some open research issues identified are presented in the following section.

2.4 Discussion

Despite the abundance of available studies in the area of resilience for the Cloud to IoT continuum, there are still open challenges regarding the availability of virtual services and functions, as well as the management and orchestration of the heavily virtualized infrastructure used in this context.

From the managing perspective, many works are solely based on proposals without including some sort of validation [Bassi et al., 2013; Atzori et al., 2012; Hao et al., 2014; Jin et al., 2012; Qiu et al., 2020], with some works even focusing only on social-based approaches [Atzori et al., 2012; Hao et al., 2014; Montori et al., 2018]. On the other hand, some implementations were carried out [Montori et al., 2018] and some were tested in real Smart City scenarios [Pöhls et al., 2014].

The Cloud to IoT (not considering the Fog or Mist) was taken into account in most proposals [Bassi et al., 2013; Datta et al., 2014; Atzori et al., 2012; Hao et al., 2014; Montori et al., 2018] with some works only based on IoT [Datta et al., 2014]. Most recent works include the notion of Fog computing [Qiu et al., 2020], since this is a more recent paradigm. Resilience was not included as a main characteristic in some of the studied architectures [Datta et al., 2014; Atzori et al., 2012; Hao et al., 2014; Montori et al., 2018].

Taking into consideration the characteristics and requirements of the Cloud to IoT infrastructure that support Smart Cities [Jiong et al., 2014], and the revision of the state of the art; a proposal of a novel Cloud to IoT infrastructure architecture for Smart Cities, emphasizing the survivability of the services is presented in this research. The lack of support from the communication infrastructure for the *IoT Services* is addressed in this proposal by offering heterogeneity, flexibility, and resilience support. The proposed architecture is detailed in Chapter 3 considering the SusCity project specifications.

The SusCity project [SusCity-Project, 2015] is taken as a case study to gather the requirements of the Smart City services using the Cloud to IoT continuum as interconnection backbone. The SusCity project involves the collaboration of several universities (University of Coimbra, University of Minho, Instituto Superior Técnico de Lisboa, Massachusetts Institute of Technology) and companies (IBM, Energias de Portugal - EDP), with the objective of facilitating the transition of the city of Lisbon to a Smart City. The idea of this project is to collect data from different sources in order to build models that can produce predictions about possible smart solutions like mobility, buildings, and energy grids, to help government and citizens make appropriate decisions.

For the communication perspective, the research area shows maturity regarding the protocol and algorithms used to enable the interconnection of devices along the tiers in the Cloud-Fog-Mist-IoT, favoring the decoupling of the data and control planes. SDN was used in many of the works [Ros and Ruiz, 2014; Beheshti and Zhang, 2012; Stephens et al., 2013; Reitblatt et al., 2013; ElDefrawy and Kaczmarek, 2016; Rehman et al., 2019; Malik et al., 2020]; however these proposals were generic enough to not focus on the Cloud or IoT, unlike other works that were aimed at connectivity in the IoT [Han et al., 2010; Al-Turjman et al., 2011; Le et al., 2014; Pavković et al., 2011]. The solutions were based on mathematical models [Han et al., 2010; Al-Turjman et al., 2011; Ros and Ruiz, 2014; Stephens et al., 2013; Malik et al., 2020], heuristics [Han et al., 2010; Al-Turjman et al., 2011; Ros and Ruiz, 2014; Beheshti and Zhang, 2012; Stephens et al., 2013; Malik et al., 2020], protocols [Pavković et al., 2011; Le et al., 2014], and prototypes for SDN controllers [ElDefrawy and Kaczmarek, 2016; Reitblatt et al., 2013].

As expected, the majority of works based on connectivity were focused on link failures [Le et al., 2014; Pavković et al., 2011; Ros and Ruiz, 2014; Beheshti and Zhang, 2012; Stephens et al., 2013; Reitblatt et al., 2013; ElDefrawy and Kaczmarek, 2016; Rehman et al., 2019; Malik et al., 2020], with some exceptions based on node and link failures [Han et al., 2010; Al-Turjman et al., 2011]. Simulation was the evaluation tool used in most of the works [Han et al., 2010; Al-Turjman et al., 2011; Le et al., 2014; Pavković et al., 2011; Ros and Ruiz, 2014; Beheshti and Zhang, 2012; Malik et al., 2020] although some works used a testbed [Le et al., 2014; Pavković et al., 2011; Stephens et al., 2013; Reitblatt et al., 2013; Malik et al., 2020] and even emulation [ElDefrawy and Kaczmarek, 2016].

Virtualization is the fundamental property of the new communication infrastructure, making necessary to deal with the mapping of virtual services, objects, and functions in the proper substrate physical components. Thus, in recent years a considerable effort has been done concerning how to deal with the embedding in the context of the Cloud to IoT continuum. For the analysis performed in the literature regarding embedding, some observations arise. First, simulation seems to be the main method for evaluation [Chowdhury et al., 2009; Guerrero et al., 2019; Alam Khan et al., 2016; Souza et al., 2017; Aidi et al., 2018; Lera et al., 2019a], which is due to the complex scenario comprised by the Cloud to IoT continuum. Mathematical solutions are also broadly used, following ILP models [Chowdhury et al., 2009; Bays et al., 2016; Alam Khan et al., 2016; Aidi et al., 2018; Lera et al., 2019a]; even though these models perform well in offline studies, they suffer from time constraints when applied to online scenarios. Heuristics were also explored [Chowdhury et al., 2009; Beck and Botero, 2017; Guerrero et al., 2019; Alam Khan et al., 2016; Rahman and Boutaba, 2013; Aidi et al., 2018; Lera et al., 2019a].

Although the embedding problem has been previously addressed, SCs resiliency has been mostly overlooked [Chowdhury et al., 2009; Beck and Botero, 2017; Guerrero et al., 2019; Bays et al., 2016; Mehraghdam et al., 2014]. The works considering resiliency can be grouped into link failures [Alam Khan et al., 2016; Rahman and Boutaba, 2013], and node failures via resource allocation [Souza et al., 2017] or replication [Aidi et al., 2018; Lera et al., 2019a]. Thus, there is still room for improvement of the resilience of computing and networking elements in the Cloud to IoT continuum.

The embedding of computing and communication virtual services/functions is a key aspect in the Cloud to IoT continuum to guarantee resilience to applications, particularly for Smart Cities scenarios. This research presents a framework for embedding SCs and their VFs in a Cloud to IoT infrastructure using a method to formalize customized Service Chains requests, depicted in Chapter 4; and a set of mechanisms that take into consideration information about the substrate components(i.e., the availability factor of the nodes, and the tiered infrastructure) for the embedding process, introduced in Chapter 5. The mechanisms are validated via simulations in Chapter 6, since this is a common evaluation tool used for this type of complex scenarios.

2.5 Summary

In a Cloud to IoT environment, smart devices communicate with each other and with the end-users through the Internet to gather, process, and analyze data, without much (or any) human intervention. This inevitably will enable the rise of new generation services and applications where unique and customized information will be processed for end-users on demand. This brings along different challenges that have to be addressed to guarantee their proper function while providing acceptable performance for end-users.

Thus, it arises the need of automating application workflows in the sense of providing dynamic policy-based life cycle management of the communication infrastructure and services. This includes provisioning, management, and monitoring on a large number of nodes and devices with a broad range of capabilities that include computing (computer resources), routing (network), and distributed databases (storage). This calls for the design and development of an efficient management/orchestration architecture to supervise and administrate heterogeneous, and distributed systems spread across a wide geographical area while delivering services to end-users maintaining their availability even in the face of failures, taking into consideration that Smart City services handle critical functions that can not be disrupted.

Virtualization has evolved as a technique that allows improving the usage of physical resources. However, the decision on how to use the physical resources, for instance, to select the physical nodes on which to embed the virtual services and/or network functions, becomes crucial in such a complex landscape. Some work has been dedicated to address resource provisioning and management of SFCs assuming the complete availability of the physical infrastructure, which is not realistic as failures are common in the Cloud to IoT continuum. Consequently, it is necessary to have mechanisms that allow dealing with failures in the Cloud to IoT infrastructure. A possible approach is to study the problem of VFs placement using smart replication as a resilience mechanism since replicas can be activated or instantiated in case of failures.

This chapter presented a revision of the research context, including a description of the Cloud to IoT continuum and the virtualization techniques applied to facilitate the management and deployment of services in such environments, to later describe resilience in communication infrastructures also identifying the resilience challenges that the Cloud to IoT bring. Moreover, a revision on existing efforts to improve the resilience of the Cloud to IoT continuum is also introduced, including visions on managing, interconnection, and embedding in the Cloud to IoT. This revision allowed to identify novel research paths that enable the further improvement of the resilience in the Cloud to IoT continuum using Smart City applications as a case study.

The research paths identified in this chapter include the design of an architecture to overlook the proper functioning of smart services in the Cloud to IoT continuum, focused on improving the resilience of the services, and the proposal of smart service composition and embedding solutions for the Cloud to IoT continuum. Replication of the VFs was chosen as the resilience approach to use during the embedding process

Chapter 3

Improving the Resilience of the Cloud to Internet of Things Continuum

Contents

To deal with the complexity and variety of the Cloud to Internet of Things continuum, an architecture that improves the resilience of this environment following an end-to-end approach (i.e., from the IoT Infrastructure, traversing the network and Cloud/Fog domains to the IoT services) is discussed in this chapter.

The architecture proposed takes into consideration the design and implementation aspects of each layer, proposing protocols and technologies that support key resilience features. Additionally, an ontology for the IoT infrastructure in Smart City scenarios is designed to deal with the plethora of heterogeneous devices and data sources in a unified and standard way.

3.1 Understanding Smart City Scenarios

Many cities worldwide are trying to evolve to the paradigm of Smart City. In this paradigm shift process, ICT have the important role to provide the infrastructure to deploy a surplus of devices and services. This infrastructure can be quite large and heterogeneous, composed of different types of devices and Smart Objects (e.g., sensors, actuators), that need to use diverse communication networks (e.g., cellular, satellite, the Internet) to provide services to final users. Frequently, these services are running totally or partially in Cloud and Fog environments; where a vast number of digital equipment such as servers, network devices, and storage components interact to fulfill the requirements of end-users.

According to IBM [IBM Industry Solutions, 2013], the services that need to be deployed in a Smart City could be grouped in the following key domains: *Government and Agency Administration*, *Public Safety*, *Urban Planning*, *Social and Health*, *Education*, *Transportation*, *Energy and Water*, and *Environmental.* These services have the common goal of improving the quality of life of the citizens. Figure 3.1 shows a typical scenario of a Smart City highlighting these domains.

In a Smart City, data is collected from sensors deployed in strategic positions. In order to process the data and make smart decisions, given the resource constraints of the Smart Objects, the gathered data has to be sent to the Cloud or Fog, where particular services perform an analysis of the urban activities of the city. Furthermore, it is important to highlight the role of the actuators in this context, since these devices carry out actions defined by the *IoT Services* in the physical environment (e.g., controlling traffic lights).

An example of the tasks previously described is depicted in Figure 3.1. Environment monitoring and noise sensors are deployed in the Smart City to collect data that is sent to *Urban Analytics* services hosted in the Cloud, using a communication network infrastructure. The combination of Smart Objects and *IoT Services* is becoming a reality in several cities across the world [TUWIEN. Vienna University of Technology , 2015]. To guarantee the reliability of the *IoT*

Services in this new paradigm, novel solutions must be designed. An architecture to ensure resilience in the deployment of Smart City services is presented in the following section. The proposed architecture was designed and validated in the scope of the Suscity project [SusCity-Project, 2015] taking advantage of the requirements and use cases embedded in it.

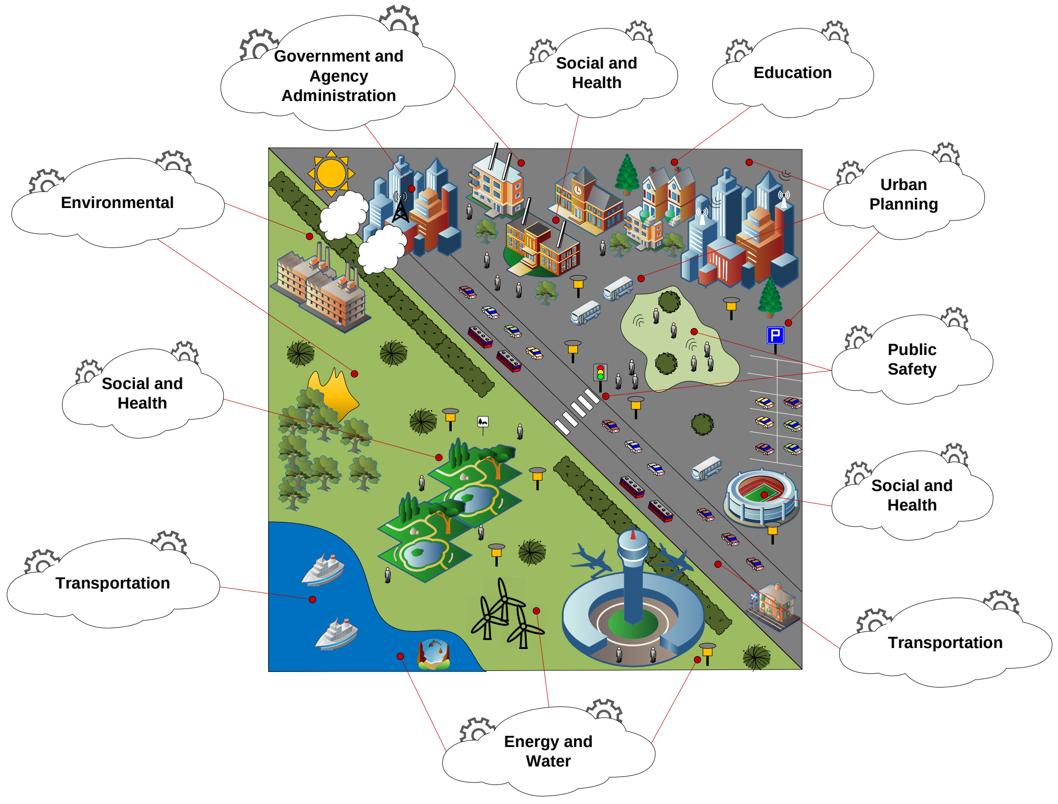


Figure 3.1: A Smart City Example with its Services Domains.

3.2 A Resilience Architecture for the Cloud to IoT Continuum

To satisfy the requirements of the scenario described in Section 3.1 while guaranteeing a high resilience level to the services and infrastructure, the Cloud to IoT continuum architecture depicted in Figure 3.2 is proposed. This architecture has three layers (*IoT Infrastructure*, *IoT Middleware*, and *IoT Services*) that tackle specific functions to make possible the support of the Smart City paradigm using the IoT, Cloud, and Fog nodes. A key feature of the architecture is the possibility to have more than one instance by layer; this is represented by the overlap of boxes (i.e., slices). Each group of instances from the architecture, represented by the slices, compose an *IoT Domain*; denoting the specific components and modules that belong to a particular entity. Additionally, in order to reach ubiquity and more flexibility of the components that shape the architecture; the *IoT Middleware* and the *IoT Services* layers reside in the Cloud/Fog environment. Furthermore, the architecture allows the deployment and virtualization of crucial components (i.e., *IoT Gateways* and *IoT Services*) at the edge of the ICT infrastructure, in Fog Nodes, achieving a latency reduction which is important particularly for real-time and critical applications.

In the remaining of this section, the components and modules of each layer and their interactions will be discussed in detail, identifying possible protocols/technologies that could be applied to enhance the resilience of each component.

3.2.1 IoT Infrastructure

The lower layer of the architecture deals with the physical *Devices* deployed in the city. These devices are Smart Objects that enable the gathering of data and react to specific situations. A group of devices is referred to as an *IoT Island*. Typically, the data collected by an *IoT Island* has to be sent to *IoT Services* to be processed and to perform a complete analysis. Given that the devices usually are limited from the performance point of view, it is common to deploy special devices (i.e., *IoT Gateways*) at the edge of the network topology to connect the *IoT Island* with the Internet. These *IoT Gateways* together with virtual devices and services hosted in Fog nodes can perform data compression and aggregation to minimize network load and add intelligence to the IoT environment. A possible approach to improve the performance of the IoT infrastructure is to virtualize the *IoT Gateway* functions allowing to instantiate and deploy them on-demand; this could be achieved, for example, using Kura [IoT-Eclipse, 2015a].

An *IoT Gateway* provides access to multiple *IoT Islands* during the data transmission from the *IoT Infrastructure* to the upper layer. The communication between objects in this layer has to be ruled by an efficient routing protocol that achieves the special requirements of the IoT. RPL [Winter et al., 2012], defined by the Internet Engineering Task Force (IETF) IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) working group [IETF, 2015], could be used to deal with the dynamism of the IoT network in an automatic way, by using different objective functions to react to the changes. Moreover, RPL supports two basic mechanisms to recover from failures (Global and Local repair) that could be extended in order to improve the resilience features of this protocol. Also, RPL allows the use of multipath techniques, which further enhance the resilience of the solution [Le et al., 2014; Pavković et al., 2011]. In addition, 6LoWPAN [Mulligan, 2007] could pave the way to use the well-known Internet technologies and protocols in the IoT to enable the use of Wireless Sensor Networks (WSN) approaches (e.g., relay node placement to provide fault tolerance [Han et al., 2010; Al-Turjman et al., 2011]). Regarding ultra-low latency applications and critical services, it is possible to deploy them directly at the Fog level, near to the *IoT Infrastructure* [Satyanarayanan et al., 2009].

After having defined the layer that is in charge to map the world of things into the world of computationally processable information, in the next section, we discuss the layer responsible for guaranteeing that the data gathered in the *IoT Infrastructure* reach its destination, the *IoT Middleware*.

3.2.2 IoT Middleware

Due to the vast number of technologies typically in place within an IoT scenario, it becomes necessary to have a layer to offer a seamless integration of devices and data that build the IoT, the *IoT Middleware*. This layer encompasses common functionalities and abstraction mechanisms that wrap the IoT infrastructure details to developers and users in order to achieve an easier interaction between these actors [Bandyopadhyay et al., 2011].

Research projects such as LinkSmart [LinkSmart, 2015] and OpenIoT [OPENIoT, 2015] provided important contributions to this integration layer; however, regarding the resilience mechanisms needed in the infrastructure, there is still room for significant improvements. With this in mind, we propose an *IoT Middleware* layer focused on the enhancement of the resilience of the IoT. A discussion of the modules of this layer is presented next.

Heterogeneity Manager

Given the devices deployed in the *IoT Infrastructure* have a heterogeneous nature, it is necessary to have a standard language to reach an agnostic communication between the *IoT Infrastructure* and the upper layers. The *Heterogeneity Manager* works as an interpreter between the components of the IoT located in the Cloud and the *IoT Islands*. The translation of the communication protocols used in the *IoT Island* to a common language is the main role of this component. For the tasks that have to be performed in this layer, the IoT Eclipse project [IoT-Eclipse, 2015b] looks useful; since it offers open source implementations of well-known communication protocols for the IoT. In this context, the framework proposed by Sköldström et al. [Sköldström et al., 2014] addresses the unification of the network and Cloud resources, which is a key aspect for the IoT.

Communication Manager

The *Communication Manager* rules how the data is exchanged between services, applications, and Smart Objects. The main goal of this component is to offer an efficient and flexible way to control the route of the data, the structure of the communication infrastructure, the mobility of the entities, and the specific requirements in the context of the IoT.

Four modules integrate the Communication Manager. The *Path Control* deals with the dynamism of the IoT environment and support mechanisms to change in real-time the path for specific flows. The *Topology Control* provides a global view of the infrastructure by using the knowledge of the status of the devices that support the communication process. An additional feature supported by this module is the possibility of having different VN, as disjoint as possible, over the same physical infrastructure, enabling the overlaying of network topologies. The *Mobility Control* is responsible for storing data in temporary buffers at intermediate devices, while the mobile service or object reaches its destination. The *QoS Control* module guarantees that the traffic demands of services and

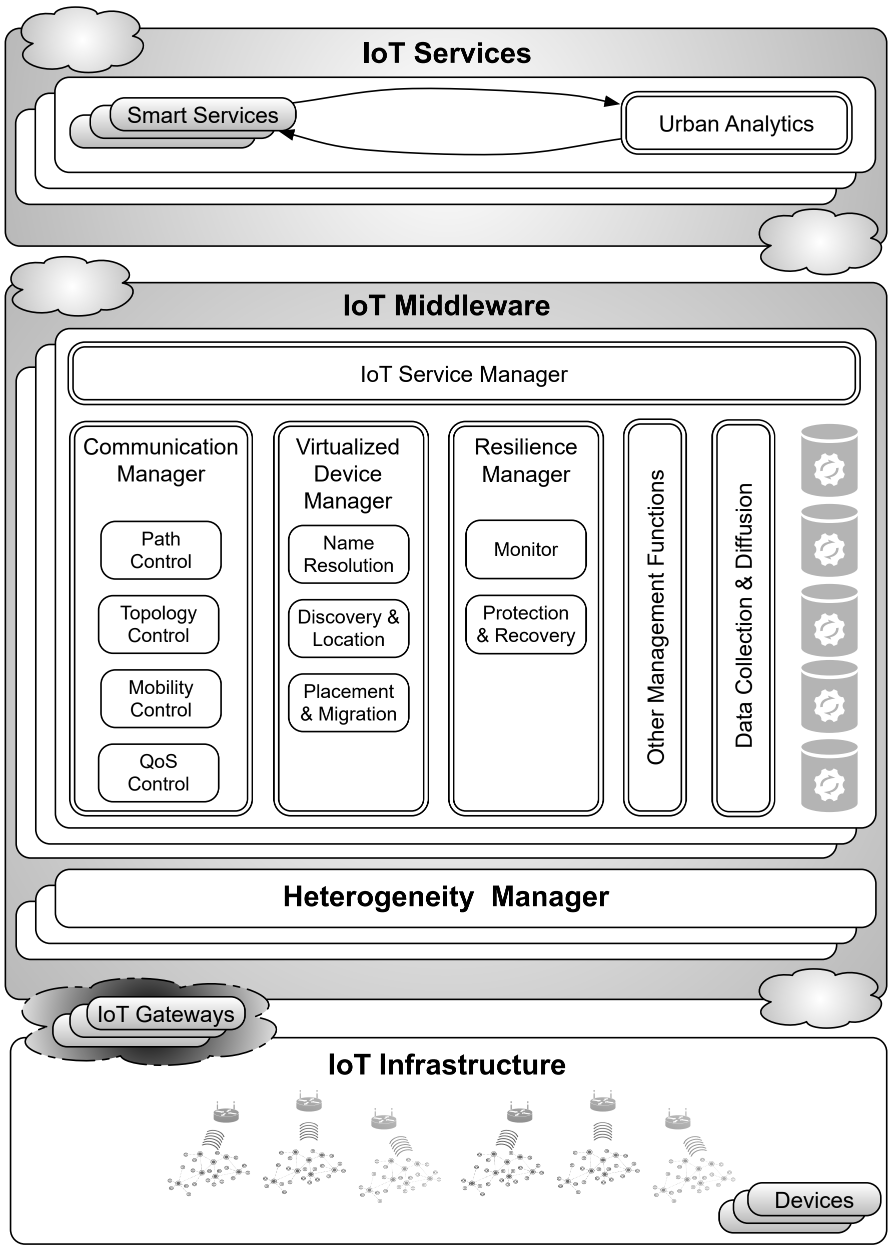


Figure 3.2: A Resilience IoT Architecture for Smart Cities.

applications are met by offering the possibility to manage different traffic configurations and also supports the proper mechanisms to achieve the service-level agreements.

A possibility to carry out the tasks described before is using a combination of the SDN [Kirkpatrick, 2013] and VN [Chowdhury and Boutaba, 2010] approaches, obtaining a Virtual Software Defined Network (VSDN) [Gomes et al., 2016]. By decoupling the control and forwarding functions inside the *Communication Manager* component, it is feasible to incorporate desirable features to the infrastructure that support the services and applications of a Smart City, for example, dynamism, mobility, and resilience. From the infrastructure resilience point of view, the possibility to integrate disjoint VNs with a multipath approach using VSDN is a promising strategy.

Virtualized Device Manager

The administration of the devices is supported by the *Virtualized Device Manager*. This component offers the mechanisms to identify, discover and locate services and devices deployed in the *IoT Infrastructure*, allowing the elements inside of the IoT to move across domains.

The three modules in this component and their functions are the following. The *Name Resolution* module has to take into consideration two basic tasks; the first is name mapping, where a local name could be translated to a flat or agnostic name; then in a second phase, it is necessary to apply a name resolution strategy. The *Discovery & Location* module tracks the devices and objects deployed in the *IoT Infrastructure*. The *Placement & Migration* module allows to instantiate services or devices to improve or maintain the requirements of the IoT applications while minimizing the resource consumption. Using a migration strategy makes it possible to keep alive important services even in a fault scenario.

We identified some useful approaches to perform the tasks of the modules inside the *Virtualized Device Manager*. With NFV [Matias et al., 2015], it will be possible to virtualize network services and devices on-demand. Moreover, Jemaa et al. [Ben Jemaa et al., 2016] combined the approach of NFV with the Fog to make flexible the deployment of services. These technologies allow to decrease the resource consumption in the backbone of the IoT; furthermore, it is feasible to handle a higher number of devices, an essential requirement for the IoT. In addition to NFV, the adoption of smart and cognitive techniques to improve the service placement and migration, as well as machine learning approaches, could bring significant benefits to the IoT adding self-management features.

Resilience Manager

The main goal of the *Resilience Manager* component is to offer robustness to the IoT infrastructure by a permanent supervision of the activities (*Monitor* module) performed by the fundamental modules of the IoT architecture. Additionally to the monitoring actions, the module of *Protection & Recovery* works in conjunction with the *Path Control*, *Topology Control*, and *Placement & Migration* modules to guarantee that the proper actions will be applied in case of faults.

Consequently, with the discussion of the architecture so far, it is important to make clear that the *Resilience Manager* is not solely in charge of increasing the resilience of the overall system, but instead works as an orchestrator of the resilience-related tasks that are disseminated throughout the architecture. Hence, the architecture follows a distributed approach towards the resilience task application, minimizing the bottleneck result of grouping these functions in a single component.

Another key aspect to be considered in this module is that given that the infrastructure is broad and heterogeneous, recovery tasks must be automatized.

Having a knowledge base of the infrastructure and its status and using this knowledge with some cognitive strategies is a feasible solution. Both online/offline mechanisms to recover the IoT infrastructure after failures have to be implemented.

Other Management Functions

The component *Other Management Functions* represents additional management tasks that could be required by the *IoT Middleware*. An important component that could be required by an entity would be a *Security Manager*. This component should deal with the threats present in the IoT in order to mitigate vulnerabilities.

Data Collection & Diffusion and Knowledgment Database

The *Data Collection & Diffusion* component is in charge of gathering the data from the sensors in the *IoT Infrastructure* and disseminate this data to the requiring *IoT Services*. On the other hand, the *Knowledgment Database* stores the information about the devices and the physical topology of the network. This module needs to deal with the heterogeneity of the devices in the lower layer. Thus, it is important to use a standard to manage and describe these devices uniformly. A tentative approach for this last component could be using semantic web techniques, such as ontologies [Wei et al., 2012].

IoT Service Manager

The *IoT Services* layer requires a standardized way to access the functionalities provided by the *IoT Middleware*. This is the task of the *IoT Service Manager*, which implements a common Application Programming Interface to enable a transparent communication of the services with the south-bound region of the architecture.

After discussing the tasks performed by the *IoT Middleware* and pointing out its importance; in the next section, we explain the components of the *IoT Services* layer.

3.2.3 IoT Services

The management of applications and services that support the Smart City, as well as the analysis of the data collected from the city are performed in the *IoT Services*. In this layer, the *Urban Analytics* component takes as input the data gathered by the *Data Collection & Diffusion* to perform characterization and analysis in order to generate information that feeds the *Smart Services*. In the *IoT Services*, Big Data techniques [Akusok et al., 2015] play a vital role to process the vast amount of data collected by sensors; that is consumed by the *Smart Services* using a standard protocol like the Constrained Application Protocol (CoAP) or Message Queuing Telemetry Transport (MQTT) [Palattella et al., 2013]. To cope with the resilience requirements for this layer, the use of caching and replication techniques at the service level could be applied. Furthermore, instances of *IoT Services* could be run on Fog nodes allowing access to end-user in a single-wireless-hop fashion.

So far, in this section, we have explained in detail the layers of the proposed architecture and also the interconnection of the modules and their tasks. Using the previous discussion, we now explain how this architecture overcomes fault scenarios, with the combined efforts of its modules.

3.3 Interaction among the Modules of the Architecture

The different modules described in Section 3.2 tackle individual tasks involved in the global Smart City scenario, with the goal of increasing the overall resilience level of the infrastructure and the services provided. However, the success of the architecture relies on the harmonious interaction among the modules. To illustrate this fact, two examples are provided in this section.

Figure 3.3 depicts the first example. Imagine a scenario where a fault is detected in a component of the substrate network. The *Monitor* is constantly querying the *Topology Control* module to get the status of the VNs. Once a failure is detected, the *Monitor* begins the restoration process by requesting the recovery of the failure to the *Protection & Recovery* module. This module will analyze the fault that occurred and will take the proper actions to overcome it. To achieve this, it can request information from other modules, such as *Placement & Migration*. Assuming that the fault detected in the topology is such that the sensor which the service is trying to reach is no longer available; the *Protection & Recovery* module determines that the best option is to create another instance of the Virtual Sensor (VS) that will replace the one affected by the failure detected. Afterward, a new VN must be configured in which the new VS instance is included. Finally, the *Protection & Recovery* module notifies the *Monitor* that the recovery actions are completed.

Another example is depicted in Figure 3.4. In this case, imagine that instead of a failure, a drop in the QoS is detected, for instance, by an increment in the traffic that would generate congestion and eventually a greater latency for the services. Once again, the *Monitor* will be permanently checking the system status and will get a QoS drop notification from the *QoS* module. The restoration actions begin by a notification sent to the *Protection & Recovery* module, which will determine the appropriate tasks to overcome this situation. In this example, assuming that there is an alternative path (with lower latency) to reach the same destination, the *Protection & Recovery* module indicates to the *Path Control* module that this alternative path should be upgraded as the primary path. The recovery process finishes with a notification to the *Monitor*.

It is important to notice that, although there is a specific *Resilience Manager* in the architecture, all the modules were designed with the idea of augmenting the resilience level of the infrastructure. Thus, there are different tasks that provide more resilience to the final scenario in other modules, such as the backup topologies calculated by the *Topology Control* module.

Furthermore, it is essential to emphasize the importance of using both the Cloud and Fog paradigms. By placing the modules and services on the Cloud, the architecture becomes infused with ubiquity. The positioning of the gateways in

Fog nodes allows the execution of some services and analytics in a single-hop wireless fashion, which provides an inherent resilience to the services since they become reachable even in the case of faults at the Cloud level.

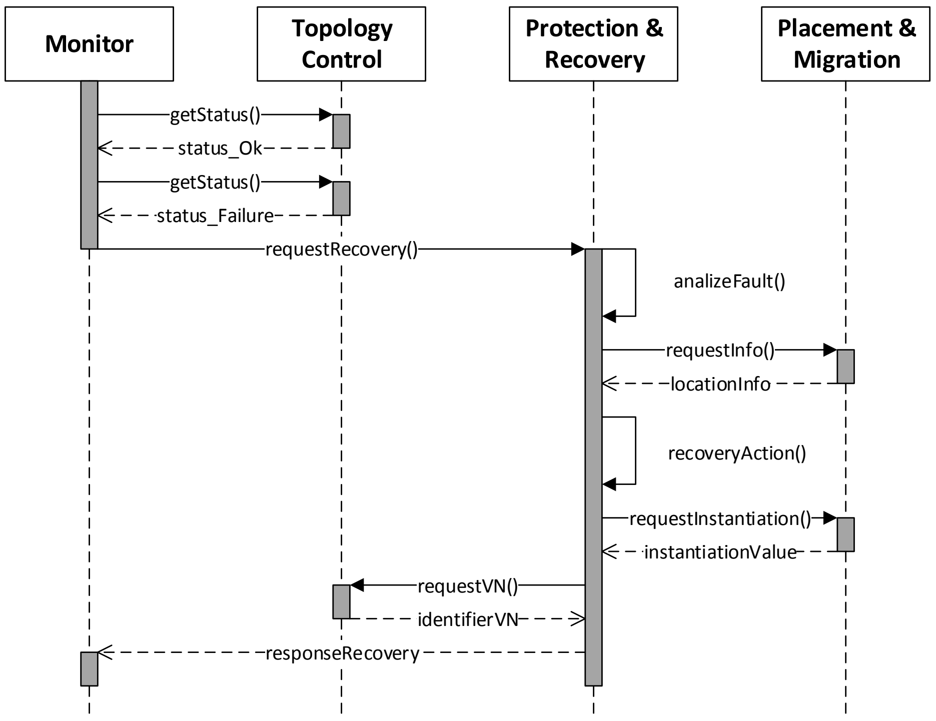


Figure 3.3: Interaction Among the Modules - Fault Detected.

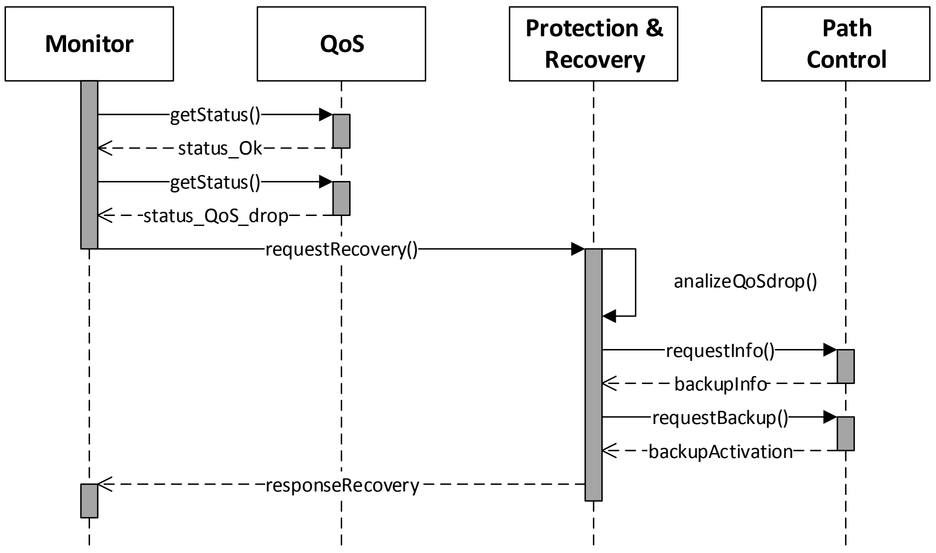


Figure 3.4: Interaction Among the Modules - QoS Drop Detected.

Moreover, by combining the proposed architecture with the proper mechanisms and protocols, an even higher resilience can be achieved. For instance, consider Figure 3.5. On the top of the figure, at the Cloud level, reside the services that are trying to reach the sources of data (i.e., sensors). Then, at the Cloudlet/Fog level, reside the gateways, in charge of the data aggregation tasks. Finally, at the lower level, reside the IoT island, composed of groups of sensors and actuators. In Figure 3.5, straight lines represent the primary paths, while dashed lines represent backup paths.

On a scenario like the one depicted in Figure 3.5, is noticeable that given the redundancy of the communication links, it is possible to create VNs allowing the reachability of an *IoT Island* via more than one disjoint path. This approach increases the availability of the services in case of failures while also allows recovery from a low-performance situation. Furthermore, since a sensor can have more than one parent on the physical topology; it is possible to combine RPL with a multipath approach [Le et al., 2014; Pavković et al., 2011] to enhance the resilience of the smart object using the network infrastructure (e.g., a sensor can be connected to different gateways from different Fog nodes), see Figure 3.5.

Notice that one *IoT Island* could be connected to several gateways located on different Fog nodes which are connected to each other. This provides an extra level of resilience in the case of a failure on a Fog node or in the occurrence of a failure in the connection between a Cloudlet/Fog and the Cloud.

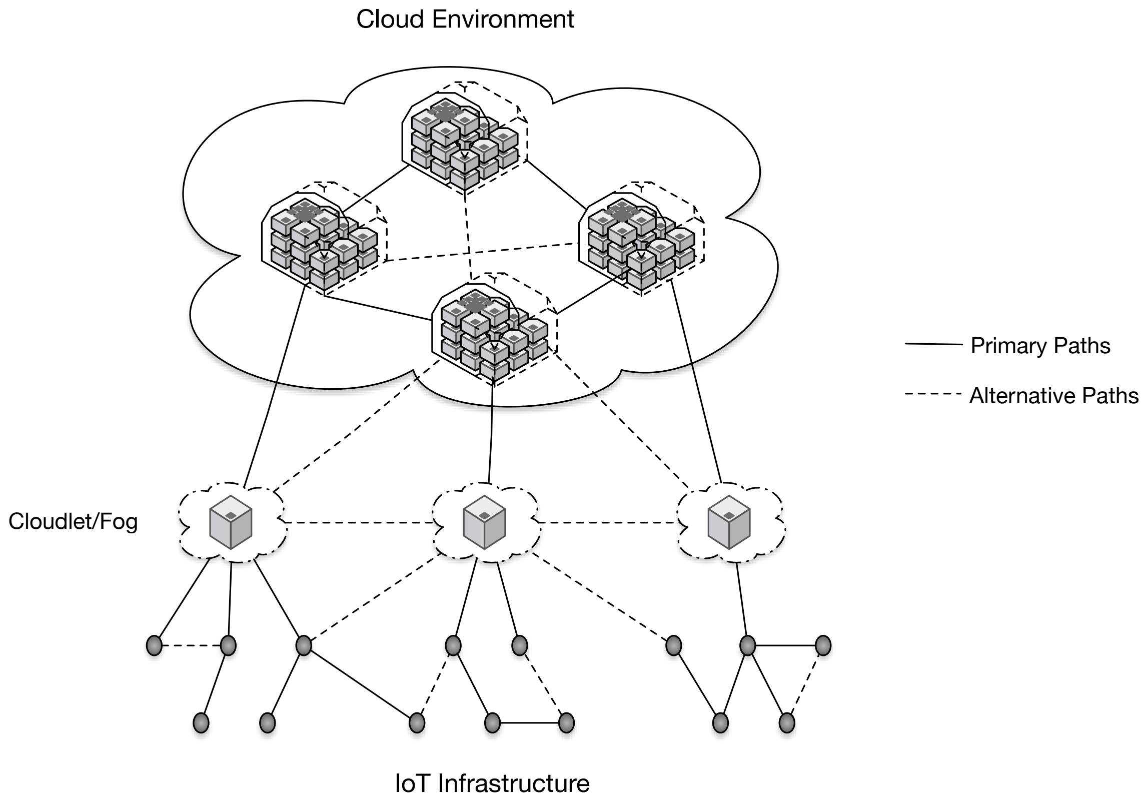


Figure 3.5: IoT Communication Infrastructure Global View.

3.4 An Ontology to Describe the IoT Infrastructure

As an additional contribution of the architecture described above, an ontology for the IoT infrastructure in Smart City scenarios is proposed. The idea behind this ontology is to unify the access to the data, making it possible to collect, in a standard and easy way the information that protocols, services, and user applications need to make smart decisions to optimize the performance of the network, according to the requirements inherent to Smart Cities. The proposed ontology was successfully tested against consistency issues and also populated and queried using SPARQL Protocol and RDF Query Language (SPARQL) [W3C, 2013]. Particularly, two optimization scenarios (i.e., lower latency and high resilience) were utilized in the design of the example queries as a way to illustrate the potential impact of using this type of solution to describe a Smart City environment to optimize the performance of the infrastructure.

The main objective of this proposal is to build a complete and detailed ontology to model all the components involved in the IoT infrastructure. The proposed ontology was tested in the context of the SusCity project [SusCity-Project, 2015], making particularly emphasis on the communication devices with the goal of gathering information that enables the decision-making process of different mechanisms that guarantee the proper function of the infrastructure. A complete version of the ontology discussed in this section is available online [Perez Abreu and Velasquez, 2016].

The IoT infrastructure is represented by an entity (*IoT_Infrastructure*) that includes different elements attached to it. After evaluating the interests and needs of the SusCity project, four additional entities were identified (*Device*, *Link*, *Interface*, and *Metric*). Two additional classes (*Action* and *Location*) were added to simplify the modeling of some required information that is going to be detailed later in this section. The main class hierarchy is shown in Figure 3.6.

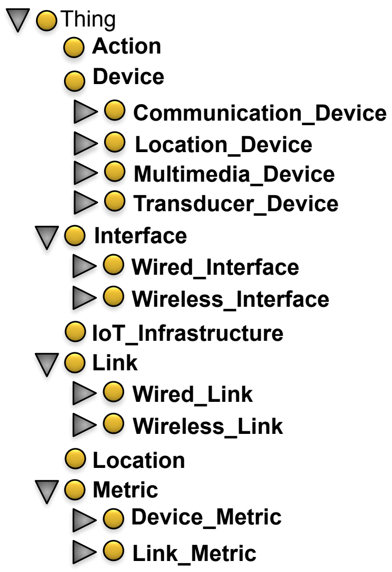


Figure 3.6: Class Hierarchy.

There are several facts of interest at this point; for instance, the support for both wired (e.g., Fiber, Ethernet) and wireless (e.g., Bluetooth, Cellular, Wireless Fidelity (WiFi)) communication interfaces and links, the wide variety of heterogeneous devices included in the infrastructure (e.g., Communication, Location, Multimedia, Transducer), and also the inclusion of both devices and link metrics that will enable the monitoring of the network status.

Once the class hierarchy is described, it is important to further detail the classes in the ontology. Since the main goal of this work is to design and implement mechanisms to manage and optimize the resilience of the computing and communication infrastructure, this becomes the main component of the developed ontology. The description of the *IoT Infrastructure* class is provided below.

IoT_Infrastructure $\sqsubseteq$ **Thing**
IoT_Infrastructure $\sqsubseteq \exists$ **isComposedBy Interface**
IoT_Infrastructure $\sqsubseteq \exists$ **isComposedBy Metric**
IoT_Infrastructure $\sqsubseteq \exists$ **isComposedBy Device**
IoT_Infrastructure $\sqsubseteq \exists$ **isComposedBy Link**

The composition of the IoT infrastructure is also depicted in Figure 3.7. Figure 3.7a shows how the IoT infrastructure is composed of devices and the communication interfaces and links that they use to exchange information. Additionally, it is of particular interest to store some metrics related to the devices (e.g., Continuity, Downtime, and Packet Loss) and links (e.g., Jitter, Latency) that can be useful for the different management and monitoring mechanisms that are going to be implemented.

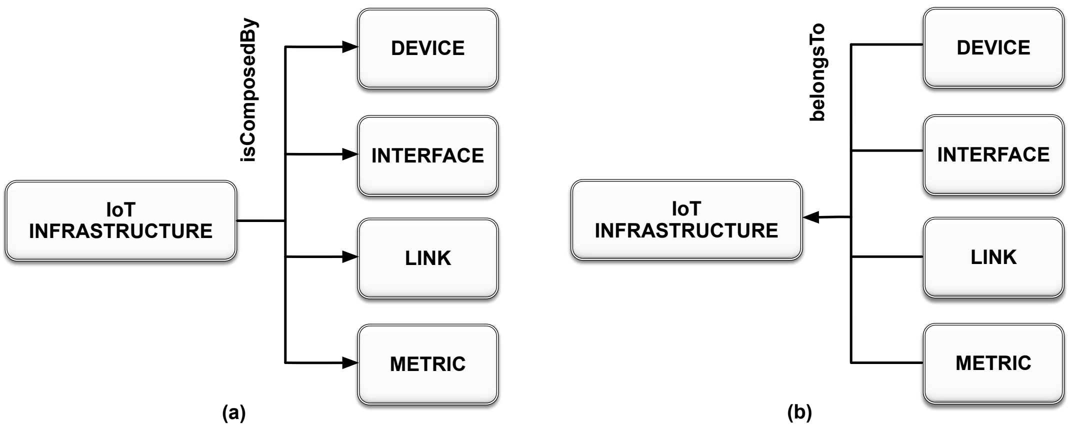


Figure 3.7: IoT Infrastructure Entity and its Relationships.

On the other side, on Figure 3.7b is depicted the reversed relationship or *Object Property*, that indicates that the devices, interfaces, links, and metrics *belongs to* the IoT infrastructure.

For the SusCity project, different devices are considered to take part of the IoT infrastructure and are grouped in the *Device* class, that is described below. It is important to mention that the classes Device, Metric, Interface, and Link are pairwise disjoint.

Device $\sqsubseteq$ **Thing**
Device $\sqsubseteq \neg$ **Metric**
Device $\sqsubseteq \neg$ **Interface**
Device $\sqsubseteq \neg$ **Link**
Device $\sqsubseteq =$ **belongsTo IoT_Infrastructure**
Device $\sqsubseteq \exists$ **hasDeviceMetric Device_Metric**
Device $\sqsubseteq =$ **isBackupOf Device**
Device $\sqsubseteq \exists$ **isPrimaryOf Device**
Device $\sqsubseteq \exists$ **hasInterface Interface**

For the *Device* entity, it was particularly important to take into consideration the use of a primary and backup device that would enable the use of self-healing mechanisms. This was modeled as depicted in Figure 3.8, with the functional object property *isBackupOf* and its inverse functional object property *isPrimaryOf*, both characterizing that some devices are primary and some are backup for the aforementioned primary devices.

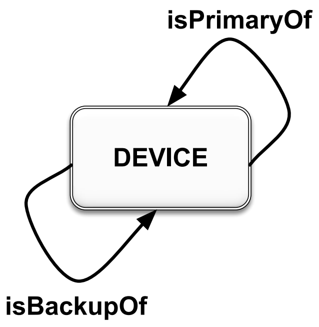


Figure 3.8: Device entity model.

This will allow instantiating a different backup device in case of a fault in the primary one, whether accidental or provoked. Additionally, the *isBackupOf* property is irreflexive, to avoid that a *Device* becomes backup of itself, rendering to a wrongful solution. The same approach was used for the *Link* class, using the concept of primary and backup links.

The collection of data gathered by the sensors could be automatized or accomplished through the use of *Data Loggers*. In the case of the automatized data collection, it is essential to include in the model how the devices communicate. The communication interfaces could be wired or wireless. The description of the *Interface* class is provided below.

Interface $\sqsubseteq$ **Thing**
Interface $\sqsubseteq =$ **belongsTo IoT_Infrastructure**
Interface $\sqsubseteq =$ **isAttachedTo Device**
Interface $\sqsubseteq =$ **uses Link**

Another important issue is keeping track of the actual physical location of the devices. This was modeled with the entity *Location* that has the Data Properties: *Latitude*, *Longitude*, and *Elevation*. This relationship, depicted in Figure 3.9, allows to identify the physical location of the device which will ease the recovery process in case of a failure. A location can host some devices (*hostsDevice*), and a device is located at exactly one physical location (*hasLocation*). Additionally, some Data Properties were also used to model the IoT infrastructure. Table 3.1 shows a list summarizing some of the Data Properties with their description.

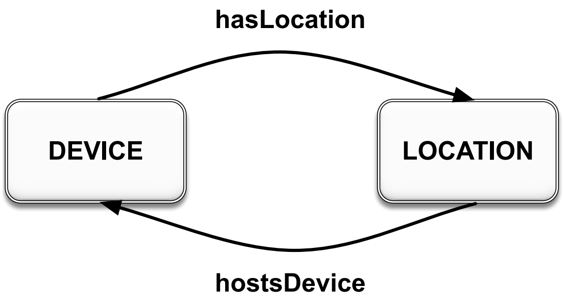


Figure 3.9: Device - Location relationship.

The final ontology, developed using Protégé [Stanford University, 2015], was consulted with different members from the SusCity project to verify that it included all the required elements to model the IoT infrastructure to properly use it in the various mechanisms aimed at managing the network. Further validation was carried out and is discussed in the next section.

3.4.1 Evaluating the Ontology

To corroborate the correctness of the ontology, a validation against consistency issues was performed using HermiT 1.3.8.3 [University of OXFORD Information Systems Group, 2015]. The results obtained showed that the ontology has no consistency issues. Furthermore, DL expressivity metrics were also calculated using Protégé [Stanford University, 2015], obtaining SHIQ(D). The expressivity evaluation allows obtaining an upper bound on the performance of querying the ontology once it is fully populated. Some restrictions modeled (e.g., inverse object properties) will enable faster response times while querying. The ontology was also evaluated by performing a set of queries. The results are discussed in this subsection.

The ontology was populated with the description of the devices being used in the SusCity project (e.g., smartmeters, sensors) and the communication links that connect them. With this information, some sample queries were designed to verify if the results match the expected values, thus validating the ontology. The goal is to use this ontology with different network, service, and application managing mechanisms. Some possible scenarios are focused on the resilience of the infrastructure and the latency of the communication network, which are two of the key features to optimize in the network. The scenario used in the tests is described in Figure 3.10.

Table 3.1: Data Properties for the SusCity IoT Ontology.

Entity	Data Property	Description
Device	hasIdentifier	Integer representing an unique identifier
	hasFirmwareName	String describing the name of the firmware
	hasFirmwareVersion	String describing the version of the firmware
	hasRole	String indicating if the device is primary or backup
	hasStatus	String indicating the device status, e.g., up, down
	hasMode	String describing the device mode, e.g., power safe
	hasMeasurementFrequency	Float indicating the time between each capture interval
Link	hasCapacity	Float indicating the link capacity in Mbps
	hasPhyTechnology	String describing the link technology, e.g., fiber, satellite
Metric	hasValue	Float containing the value of the metric measured
Location	hasElevation	Float indicating the elevation in meters of the location
	hasLatitude	Float indicating the latitude of the location
	hasLongitude	Float indicating the longitude of the location
Interface	hasNumberOfAntennas	Integer indicating the amount of antennas for a wireless interface
	hasPhyAddress	HexBinary depicting the physical address of the interface

The scenario is divided into four zones (Zone 0 to Zone 3) interconnected. Zone 0 has one smartmeter (SM0) with redundant links (L0, L1) connected to two different routers (R0, R1). Zone 1 has two smartmeters (SM5, SM6) connected to R0 and R1 via L4 and L5, respectively. Similarly, Zone 2 has four smartmeters (SM1 to SM4) connected through L2 and L3; and finally, Zone 3 has two access points (AP0, AP1), where AP0 has redundant links to R0 (L6) and R0 (L7) and also is connected to eight CO2 sensors (S0 to S7), and AP1 is connected via L8.

To improve resilience, one common approach is using backup devices that are instantiated once the primary device is down in order to increase availability. In the test scenario described in Figure 3.10 some devices were configured as primary while some others were identified as a backup. Additionally, information about the status of the devices (e.g., Up, Down, Rebooting) is also stored.

The first test consisted of querying the ontology to find out which devices are down (see Figure 3.11). The query provided this information, and the results were as expected according to the information used to populate the ontology.

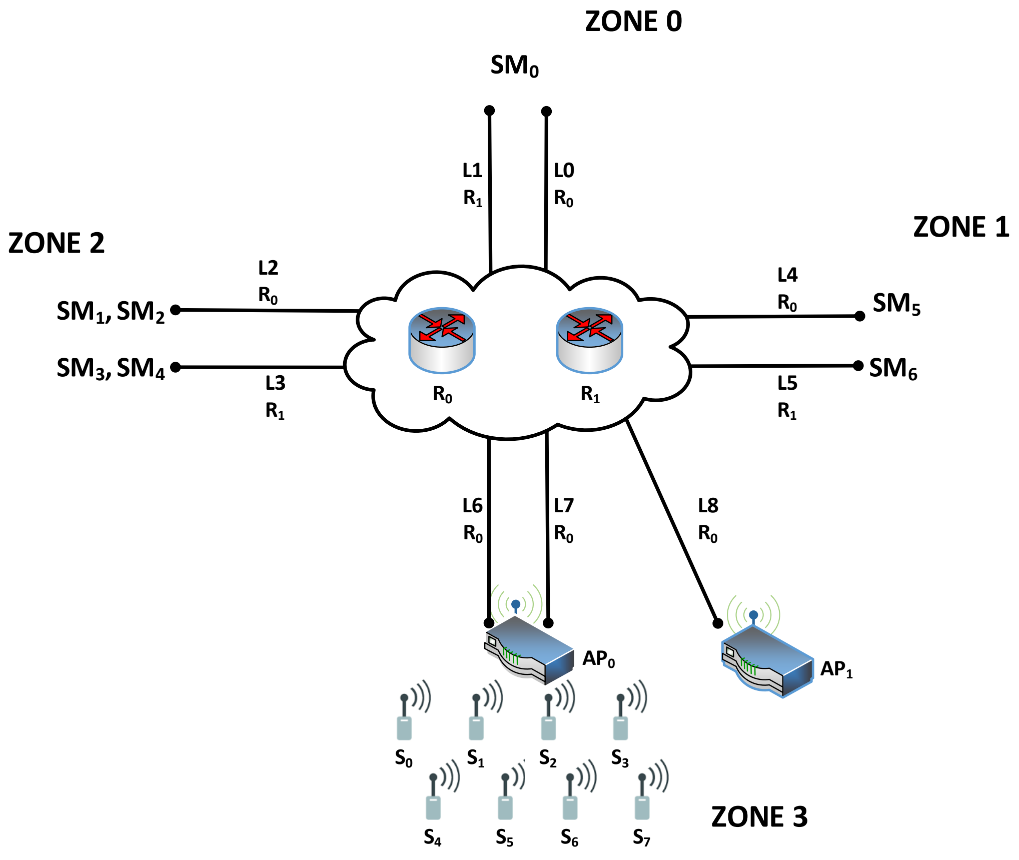


Figure 3.10: Testbed used for Validation.

This example confirms that this type of queries can be used to build intelligent IoT infrastructure management services, in this case, aimed at the improvement of resilience.

```
SELECT ?device
WHERE {
?device hasStatus ?status .
FILTER (?status="DOWN")
}
```

Figure 3.11: Query #1 - Down devices.

The query depicts in Figure 3.11 effectively listed all the devices with status "*Down*". This information is beneficial for a mechanism aimed at improving resilience since it clearly identifies the devices that need to be replaced. As stated before, a common technique is to upgrade a backup device to a primary status, thus recovering the failure. To test this, a second query was designed, with the objective of listing the backup devices associated with the primary devices with a current status "*Down*". The query also provided the status of the backup device (see Figure 3.12).

```
SELECT ?devbac
WHERE {
?devbac isBackupOfDevice ?devpri .
?devpri hasStatus ?statusp .
?devbac hasStatus ?statusb .
FILTER (?status="DOWN")
}
```

Figure 3.12: Query #2 - Status of backup devices from the primary ones that are down.

For the query showed in Figure 3.12, *devpri* refers to the primary device and *devbac* to the backup device. Similarly, *statusp* and *statusb* refer to the status of the primary and backup devices, respectively. Once again, this type of queries could be used on smart management services for the ICT in the IoT.

Other queries were designed to further evaluate the ontology, this time with a different management objective in mind. For the following queries, a mechanism to improve the network latency was used as a possible example to test the ontology.

In the scenario designed, the communication links are modeled as well as some metrics of interest such as the jitter. It could be useful to compare the jitter from different links and use this information, for instance, in a routing mechanism giving higher priority to links with lower jitter to select these links for the routes. The query depicted in Figure 3.13 lists the links ranked by their jitter in ascendant order.

```
SELECT ?link ?metric ?jitter
WHERE {
?link hasLinkMetric ?metric .
?metric sus:hasValue ?jitter .
ORDER BY (?jitter)
}
```

Figure 3.13: Query #3 - Links ranked by their jitter.

Using the same concept applied to the devices, the links were also modeled as primary and backup. The query showed in Figure 3.14 was used to list the backup links whose jitter was smaller than its primary correspondent link. This information could be used to update routes and eventually improve the service for final users.

For the query presented in Figure 3.14, *linkp* and *linkb* refer to the primary and backup links respectively, *metricp* and *metricb* indicate the metric associated to the primary and backup links, and analogously *jitterp* and *jitterb* specify the jitter for their corresponding links.

```
SELECT ?linkp ?jitterp ?linkb ?jitterb
WHERE {
?linkb isBackupOfLink ?linkp .
?linkb hasLinkMetric ?metricb .
?linkp hasLinkMetric ?metricp .
?metricb hasValue ?jitterb .
?metricp hasValue ?jitterp .
FILTER (?jitterb < ?jitterp)
}
```

Figure 3.14: Query #4 - Links with smaller jitter (i.e., primaries vs backups).

Results for all the queries described in this subsection returned the expected information according to the data that was used to populate the ontology. These results confirm the possibility to use this ontology as input for intelligent management web services (e.g., with a workflow of interaction with the user), where user's queries are translated into SPARQL queries on demand.

3.5 Summary

New solutions to support daily activities of citizens, like smart traffic control, health care, public safety, among others, are being incorporated in the paradigm that intends to interconnect objects in the scope of a city, also known as a Smart City. The Cloud computing and the IoT seem to be two of the most critical technologies that are going to constitute the foundation for this reality. However, there are still open issues that need to be tackled to bring this paradigm to reality. One of these open issues is the necessity of maintaining the robustness of the services that are hosted in the ICT infrastructure that interconnects Smart Objects and services within the IoT. In this chapter, a novel architecture with resilience features for the IoT environments was introduced.

The proposed architecture takes into consideration the communication requirements of the IoT, besides using the advantages of the Cloud and Fog paradigms to add ubiquity and scalability to the environment. The interaction between components in the *IoT Middleware* combined with the proper technologies allows to fulfill an efficient communication process between the Smart Objects and the end-users. Particularly, the proposed architecture was designed taking into consideration the SusCity project [SusCity-Project, 2015] scenario and the requirements needed for its communication platform.

Furthermore, an ontology to describe the IoT infrastructure was also introduced. The ontology comprises as main classes the IoT infrastructure, its devices, communication interfaces and links, and the performance metrics related to them. This information will ease the managing of the infrastructure in order to guarantee its proper work for user services and applications. The ontology was successfully validated with different procedures, including consistency evaluation and also queries to corroborate its correctness.

It is essential to mention that although the proposed ontology was designed for the requirements of the SusCity project, it can easily be adapted to other Smart City scenarios since it includes all the main classes involved in a Smart City infrastructure (e.g., sensors, actuators, multimedia device, communication device). For each implementation, it would only be necessary to adjust the ontology population to comply with the different scenarios.

The following contributions were produced from the literature review and the proposals presented in this chapter:

- A resilience architecture for the deployment of IoT services in Smart Cities. The main focus of the architecture is the depiction of its middleware, able to handle the vast heterogeneity of Cloud/Fog environments, and with special emphasis on the resilience component. The *Resilience Manager* provides functions for protection and recovery to maintain the survivability of the applications in case of failures;
- An ontology to describe the IoT infrastructure, to standardize the information of the underlying infrastructure to be shared among different protocols, network services and user applications; and
- Additionally, the results obtained from the research presented in this chapter also helped in the development of deliverables for the SusCity project (MITP-TB/C S/0026/2013 - SusCity).

The outcomes of this chapter allowed the identification of tasks needed to provide resilience to the Cloud to IoT continuum. The design of a resilience architecture frames these tasks into modules. The *Resilience Manager* module handles the protection and recovery of applications in case of failures to offer them survivability. The architecture yielded us the conceptual framework to design and implement the resilience mechanisms to enhance the availability of services and applications in the Cloud to IoT ecosystem. In the next chapters, we present more details about these mechanisms.

Chapter 4

Formalizing Service Chain Composition

Contents

In a smart city scenario, many services and applications are designed to improve the quality of life of the citizens, for example, urban traffic control, emergency health assistance, home energy monitoring, and evacuation routes in case of natural disasters; turning the survivability of these services into a critical aspect. The architecture described in the previous chapter offers the frame of reference to handle the challenges imposed by the Cloud and IoT paradigms that support the Smart Cities, but it is necessary to design and develop specific solutions that deal with the tasks specified for each module.

In particular, the *Resilience Manager* from the architecture, along with the *Protection & Recovery* module is in charge of infusing the Smart City solutions with robustness and survivability in the face of failures. This chapter focuses on defining a framework that can handle requests for virtualized services in an efficient manner while enhancing the survivability of the Smart City applications.

4.1 Softwarization in the Cloud to IoT Continuum

The Cloud-Fog-Mist-IoT service infrastructure can be represented as a set of hardware and software components organized in tiers (i.e., from the Cloud on the top to the IoT on the bottom going through the Fog and Mist tiers) that enables the deployment and interconnection of smart end-user applications and devices empowering low-latency and context awareness data processing in a distributed way. The capabilities, service types, and deployment models available in the Cloud-Fog-Mist-IoT have been influenced by virtualization techniques usually known as the softwarization of the Cloud to IoT continuum [De Turck et al., 2017]. Particularly, the network softwarization [Wright et al., 2015] allows the design, development, test, management, and deployment of services and applications via NFs using the available resources (i.e., hardware or software components) in the infrastructure to route the network flows through the right components [Quittek, 2014]. This approach could be extrapolated along all the tiers in the Cloud to IoT continuum in order to provide services and applications to end-users via a chain of VFs. Some examples of the softwarization approach on this domain are NFV [Yi et al., 2018], SDN [Kreutz et al., 2015], and SFC [Gupta et al., 2018].

Services, such as video streaming, online gaming, mobile connectivity, and IoT applications, are composed of different software and hardware components usually hosted on top of a network infrastructure organized according to the Cloud to IoT continuum approach. Consequently, the infrastructure operator is responsible for the embedding, management, and orchestration of a set of services that can be instantiated and used by various service providers and their final users, frequently applying softwarization techniques via VFs with the objective of fulfilling service and application requirements [Fischer et al., 2013; Quittek, 2014; Gupta et al., 2018]. These VFs can be focused on different domains, such as network infrastructures or computational activities. The VFs are grouped in structures known as SCs (see Figure 4.1), which combine the specific functionality requirements needed to fulfill more complex service and application requirements.

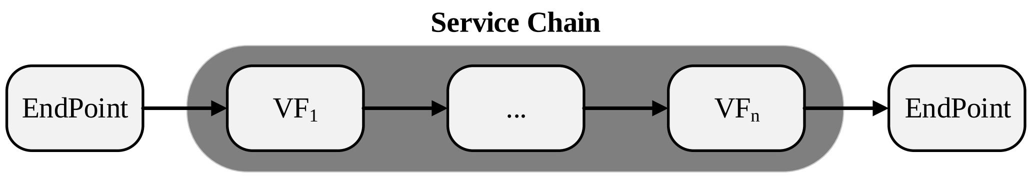


Figure 4.1: An Example of a Generic SC and its Components.

In the Cloud to IoT continuum, a common assumption adopted by infrastructure operators is based on the notion that the dependencies between a set of services or VFs in a SC are fixed [Quinn and Nadeau, 2015]. For example, consider a SC at the border of a Wide Area Network (WAN) that requires to inspect and compress the traffic; typically, the packets should pass by the Intrusion Detection System (IDS) before the WAN optimizer compresses the content.

Even though the fixed/ordered approach for the composition of SCs could be used in some particular use cases, it could be inflexible in some other situations. Thus, in cases where the VFs of a given SC do not have a strict order or dependency between them, there are multiple possibilities for the composition of the components of the SC [Mehraghdam et al., 2014]. Consider a SC composed by a set of VFs where two of them correspond to a Firewall (FW) and a Video Optimization Controller (VOC), respectively, which are independent from each other as depicted in Figure 4.2. Despite that the result of the traffic flow going from the FW to the VOC or from the VOC to the FW is the same from the end-user perspective; the processing order of these components could have some impact on the communication infrastructure. For example, regarding important decision making, such as placement decisions considering the FW could filter part of the incoming flow consuming fewer resources on the outgoing link.

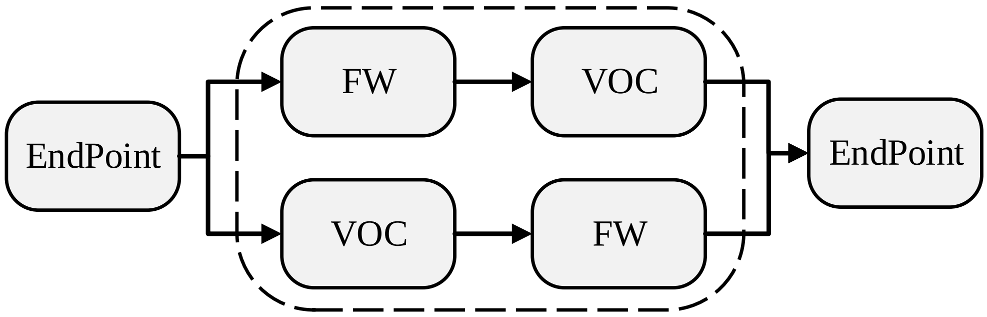


Figure 4.2: An Example of a SC with Components in an Arbitrarily Order.

Once the infrastructure operator has defined how to deal with the composition of the users' requests (i.e., SCs), it is necessary to move forward to the mapping or embedding of the requests into the physical or substrate infrastructure. At this stage, the infrastructure operator has to decide the best embedding for the SCs requested considering the current state of the physical infrastructure. For delivering services and applications in the Cloud to IoT continuum, a set of VFs and infrastructure components have to be instantiated or activated, so the corresponding communication flows can be routed between the endpoints. For this purpose, the architecture described in Chapter 3 could be instantiated and used like a MANO solution focused on the resilience of services and applications. In this context, a couple of challenges related to the softwarization and composition of SCs arise: (1) how to standardize and formalize a request of a service chain considering the communication and dependencies between the VFs?; and (2) how to embed the service chain and its components efficiently into the substrate infrastructure?

A framework to deal with the two challenges discussed above is presented in the next section. Specifically, a strategy to address the SC composition is proposed in this chapter before moving forward to the SC embedding challenge, which is tackled in Chapter 5.

4.2 A Framework for the Composition and Embedding of Service Chains

The Cloud/Fog ecosystems have been moved to more software-oriented solutions to decouple essential orchestration functions and empower a finer granularity level at the resource management level to fulfill services and infrastructure requirement changes [He et al., 2019]. Consequently, the microservice software architecture has been adopted to effectively built large-scale Cloud/Fog applications and services from reusable and independently deployable components [Chowdhury et al., 2019]. Taking advantage of the microservice architecture, applications can be designed and implemented via a SC which defines an ordered or partially ordered set of abstract VFs and ordering constraints be applied to the interactions between these components [Quinn and Nadeau, 2015].

The modeling of SCs composition on the Cloud has been addressed using fixed and pre-defined approaches [Sun et al., 2012; Keller et al., 2014]. More recently, implementing pliable strategies for the Cloud/Fog and NFV environments [Mehraghdam et al., 2014; Beck and Botero, 2017] have been proposed. These different methods were discussed in Section 4.1 leading to the identification of the main challenges in this topic. To cope with these challenges, a framework to deal with the composition of SCs from the user's requests to the embedding and instantiation of components in the substrate communication infrastructure is presented below.

The scenario considered for the design and implementation of the composition framework is depicted in Figure 4.3. The infrastructure provider offers a set of VFs via a catalog, which is maintained by an instance of the MANO [Quittek, 2014], from where users can select them. Users structure their requests by grouping different VFs, forming a SC according to a set of rules that guarantee

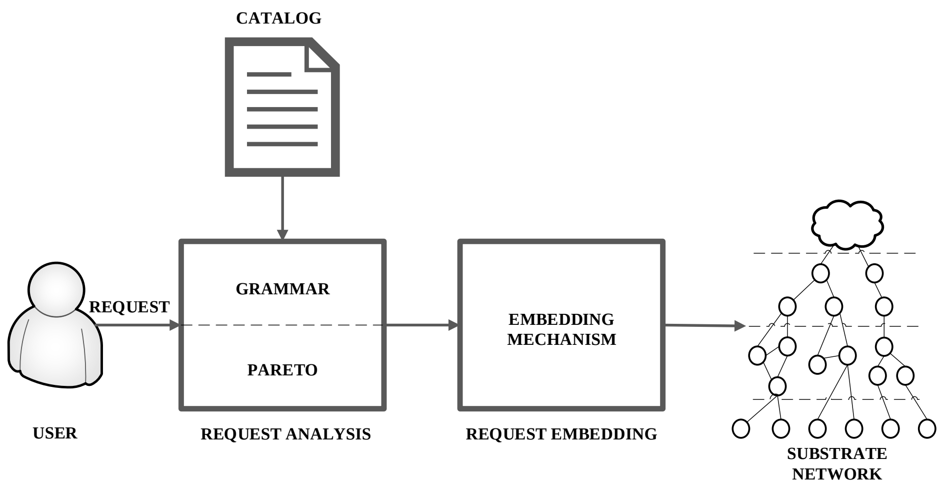


Figure 4.3: Composition and Embedding Framework.

the correctness of the request.

To validate the user requests, the embedding framework uses a context-free grammar focused on enabling flexibility and resilience for the SCs. The grammar proposed is generic enough to be enhanced and extended in order to incorporate it in the design of a data modeling language used for the management and orchestration of the components of an infrastructure provider.

Once the request of the user has been validated, a Pareto analysis is performed. Considering that a SC could contain components on a fixed or partial order, the Pareto technique allows to identify the most suitable SC on a collection of possible combinations according to a given set of optimization goals. The *Request Analysis* module is one of the elements of the embedding framework proposed (see Figure 4.3), whose tasks are performed according to Algorithm 4.1 producing a set of requests that will be the input of the next module.

Algorithm 4.1: Requests Analysis.

```
  Result: Set of SCs to embed
1 SCs ← get_requests()
2 req_list ← parser(SCs, grammar)
3 catalog ← get_catalog()
4 req_embed ← Pareto_analysis(req_list, catalog)
5 return req_embed
```

The outgoing of the *Request Analysis* module produces the set of SCs that will be embedded in the substrate network. The elements of the set are selected using service and application metrics (e.g., data rate, resource consumption, number of instances) via a Pareto analysis as previously explained.

The *Request Embedding* module is in charge of mapping the SCs and their VFs in the physical nodes of the infrastructure prioritizing the availability of the services in order to enhance the resilience of applications from failures in the physical components of the infrastructure.

The components of the *Request Analysis* module are described in detail in the following sections, while the mechanisms for the *Request Embedding* module are introduced in the following chapter.

4.3 A Grammar to Specify Service Chains

This section describes the solution designed to tackle the first challenge mentioned in Section 4.1, particularly, a context-free grammar to deal with the representation of the Service Chain requested, enabling the possibility to specify replicas for the VFs. After validating the correctness of the SCs requests, a Pareto analysis is performed before invoking the embedding mechanisms presented in Chapter 5, that deal with the second challenge.

In the Cloud to IoT continuum considered for this research, the substrate infrastructure where the VF chains will be instantiated and embedded is modeled as a graph denoted by $G = (N, L)$, where N and L are the sets of nodes and links of the communication infrastructure, respectively. Each physical node $n \in N$ has a computing capacity Ω_n (i.e., CPU, memory, storage); similarly, each physical link $\ell \in L$ has a transmission capacity Γ_l (i.e., propagation delay, bandwidth). The set of Virtual Functions V that could be instantiated in the infrastructure is accessed via a catalog, which is maintained by an instance of the MANO made available by the infrastructure provider. Detailed information about each virtual function $v \in V$, such as their resource requirements ω, is also accessible via the catalog.

Service chains are deployed in the infrastructure as a sequence of VFs, where a service s denotes a chain of Virtual Functions $vf_1, vf_2, ..., vf_{|V|}$ connected via a set of virtual links E. The order of the VFs that compose the service chain could be fixed or variable according to a given criterion that represents the interaction between the components; for example, in an IoT application that requires sensing, analytics, and storage services, the order of the last two components could swap depending on the application domain or user.

An infrastructure provider receives requests to deploy services, which are modeled as a graph $S = (V, E)$, where V denotes the set of VFs, and E represents the set of edges that connect the VFs. These requests specify particular requirements that must be fulfilled, such as the number of replicas of a particular VF to increase its availability and the order of a set of VFs inside a given chaining request. Additionally, it is considered that a VF could favor a specific tier (i.e., Cloud, Fog, Mist, IoT) for its embedding.

Besides the assumptions already presented, some additional considerations were taken into account to model the Cloud to IoT environment:

1. All the nodes in the communication infrastructure have the capacity to execute network, computational, storage, or even sensing functions (i.e., each Virtual Function can be embedded in any node);

2. Functions and services are used interchangeably, considering that the required functionalities to fulfill the desired actions can be virtualized and executed across all the network infrastructure;
3. The amount of resources, as well as the availability per node, decreases from the nodes in the Cloud to those in the IoT in the hierarchical tiered infrastructure, taking into account the constraints present in each tier; and
4. The Cloud tier has infinite resources and its availability is not affected by failures.

A context-free language was designed in order to have a standard and formal method to represent application/user's chaining requests. With this representation, it is possible to build customized complex requests composed of a set of ordered/un-ordered VFs to provide services. Each chain request is formed by a different kind of *modules*. Various of these modules can be placed in a chain request to denote a particular set of requirements for a given service. To process/recognize the Virtual Function - Chain Composition (VF-CC) requests, the production rules of the formerly mentioned grammar, in Backus-Naur Form (BNF) [Aho et al., 2007], are listed from (4.1) to (4.11).

$$
\begin{aligned}
\langle\text{start}\rangle &\models \textbf{service\{}\langle\text{chain}\rangle\textbf{\}} && (4.1)\\
\langle\text{chain}\rangle &\models \langle\text{order}\rangle\ \langle\text{chain}\rangle \mid && (4.2)\\
&\quad\ \langle\text{modules}\rangle\ \langle\text{chain}\rangle \mid \\
&\quad\ \langle\text{order}\rangle \mid \\
&\quad\ \langle\text{modules}\rangle \\
\langle\text{order}\rangle &\models \textbf{fix\_order\{}\langle\text{modules}\rangle\textbf{\}} && (4.3)\\
\langle\text{modules}\rangle &\models \langle\text{optorder}\rangle \mid \langle\text{replica}\rangle \mid \langle\text{term}\rangle\ \langle\text{moreterm}\rangle && (4.4)\\
\langle\text{optorder}\rangle &\models \textbf{opt\_order\{}\langle\text{term}\rangle\ \langle\text{moreterm}\rangle\textbf{\}} && (4.5)\\
\langle\text{replica}\rangle &\models \textbf{replica\{}\langle\text{chain}\rangle\ \textbf{:}\ \langle\text{num}\rangle\textbf{\}} && (4.6)\\
\langle\text{moreterm}\rangle &\models \textbf{,}\ \langle\text{term}\rangle\ \langle\text{moreterm}\rangle \mid \epsilon && (4.7)\\
\langle\text{term}\rangle &\models \langle\text{vf}\rangle\ \langle\text{tier}\rangle && (4.8)\\
\langle\text{tier}\rangle &\models \texttt{\textbf{cloud}} \mid \texttt{\textbf{fog}} \mid \texttt{\textbf{mist}} \mid \texttt{\textbf{iot}} \mid \epsilon && (4.9)\\
\langle\text{vf}\rangle &\models \boldsymbol{vf_1} \mid \boldsymbol{vf_2} \mid \ldots \mid \boldsymbol{vf_{|v|}} && (4.10)\\
\langle\text{num}\rangle &\models \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \boldsymbol{n} && (4.11)
\end{aligned}
$$

The terminals of the grammar, plus the empty set, are given in bold font. The grammar enables the definition of VF service chains. The chains can have a fixed order (i.e., *fix_order*) defined by the user or not, giving the service providers the possibility to rearrange the VFs to their convenience (i.e., *opt_order*). This grammar also enables the definition of *replicas* for each VF in the chain, specifying the number of copies desired. Furthermore, there is the possibility to request the deployment of the VFs in a particular *tier* of the infrastructure: *cloud*, *fog*, *mist*, or *iot*.

The grammar described deals with the first challenge introduced at the beginning of this chapter, specifically about how to standardize and formalize a service chain request with support of replication for the VFs. In the next section, the Pareto analysis performed to build the set of SCs that will be the input of the *Request Embedding* module is presented using as input a collection of requests (i.e., SCs) expressed using the context-free grammar defined in this section.

4.4 A Pareto Analysis for Service Chain Composition

Using the grammar described in the previous section, the service operator has a standard way to process and validate users' requests, even more, in certain cases has the possibility to decide the composition of a SC (i.e., when the *opt_order* token is present in the request) in the best possible way to fulfill the requirements of the user and the computing and communication infrastructure. To better understand the applicability of the context-free grammar proposed, let us consider how to formalize two adapted service chains from well-known uses cases presented and discussed by the IETF Networking Group [Liu et al., 2014] and the European Telecommunications Standards Institute (ETSI) NFV Working Group [ETSI GS NFV, 2013].

The SC depicted in Figure 4.4 represents a generic IoT application for sensing that involves the sensor (*Generic Sensor (GENSEN)*), a data aggregation function (*Data Aggregator (DAGG)*), a data compression function (*Data Compression (DACOM)*), data analytics (*Data Analytic (DANA)*), and a final database for storage (*Data Base (DB)*). The formal request for this SC using the grammar is:

service{fix_order{GENSEN,DAGG,DACOM,DANA,DB**}}**

For this particular request, the interaction between the components is specified as fixed. Thus the service provider is obliged to follow the order of the VF in the SC composition without violating the user requirements.

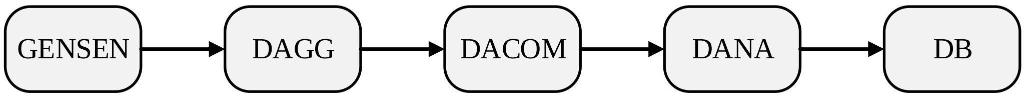
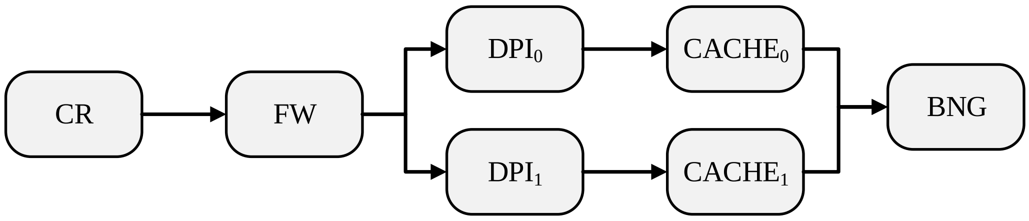


Figure 4.4: A SC for a Generic IoT Application.

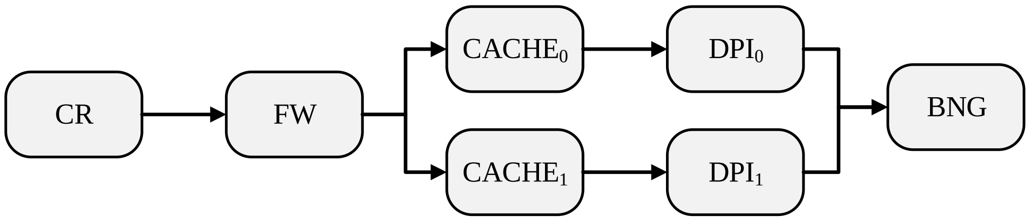
A pliable SC for a mobile operator use case scenario is depicted in Figure 4.5. The request involves three modules in **fix_order** (i.e., Cognitive Radio (CR), FW, and Broadband Network Gateway (BNG)), and two other modules (i.e., Deep Packet Inspector (DPI) and CACHE) that require 2 replicas and their interaction is flexible, this is, their composition is dictated by the descriptor **opt_order**. The formal request for this SC using the grammar is:

service{fix_order{CR,FW**}fix_order{replica{opt_order{**DPI,CACHE**}**:2**}}**BNG**}**

For requests where part of their VFs can be arbitrarily ordered, the service provider should consider all the permutations of the set of components or VFs as candidates for the set of SCs that will be embedded in the substrate infrastructure (see Figures 4.5a and 4.5b). However, this flexibility on the service composition generates a drawback regarding the cardinally of the candidates set. Thus, for a SC with n components in optional order, there are $n!$ possible candidates that represent valid solutions. To automate the selection of the optimal SC according to a given criteria, a Pareto analysis is used.



(a) First Composition of a SC with Pliable Components.



(b) Second Composition of a SC with Pliable Components.

Figure 4.5: A SC for a Mobile Operator with Replication and Optional Order in their VFs.

The Pareto analysis is a formal technique based on the “80/20” rule aimed at identifying the top portion of causes that need to be addressed to resolve a given problem [Mehraghdam et al., 2014; Noghin, 2018]. Using this approach in conjunction with meaningful metrics at the service level (i.e., SC), it is possible to find a trade-off in order to identify the most suitable SC in a set that fulfills the requirements of users and the optimization goals of the service provider. In this research, the following metrics are considered for the evaluation and comparison of a set of SCs to guide the final selection of the SC to embedded in the substrate network:

- The data rate between the endpoints of the all possible combination for a given SC;
- The computational requirements of all VFs for the possible combination of a given SC;
- The number of required instances of all service components (i.e., VFs) over all the combination for a SC, including replicas.

In a scenario where various metrics should be optimized, the Pareto analysis results lead to a set of solutions that cannot improve one metric without compromising at least one of the others. Thus, in the framework proposed, first the Pareto solution is computed and after that a SC is selected using a single metric, such as the data rate. The input data for this analysis are the requests previously validated and selected metrics provided by the *Catalog* included in the Embedding Framework. Figure 4.6 shows a graphic view of the Pareto analysis performed to the SC for a Mobile Operator discussed above (see Figure 4.5). For this SC the composition with $id = 0$ (see Figure 4.5a) was selected for embedding considering it got the best trade-off of the metrics selected.

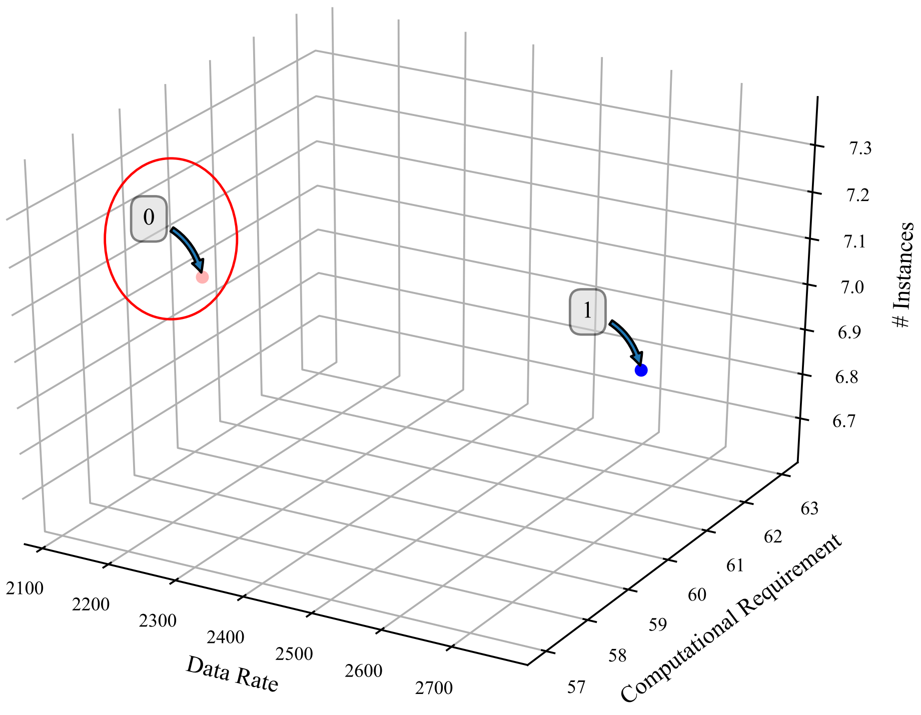


Figure 4.6: Result of the Pareto Analysis for the Mobile Operator SC.

With the Pareto analysis, the tasks from the *Request Analysis* module from the framework discussed in Section 4.2 are completed. The result from this module is a set of verified SCs to embed that are optimized according to a group of metrics of interest that take into consideration the needs of both users and service providers.

The metrics chosen in this work for the Pareto analysis can easily be replaced according to the demands of users and/or service providers, resulting in a different set of SCs to embed but maintaining the functionality of the proposed framework.

Mechanisms for the next module in the framework (*Request Embedding*) that tackles the second challenge identified in Section 4.1 regarding the embedding of the SCs resulting from the first module are presented in the next chapter; however, a context regarding the allocation of resources for SCs in the Cloud to IoT continuum is discussed in the next section for a better understanding of this particular issue.

4.5 Dealing with the Embedding of Service Chains

Virtualization has evolved as a fundamental enabler in the context of the Cloud to IoT continuum as discussed in Section 4.1. A key aspect of this scenario is deploying the different service instances over geographically distributed computational resource centers/nodes, guaranteeing a quality interconnection. In the previous sections, a discussion about how to address the manage composition of SCs was presented; now, it is time to move forward to understand how to deal with the embedding process of the SCs in the substrate infrastructure of the service provider, which is the main task of the *Request Embedding* module of the framework depicted in Figure 4.3.

Figure 4.7 depicts the general idea regarding the mapping/embedding of the SCs in the physical infrastructure. A set of requests (i.e., SCs) should be allocated in the substrate resource infrastructure of the Cloud to IoT continuum. The objective of this process is to optimize certain metrics (e.g., utilization, operational expenses, capital expenditures) from the point of view of the service provider and, at the same time, to fulfill the requirements of users, such as the response time and availability level of services/applications.

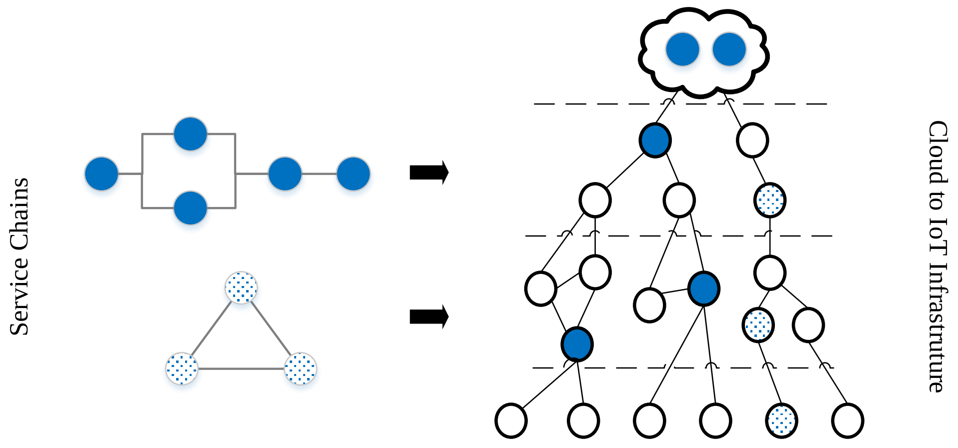


Figure 4.7: Embedding SCs in the Cloud to IoT Infrastructure.

Commodity hardware available in the substrate infrastructure of a service provider offers different functionalities leading to diverse capabilities. Some hardware components offer better support for forwarding network packets, others provide more powerful computing resources, and others come with functions more suitable for the IoT (i.e., sensing and actuating). Consequently, by virtualizing services, functionalities can be decoupled from location, allowing the software to be deployed at the most appropriate places.

The efficient use of the substrate infrastructure in the Cloud-Fog-Mist-IoT is dependent on effective techniques for Virtual Function embedding which maps the

VFs on the physical substrate infrastructure resources [Rahman and Boutaba, 2013; Beck and Botero, 2017]. The VF embedding problem is challenging due to finite nodes and link resource constraints and the online nature of the SCs requests. Additionally, the requests of users specify a set of requirements that should be satisfied and considering in the embedding process, such as the availability of services/applications in case of a failure on virtual or physical components of the infrastructure. The requirements of availability of a given SC denote the resilience degree of an application impacting directly in the QoS of the end-user. Thus, the embedding mechanisms in the Cloud to IoT context must consider how to deal with failures in the service provider infrastructure.

In the context of service mapping with failures recovery, replication mechanisms have been proposed to achieve the availability expected by the components on SCs [Rahman and Boutaba, 2013; Carpio et al., 2017; Aidi et al., 2018]. These and other approaches were studied and used as inspiration to design and implement a set of mechanisms for the embedding of SCs considering the requirement to achieve a high level of resilience that should be provided by service operators to applications in the Cloud to IoT continuum, which are presented in the next chapter.

4.6 Summary

The Cloud-Fog-Mist-IoT continuum offers the infrastructure for virtualized services at different levels (i.e., software and hardware), called Virtual Functions. These VFs are grouped in a structure called Service Chains, which are a set of VFs interconnected in order to fulfill a set of specific service/application requirements. The service provider has to furnish the proper management and orchestration for said SCs, offering certain availability even in the face of failures. Replicating the VFs among the network infrastructure allows the activation of replicas in case of failures to increase the availability of the SCs. The VFs and their replicas must be strategically embedded in order to enhance resilience. In this chapter, two challenges were identified in this context: (1) the formalization of the service request, and (2) the embedding of the SCs in a substrate infrastructure.

A framework to address the composition and embedding of SCs in the Cloud to IoT continuum is proposed in this chapter. This framework is composed of two main modules, namely *Request Analysis* and *Request Embedding* being the first one described in the context of this chapter and the second just introduced, leaving its details for the next chapter.

For the *Request Analysis* module, a formal grammar that enables the customized specification of SC requests and the replicas of their constituent VFs was presented. The grammar allows declaring chains with fixed order or not, providing the freedom to the service provider to reorganize the constituent VFs in order to optimize the resource usage of the underlying network. Additionally, the grammar supports the specification of the number of desired replicas for each VF and a particular tier (Cloud, Fog, Mist, IoT) in which the VF should be embedded (if possible). In the case of multiple options to create the SC (i.e., optional order of the constituent VFs is allowed), a Pareto analysis of the candidate SCs is used to select the final SC to embed. The Pareto analysis uses the data rate, number of instances, and resources used as metrics to select the SC to embed.

The contributions presented in this chapter include:

- A modular framework for the composition and embedding of service chains, that can be adapted to different optimization goals;
- A BNF grammar to specify Service Chains, as well as to validate their correctness according to specifications provided via a catalog; and
- A proposal to optimize the selection of alternative Service Chains that satisfy the same requirements by using a Pareto analysis.

The grammar and the Pareto analysis solve the first challenge identified at the beginning of the chapter, regarding standardization and formalization of the request of service chains. The rest of the document describes different mechanisms aimed at the embedding of VF to enhance the survivability of applications in the Cloud to IoT continuum.

Chapter 5

Resilient Service Chains through Smart Replication

Contents

THE embedding process of the Service Chains requested via the context-free grammar previously described is detailed in this chapter. The main requirement considered for the embedding of the VFs in the substrate network is the survivability of the Service Chains components (i.e., VFs) via the replication of all or part of them in disjoint physical nodes (if it is possible) of the substrate infrastructure.

Three different mechanisms for Virtual Function embedding are presented in this chapter, one based on Integer Linear Programming aimed to obtain an upper bound of availability, one based on Genetic Algorithms focused on finding near-optimal solutions consuming less computational time and resources, and the last one based on a graph community detection algorithm that advantage of the Cloud to IoT tiered architecture to achieve high resilience for online solutions. The proposed mechanisms are focused on single failures of nodes in the substrate communication infrastructure.

The mechanisms proposed in this chapter were designed and implemented in an incremental way to study their results and identify weaknesses. The analysis of each mechanism inspired ideas that were applied in the design of the next. Considering this methodology, we describe the mechanisms in this chapter in a formal way, and present their results and comparison in the next one to avoid repetition.

5.1 A Formal Model for Virtual Function Embedding

The first mechanism is based on the formulation of a mathematical model to find the optimal solution for the embedding problem. Even though the computational cost to find the optimal solution is high for online scenarios, the optimal solution was key for the study to define an upper bound. The mathematical model uses a bi-level ILP formulation. On the first level, the goal is to maximize the acceptance rate of the Service Chains; on the second level, the aim is to maximize the survivability of the Service Chains by placing the VFs in the most reliable nodes.

Table 5.1 introduces the parameters and variables used in the optimization solution. Service chains $s \in S$ are composed by virtual functions $v \in V$. Each virtual function can be replicated up to R times. The virtual function instances are to be deployed in nodes $n \in N$ belonging to the physical topology. For the feasibility restrictions, the abstraction of *resource unit* is applied, where a resource unit reflects the resources (i.e., CPU, memory, storage) of node $n \in N$, depicted in Ω; or the resource requirements of VF $v \in V$ listed in ω.

Table 5.1: Parameters and Variables for the ILP Model.

Parameters	
Parameter	**Description**
V	Set of Virtual Functions
S	Set of Service Chains requested composed by a set of Virtual Functions $v \in V$
R	Set of instances of Virtual Functions (i.e., replicas) for a single Service Chain
N	Set of nodes in the physical topology
Ω	Capacity vector. Resource capacity for $n \in N$
F	Survivability vector. Availability for $n \in N$
I	Instance matrix. An $\|S\| \times \|V\|$ matrix indicating the amount of instances of each Virtual Function $v \in V$ in a Service Chain $s \in S$
ω	Requirement vector. Resource requirement for the Virtual Function $v \in V$
T	Tier vector. Indicates the tier in the infrastructure to which node $n \in N$ belongs to
Variables	
Variable	**Description**
A	Acceptance vector. An $\|S\|$ vector
P	Placement matrix. An $\|R\| \times \|V\| \times \|S\| \times \|N\|$ matrix

5.1.1 Maximizing Acceptance Rate

Equation (5.1) depicts the goal of the first optimization level, which is maximizing the acceptance ratio. Vector A is a variable that holds the accepted service chains $s \in S$. In case of shortage of resources, it would lead to the prioritization of the service chains that request a lower amount of resource units; thus, it penalizes the requests that include an excessive amount of replicas for their virtual functions.

$$\max \sum_{s \in S} A_s \tag{5.1}$$

The following constraint (see Equation (5.2)) guarantees that only the replicas requested are placed.

$$\sum_{r \in R} \sum_{n \in N} P_{i,n}^{s,v} \leq I_{s,v} \quad \forall s \in S, \forall v \in V \tag{5.2}$$

The next constraint (see Equation (5.3)) limits the placement of the replicas in different nodes. In case of a failure in a node, the replica is not affected and can be activated.

$$\sum_{n \in N} P_{i,n}^{s,v} \leq 1 \quad \forall s \in S,\ \forall v \in V,\ \forall r \in R \tag{5.3}$$

Equations (5.4) and (5.5) are interdependent and ensure that only the service chains placed are accepted for embedding. $M \in \mathbb{N}$ is a constant with a large value usually called "big M".

$$\sum_{n \in N} P_{r,n}^{s,v} \leq M \times A_s \quad \forall s \in S,\ \forall v \in V,\ \forall r \in R \tag{5.4}$$

$$A_s \leq \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \quad \forall s \in S \tag{5.5}$$

Equation (5.6) guarantees that all the replicas of the virtual functions $v \in V$ that belong to the accepted service chain $s \in S$ are placed.

$$A_s \times I_{s,f} = \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \quad \forall s \in S,\ \forall v \in V \tag{5.6}$$

Equation (5.7) enforces the feasibility constraints. It only allows the placement of virtual functions $v \in V$ when node $n \in N$ has enough resources to host them.

$$\sum_{s \in S} \sum_{v \in V} \sum_{r \in R} P_{r,n}^{s,v} \times \omega_s \leq \Omega_n \quad \forall n \in N \tag{5.7}$$

5.1.2 Maximizing Survivability

The second level of the optimization problem aims to maximize the survivability of the Service Chains. The vector F holds the availability indicator of each node $n \in N$ in the physical topology. This factor is periodically updated via orchestration mechanisms, executed by the MANO, that collect and maintain information regarding the availability of the infrastructure nodes. This information is used to calculate the most recent availability values and update the F vector periodically with the current state of the infrastructure.

The availability is computed as the ratio between the uptime of the nodes and the aggregate of the expected values of uptime and downtime [Bauer and Adams, 2012]. Equation (5.8) shows the calculation for the availability.

$$Availability = \frac{E[uptime]}{E[uptime] + E[downtime]} \tag{5.8}$$

Equation (5.9) shows the formulation for the objective function of this level of the optimization problem. The T vector indicates the tier in the network infrastructure to which node $n \in N$ belongs; there are four possible tiers and their associated values go from lower to higher as the tiers go closer to the user. The tiers and their respective values are: Cloud (0.6), Fog (0.8), Mist (0.9), and IoT (1). These values represent a penalty associated with the propagation delay between the nodes according to the tier where they belong, based on previous work [Souza et al., 2016; Mahmud et al., 2018]. Thus, this objective function tries to balance the embedding in nodes with the highest survivability factor and also those nodes that are closer to the user, hence taking advantage of the entire network infrastructure by performing a vertical search (i.e., between all the tiers) instead of focusing only in a subset of nodes (e.g., Fog nodes).

$$\max \sum_{s \in S} \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \times F_n \times T_n \tag{5.9}$$

Equation (5.10) is added to ensure that the results from the first optimization problems are kept. This means, the service chains $s \in S$ that were accepted in the first step are fixed, and the selection of the nodes $n \in N$ can be changed in order to satisfy the second optimization goal.

$$A_s \times I_{s,v} = \sum_{r \in R} \sum_{n \in N} P_{s,n}^{r,v} \quad \forall s \in S,\ \forall v \in V \tag{5.10}$$

In order to keep the results from the first level, the acceptance vector resulting from the first optimization level is fixed and used as a parameter for the second optimization level. Constraints depicted from (5.3) to (5.7) are kept at this stage to guarantee feasibility.

The bi-level formulation presented for Service Chain embedding is more suited for offline scenarios taking into consideration the time required to get a solution in complex online scenarios. Additionally, as is the case with optimization, the optimal node(s) in the topology to embed VFs end up being overloaded. Alternative solutions are presented in the following sections.

5.2 A Genetic Approach for Virtual Function Embedding

The second mechanism was conceived to shrink the search space of possible solutions for the embedding problem. The idea was to find a near-optimal solution consuming less computational time and resources. With this in mind, we used a genetic approach using a multi-objective fitness function to improve the resilience of SCs using the availability and disjointedness of the nodes, as well as the tiered nature of the Cloud to IoT infrastructure.

Genetic Algorithms are a type of search algorithms where the search space is a group of potential solutions, and the search is performed using notions of natural evolution [Renner and Ekárt, 2003; Pham et al., 2020]. As new organisms in nature evolve to adapt to their environment, GAs *evolve* solutions to solve a given problem. Table 5.2 lists the nature terms used in GA and its correspondence with artificial evolution.

Table 5.2: Correspondence of Nature and GA terms.

Nature	**GA**
Individual	Solution
Population	Set of solutions
Generations	Iterations until reaching the final solution
Offspring	Set of solutions from a follow-up iteration
Fitness	Quality of a solution
Chromosome	Representation of a solution
Gene	Part of the representation of a solution
Crossover	Binary search operator
Mutation	Unary search operator
Reproduction	Reuse of solutions
Selection	Keeping good solutions

A GA generates a set of solutions called *population*, out of this population, the *individuals* are ranked according to the optimization goal for the problem using a *fitness* function. New solutions are created by combining two parents by *crossover* and *mutation*. The fittest individuals will be passed to the next *generation*, the *offspring*, while weaker individuals die off. Individuals are represented as *chromosomes* composed by *genes*. For the sake of clarity, the same variables used for the ILP model listed in Table 5.1 are used for the GA.

A solution or *individual* is represented by the placement matrix P. The solution $P^{r,v}_{s,n} = 1$ indicates that instance $r \in R$ from VF $v \in V$ in SC $s \in S$ is embedded in node $n \in N$, and $P^{r,v}_{s,n} = 0$ otherwise. To generate the *offspring population*, the *crossover* binary operator combines two *individuals* (i.e., parents) to generate two new individuals (i.e., the children). Figure 5.1 shows an example of the crossover. A portion of the SC from the first solution is mixed with a portion of the SCs from the second solution using a crossover point which is chosen randomly, as depicted in Figure 5.1. After generating the new individuals via the *crossover*, a *mutation* is randomly introduced in some of the new individuals. This means, in the placement matrix P, some of the nodes $n \in N$ are changed for the solution.

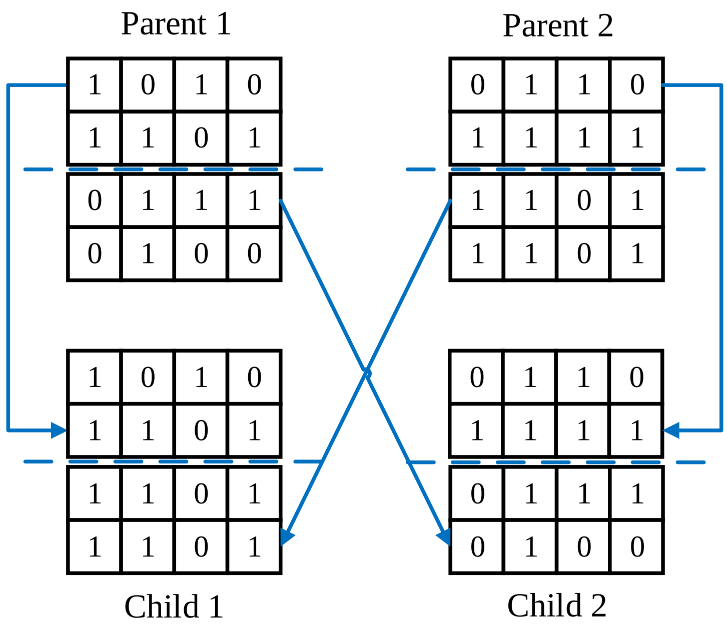


Figure 5.1: Representation of the Crossover.

In a GA the individuals should be ranked according to their quality using a *fitness function*. One possible function to compute the fitness of individuals is the WSGA; this approach allows to combined different objectives in a single fitness function [Mehboob et al., 2016; Guerrero et al., 2019]. Equation (5.11) shows the calculation for the fitness function, where ω_i is the scaling factor, θ_i is the weight, and X_i is the value of the objective function.

$$\sum_{i \in numObj} \omega_i \times \theta_i \times X_i \tag{5.11}$$

Algorithm 5.1 shows the implementation of the WSGA used in this work. The first step is to randomly generate the first population with a given size (*pop_size*) of individuals, as shown in line 1. For this population, the objective values (line 2) and the fitness function (line 3) of each solution are calculated using the WSGA approach before move forward to the evolution of the first population.

For a given number of generations (line 4) the offspring population is initialized as the empty set (line 5). From the individuals in the current population (line 6) two parents are randomly selected using a binary tournament (see lines 7 and 8). The two parents reproduce using the crossover operator, generating two children (line 9). The crossover mixes the two parents combining a portion from the first parent with a portion of the second. The resulting children are then mutated with a uniformly random probability of 25% (see line 10); this means that some of the nodes (i.e., with cardinality *num_genes*) marked in the solution for embedding are changed. The number of nodes to alter on each solution correspond with 10% of the VFs to embed in the entire solution, including

Algorithm 5.1: Weighted Sum Genetic Algorithm.

```
   Result: Embedding of SCs and their VFs
1  $P_t$ ← generate_random_population(pop_size)
2  obj_values ← get_obj_values(Pt)
3  fitness ← ws(obj_values, $\omega$, $\theta$)
4  foreach i in generations do
5      $\mathrm{P_{off}}$ ← $\emptyset$
6      foreach j in pop_size do
7          parent1 ← select_parent($\mathrm{P_t}$, fitness)
8          parent2 ← select_parent($\mathrm{P_t}$, fitness)
9          child1,child2 ← crossover(parent1, parent2)
10         if random() $\leq$ mutation_prob then
11             mutate(child1, child2, num_genes)
12         end
13         $\mathrm{P_{off}}$ ← $\mathrm{P_{off}} \cup$ {child1, child2}
14     end
15     obj_values ← get_obj_values($\mathrm{P_{off}}$)
16     fitness_off ← ws(obj_values, $\omega$, $\theta$)
17     fitness ← fitness $\cup$ fitness_off
18     $\mathrm{P_{off}}$ ← $\mathrm{P_{off}} \cup \mathrm{P_t}$
19     $\mathrm{P_{off}}$ ← order($\mathrm{P_{off}}$, fitness)
20     $\mathrm{P_t}$ ← $\mathrm{P_{off}}$[1...pop_size]
21 end
22 solution ← max ($\mathrm{P_t}$, fitness)
23 return solution
```

replicas (see line 11). The new mutated children are joined to the offspring population in line 13. The objective function (see Subsection 5.2.1) and fitness values (using Equation (5.11)) are recalculated for all new individuals in the offspring population in lines 15 and 16 respectively.

All the newly calculated fitness values are combined in line 17, and the new offspring population is combined with the current population in line 18. Individuals are sorted according to their fitness values in line 19 and only a given number of individuals (i.e., pop_size) with better (i.e., higher) fitness value will *survive* and be passed on to the next generation. At the end of the evolutionary process, when all generations were processed, the individual with the highest fitness value (line 22), i.e., the best solution, is returned in line 23. The computation of the objective functions for the fitness used in this work is described in the following subsection.

5.2.1 Combining Node Availability, Disjointedness, and Tiered Infrastructure in the Fitness Function

Different objective functions are combined in the WSGA and are defined in this section, the notation and variables from Table 5.1 are used. The first objective function is related to maximizing the availability of the SCs, to achieve this the availability vector F is used. The availability of node $n \in N$ that holds instance $r \in R$ of the VF $v \in V$ that belongs to the SC $s \in S$ is added up in Equation (5.12). Since the fitness function maximizes the results from the objective functions, higher availability values are prioritized.

$$\sum_{s \in S} \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} F_n \tag{5.12}$$

The next objective function encourages the embedding of different instances of the same VF $v \in V$ in different nodes $n \in N$. Equation (5.13) shows the calculation. If the instance $r \in R$ from the VF $v \in V$ is embedded in a different node for the same SC $s \in S$, the availability factor of node (i.e., $n2 \in N$) where the second instance (i.e., replica) of the VF is embedded is added, on the other hand, if different instances of the same VF are stored in the same node, zero (0) is added.

$$disjointedness = \begin{cases} \sum_{s \in S} \sum_{v \in V} \sum_{r1,r2 \in R} \sum_{n1,n2 \in N} F_{n2} & \text{if r1}\neq\text{r2; n1}\neq\text{n2} \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

Upper tiers of the network infrastructure have nodes with higher availability than those in the lower tiers. Thus, nodes closer to the edge have a lower availability factor than nodes closer to the core. This reflects the fact that nodes in the IoT have a higher probability of failure while the nodes in the Cloud are less prone to failure, and an availability uptime of 99.999% of the times is required, following the *five nines* principle [Bauer and Adams, 2012]. For this reason, the previous objective functions will lead the embedding towards the upper tiers, potentially affecting the response times of the SCs. The final objective function seeks to mitigate this side effect by prioritizing embedding in the lower tiers. Equation (5.14) reflects this objective function. Tier vector T has the following values for the different tiers: Cloud (0.6), Fog (0.8), Mist (0.9), and IoT (1). On the other hand, Ψ denotes a constant that allows normalizing each value between 0 and 100. Thus, higher values would lead the embedding to lower tiers, improving response times.

$$\sum_{s \in S} \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} \Psi \times T_n \tag{5.14}$$

The combination of the different objective functions via Equation (5.11) will lead to a spread embedding throughout the different tiers of the network infrastructure, in a vertical search space.

Unfeasible solutions where the embedding of the VFs $v \in V$ surpasses the physical capacities of the node $n \in N$ are assigned a fitness value of -1, so that they are discarded for the following generation, since the algorithm seeks to maximize the fitness value of the solutions.

Genetic Algorithms have successfully been used before for embedding solutions [Pham et al., 2020; Carpio et al., 2017; Ma et al., 2017; Chen et al., 2020]. Among different genetic approaches, WSGA has proven to have short convergence times to find optimal solutions in placement problems [Guerrero et al., 2019].

According to the conditions of the scenario (e.g., size of the topology, traffic load) the number of generations needed to reach a near-optimal solution increases, impacting the computation resource and execution time needed. An alternative heuristic for more complex scenarios is presented in the following section.

5.3 A Fluid Community Heuristic for Virtual Function Embedding

For online and more complex scenarios, optimal or near-optimal solutions usually are not an option given the number of resources and time that has to be invested in finding it. Additionally, after the design and implementation of the mechanism described in Section 5.1, it was noticed that some nodes were saturated. On the other hand, for the GA described in Section 5.2, sometimes the VFs were too spread in the infrastructure, impacting the response time of the SCs. To overcome these issues, an additional mechanism able to address the embedding problem in a practical way, leading to satisfactory results, was designed and proposed.

Fluid Communities by Tiers is a heuristic-based mechanism to enhance the resilience of SCs based on graph partition. Graph theory is frequently used to tackle problems in a vast range of engineering and computer science applications. A graph usually allows representing almost any physical situation involving discrete objects and a relationship between them (e.g., a communication network infrastructure) [Deo, 2017]. A relevant feature of performing graph representation is the partition or community structure that allows the study of edges and vertices that belong to a cluster with common interests and/or metrics [Bichot and Siarry, 2011].

An approach that has proven to be helpful to deal with the saturation issues found with the ILP model is building communities in the communication network topology to achieve load balancing [Velasquez et al., 2020; Skarlat et al., 2017a; Lera et al., 2019a]. Furthermore, exploring the entire topology, taking advantage of its multi-tier nature, could help overcome the issue detected with the genetic approach.

Lera et al. [Lera et al., 2019a] proposed a service placement policy focused on enhancing the availability and QoS of applications using graph partition. The community detection used for the placement mechanisms was the method proposed by Newman and Girvan [Newman and Girvan, 2004], which progressively removes edges from the original graph to identify the communities on it. The method removes the most valuable edge, usually the edge with the highest betweenness centrality, at each step. Thus, the graph breaks down into pieces, and the tightly knit community structure is exposed. This research adopted a different approach to build more reliable and balanced communities using the Fluid Communities (FluidC) method.

5.3.1 Building Fluid Communities

The Fluid Communities approach is based on a Community Detection (CD) algorithm that could be applied to any graph to create groups called *communities* as sets of vertices densely interconnected that at the same time are sparsely connected with the rest of the graph [Parés et al., 2018]. Particularly, the FluidC algorithm creates the communities by mimicking the behavior of several fluids, expanding and pushing one another in the graph until an equilibrium is found. The algorithm receives an input parameter K, which indicates the number of fluids, i.e., communities, that are going to be created.

Given a graph $G = (N, L)$ with N the set of vertices and L the set of edges, the algorithm will initialize K fluid communities $C = \{c_1, \dots, c_K\}$ with $0 < K \leq |N|$. Each community $c \in C$ is initialized with a distinct and randomly selected vertex in N. The density d of a community $c \in C$ is defined according to (5.15), as the inverse of the number of nodes in the community c.

$$d(c) = \frac{1}{|c|} \tag{5.15}$$

For a vertex n, the update rule returns the community or communities with maximum aggregated density d within the ego network [1] of n. This rule is formally defined by (5.16) and (5.17), where n is the vertex being updated, $C_n^{'}$ is the set of candidate communities, each of which could be the new community for n, $B(n)$ are the neighbors of n, $d(c)$ is the density of community c, $c(z)$ is the community to which vertex z belongs to, and $\delta(c(z), c)$ is the Kronecker delta.

$$C_n^{'} = \arg\max_{c \in C} \sum_{z \in \{n\} \cup B(n)} d(c) \times \delta(c(z), c) \tag{5.16}$$

[1]Ego networks consist of a main node ("ego") and the nodes directly connected to it.

$$\delta(c(z), c) = \begin{cases} 0 & \text{if } c(z) \neq c \\ 1 & \text{otherwise} \end{cases} \qquad (5.17)$$

Figure 5.2, adapted from Parés et al. [Parés et al., 2018], describes the process of building the communities. Assume it selects the red node and the green node shown in the topology in Figure 5.2. After the first iteration, the density of both communities is 1, the maximum possible value for density. The node highlighted in blue represents the vertex where the update rule will be evaluated. For the following iterations, the algorithm will randomly select a node from the neighbors of the communities, which will be assigned to the community with the highest density value. In the case that there are several communities with the same density value, it will randomly select one of those communities. For the example depicted in Figure 5.2 the FluidC converges after one complete superstep [2]. The last stage depicted in Figure 5.2 shows the final two communities in a tiered environment with two tiers. The tiered communities approach is used for the embedding mechanism described in the next subsection.

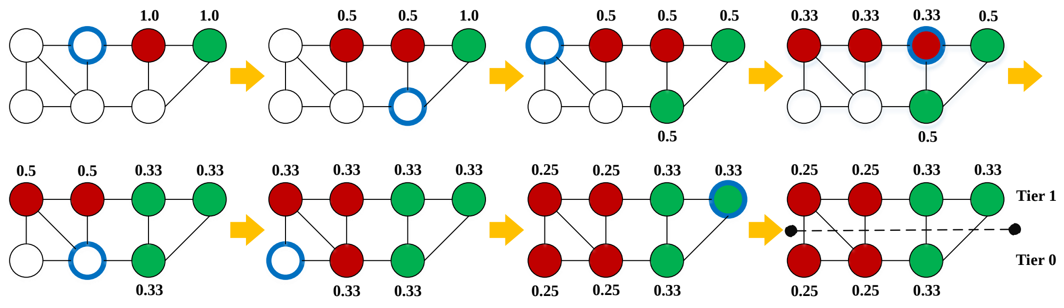


Figure 5.2: Workflow of FluidC for $K = 2$ communities and $tiers = 2$ (Adapted from [Parés et al., 2018]).

The FluidC algorithm allows the definition of the desired amount of communities and also that there will not be a *monster* community [3], in comparison with the rest of communities in the graph. An adaptation from the original Fluid Communitiess algorithm is used in this work to partition the topology graph. Algorithm 5.2 describes the process to build the communities.

The algorithm begins by collecting the network topology information, in line 1, to organize the nodes by communities. The Cloud node is an independent community, handled differently considering the model assumption described in Chapter 4 (i.e., infinite resources and absence of failures), which is why it is not taken into consideration for this process (line 2).

The highest modularity value for the topology given is used to specify the value of k (see line 3), required as an input value for the FluidC algorithm, following the approach used by Pares et al. [Parés et al., 2018]. The modularity is a metric that

[2]An iteration over all the vertices of the graph.

[3]A community significantly larger than the rest.

Algorithm 5.2: Build Communities.

```
Result: $k$ communities for a given network infrastructure
1 topology ← get_topology()
2 remove_node(topology, Cloud_node)
3 $k$ ← highest_modularity(topology)
4 for $i$ = 1 to $k$ do
5     communities[$i$] ← FluidC(topology,$k$)
6 end
7 fcommunities ← highest_performance(communities)
8 return fcommunities
```

measures the strength of the division of a graph, often used in optimization to detect community structure in a network. As the modularity grows, the groups inside the topology have denser connections between the nodes inside the groups and sparser with the nodes in other groups, enabling a resilient communication between the nodes inside a community or group [Fortunato, 2010].

Considering that the FluidC algorithm evaluates the update rule using a random approach, the results of invoking this algorithm are not deterministic. Thus, the algorithm is invoked k times (lines 4 to 6) before choosing the final community set that will be used for the embedding process.

The resulting communities set will be the one with the highest performance, in line 7, where the performance measures the ratio of the number of intra-community edges plus inter-community non-edges with the total number of potential edges [Bichot and Siarry, 2011]. Line 8 returns the final set of communities.

The communities created are balanced in the sense that they all have a similar amount of nodes (i.e., the standard deviation of the average community size is smaller than other community-building processes), which is a desired characteristic for an embedding process that takes into account resilience and load balancing. The characteristics of the communities created are discussed in Chapter 6.

5.3.2 Embedding Service Chains in the Fluid Communities

The Service Chains have to be embedded in the computing and communication infrastructure considering the set of communities selected after invoking Algorithm 5.2. This corresponds to the second challenge described in Section 4.1. Each *sc* will be embedded in a single community, respecting the resource constraints of the nodes and avoiding placing replicas of the same *vf* in a single node (except in the case of embedding in the Cloud node). The nodes in the community are also sorted according to the tier to which they belong (i.e., Fog, Mist, IoT); the Cloud node has its own community (see the last stage depicted on Figure 5.2).

The embedding process, named Fluid Communities by Tiers (FCT), and depicted in Algorithm 5.3, begins by getting the information about the topology and its tiered hierarchical structure, as seen in lines 1 and 2, respectively. This information is provided via a catalog of the network updated continuously by the MANO module of the Cloud to IoT infrastructure manager. With this information, the communities are built by invoking Algorithm 5.2 in line 3. Line 4 obtains the values that correspond to the *SCs* and their constituent *VFs*, such as response time, resource consumption, hardware and software specific requirements. The tiered communities *tcomm*, initialized in line 5, contains the list of nodes that belong to the communities sorted by tiers.

The actual embedding process takes place between lines 6 and line 34, for each *sc* requested and their *VFs*. The first step of the embedding is to initialize the community pointer, *commp*, denoting the community where the *vf* is to be embedded (line 7). For each *vf*, a flag variable (*vf_deployed*) controls if the *vf* was successfully embedded in a substrate node of a given community. The tiered communities are sorted by their amount of free computational resources, from highest to lowest, in line 10. The tiered communities ordered, *tselect*, is an array of three elements that represents the nodes inside the community that belong to the Fog, Mist, and IoT.

From line 11 to line 15, the algorithm verifies if the given *vf* can be hosted by the node with the highest available resources in the previously selected tier (line 11). If the embedding process is successful (line 12), which means that the *vf* instance was deployed in a disjoint node from its replicas, the flag variable *vf_deployed* is updated to *True* in line 13. If the embedding process was not successful, two new attempts are launched for the following two tiers with more available resources, as seen in the blocks from line 16 to 22 and from line 23 to 29. If all previous attempts fail, the embedding will take place in the Cloud, as depicted in lines 30 and 31. The result from the embedding process is returned in line 35.

In summary, the embedding process deploys a *vf* and their replicas in disjoint nodes, prioritizing the tiers with more available resources in order to balance the load between the Fog, Mist, and IoT, and just considering the Cloud if all previous attempts failed. Thus, the mechanism also promotes the embedding of the VF in nodes with high availability but also closer to the end-users in order to enhance the response time of services and applications.

5.4 Summary

To cope with the embedding challenge of SCs in the Cloud to IoT continuum achieving a high level of resilience, three mechanisms for the mapping of VFs and their replicas in the substrate service provider infrastructure are proposed in this chapter; one based on ILP, one based on GAs, and a heuristic based on graph partition. The bi-level ILP model aims to maximize the acceptance ratio of SCs embedding them in the nodes with higher availability; the GA solution uses a fitness function that mixes the node availability with placing replicas in

Algorithm 5.3: Fluid Communities by Tiers - FCT.

```
Result: Embedding of SCs and their VFs
topology ← get_topology()
ttiers ← get_tiers(topology)
communities ← build_communities()
SCs, VFs ← get_service_chains()
tcomm ← sort_communities(communities,ttiers)
foreach sc in SCs do
    commp ← commp  mod size(communities)
    foreach vf in VFs do
        vf_deployed ← False
        tselect ← highest_resource(tcomm,commp)
        if get_resource(vf) ≤ max(tselect[0]) then
            if embedded(vf,tselect[0]) then
                vf_deployed ← True
            end
        end
        if vf_deployed = False then
            if get_resource(vf) ≤ max(tselect[1]) then
                if embedded(vf,tselect[1]) then
                    vf_deployed ← True
                end
            end
        end
        if vf_deployed = False then
            if get_resource(vf) ≤ max(tselect[2]) then
                if embedded(vf,tselect[2]) then
                    vf_deployed ← True
                end
            end
        end
        if vf_deployed = False then
            embedded(vf,Cloud)
        end
    end
end
return final_embedding
```

disjoint nodes, and the tier to which the selected node belongs to; finally, the heuristic called FCT creates communities to perform a vertical search of the node to embed the VFs, trying to bring them closer to the edge of the network thus improving the response time of the SCs.

The three mechanisms described were sequentially developed and evaluated via simulation. In order to avoid repetition and to improve their understanding, these mechanisms were discussed in this chapter, and their evaluation is presented separately in the next chapter.

The contributions from this chapter are the three embedding mechanisms that use replication of VF as a means to maximize the availability of the Service Chains. In detail, these contributions are the following:

- A formal mathematical model, based on ILP, for the embedding of VFs. The model aims at maximizing the acceptance rate of the SCs while also maximizing their survivability by selecting the nodes with higher availability for the embedding;
- A genetic algorithm for VFs embedding that uses three objective functions, namely maximizing the availability of the embedding nodes, using disjoint nodes for the embedding of the replicas, and distributing the VFs throughout the Cloud-Fog-Mist-IoT landscape different tiers; and
- A heuristic based in the Fluid Communitiess approach for graph partitions. The VFs are to be embedded by communities performing a vertical search in the Cloud to IoT continuum to find the most suitable node considering its availability, to enhance response times for the SCs.

To validate the mechanisms introduced in this chapter, a simulation approach was selected since it has proven to be a standard solution to evaluate mechanisms for the complex environment offered by the Cloud to IoT continuum, as seen in the evaluation of the State of the Art in Chapter 2. To move forward to the assessment, the first step is identifying the proper simulation tool that allows the design and implementation of the experiments. The next chapter provides a conceptual and technical analysis of simulation tools, the description of the experimental setup used, and the experiments designed to validate the mechanisms, as well as the discussion of the results obtained.

Chapter 6

Assessing the Service Chain Embedding Mechanisms

Contents

To model the Cloud to IoT continuum, evaluation tools such as simulators must include some characteristics that would lead to more realistic results in such a complex environment. For instance, it must be possible to model features such as location awareness and low-latency, mobility support, as well as interoperability and scalability mechanisms. A proper Cloud to IoT simulator should include support for these features. An analysis of six different Cloud/Fog simulators was conducted to ultimately select the simulator used for the validation of the embedding mechanisms.

In this chapter, the experimental evaluation characteristics are also detailed, including descriptions of the load, replication methods, and topology used. Finally, results from the experiments are presented and analyzed. The experiments assess the embedding mechanisms on their failure ratio, node utilization, and response time of the Service Chains.

6.1 An Analysis of Cloud to IoT Simulation Tools

Simulation has proven to be a standard tool for early validation before testing solutions for complex scenarios in real testbeds (see Chapter 2). However, selecting the appropriate simulation tool can be complex in itself. This section presents a conceptual review on six Cloud/Fog Simulation tools (i.e., iFogSim, CloudSimSDN, Yet Another Fog Simulator (YAFS), EmuFog, FogTorchπ, and EdgeCloudSim), describing their main characteristics and what they allow to experiment. The final output of this section will lead to the selection of the simulation tool to use for the validation of the embedding mechanisms proposed in Chapter 5.

The selection of the simulation tool was based on the fulfillment of the Cloud-Fog-Mist-IoT features previously mentioned (see Chapter 2), as well as regarding their acceptance (measured by the citation number) and the support provided by the community (measured by the availability of tutorials, documentation, examples, and/or discussion groups). The acceptance and community support help in the selection by allowing to discard tools that have not been adopted by the community.

The simulators selected are compared among each other regarding their features. The descriptions presented in this section come from the information provided by their authors in the respective papers, but also from experience working with them. Table 6.1 summarizes the non-technical features used in the comparison, and Table 6.2 comprises the technical features.

In Table 6.1, *Latest release* refers to the year in which the latest release of the tool was published; *First release* corresponds to the year of the launch of the first version of the tool (as listed on its GitHub repository), in order to have an idea on its maturity; *Date of publication* indicates the year on which the paper that introduces the tool was published; *Community support* measures the backing of the creators of the tool and the research community in terms of availability of examples, tutorials/documentation, and/or existence of community groups. The *Community support* is measured in High (all three factors are present), Moderate (two factors present), and Low (only one-factor present); and *Citations* accounts for the number of hits obtained in Google Scholar when searching the name of each tool. *Citations* and *Latest Release* are accounted as to July 2019, when the research was performed.

Table 6.1: Non-technical Comparison of Cloud/Fog Simulators.

Characteristics	**iFogSim**	**CloudSimSDN**	**YAFS**	**EmuFog**	**FogTorchπ**	**EdgeCloudsim**
Latest Release	2017	2018	2019	2018	2017	2018
First Release	2016	2015	2018	2017	2016	2017
Date of publication	2017	2015	2019	2017	2017	2017
Community support	Low	Moderate	Moderate	Low	Low	Moderate
Citations	334	65	34	20	29	56

From Table 6.1, it is noticeable that CloudSimSDN, YAFS, EmuFog, and Edge-CloudSim are the most recently updated projects, while FogTorchπ and iFogSim are not so far behind. The most mature tools (according to the Release Date and Date of Publication) are iFogSim, CloudSimSDN, and FogTorchπ; with YAFS being the most recent. This is also reflected in the *Citation* category, where EmuFog and YAFS have significantly fewer citations; this could be highly influenced by the lack of maturity of the tools and their recent creation.

Regarding the Community support, there is no official discussion group for any of the simulators. However, there are a couple of groups related to CloudSim [ResearchGate, 2019; Groups, 2019], which is the base simulator from where iFogSim, CloudSimSDN, and EdgeCloudSim build up, and some threads in these groups are dedicated to these simulators. Respecting the examples, almost all the simulators include them in their source code, except for EmuFog. Finally, about the tutorials or documentation, iFogSim and FogTorchπ do not offer any documentation; for iFogSim just a couple of examples are provided with the source code. On the other hand, CloudSimSDN, YAFS, EmuFog, and Edge-CloudSim include in their repositories a guide for installation and examples on how to use them; furthermore, YAFS also has a more detailed document covering the basic concepts related to the simulator [Lera and Guerrero, 2019].

As for the number of hits in Google Scholar, iFogSim shows the highest numbers (in the hundreds), which leads to thinking that it is the most used tool (more referenced). This could be because iFogSim has been available for longer (more mature), which will also explain the lower numbers of YAFS and EmuFog. On the other hand, FogTorchπ is at the bottom of the citations category, while being one of the most mature tools being under evaluation, which might lead to think it is not too popular among researchers.

The next evaluation is focused on the technical features. Since all evaluated tools include support for Cloud/Fog environments, and all use Discrete-Event Simulation (DES) (except EmuFog, which uses emulation), these two features are not included in the table. In Table 6.2, the row *Language* means the programming language used to code the experiments; the *Topologies* category refers to the types of network topologies that are supported by the tool; the presence of the feature *Fault Injection* indicates if the tool is able to simulate random failures in the topology or if dynamic topologies are supported; *Application Model* refers to the representation of an application in the context of the tool; *Mobility* indicates whether the simulator has support for this feature; the category *Cost and Energy Model* indicates if there is already a model (for cost and/or energy) implemented or the possibility for it to be added by the user via built-in features; and the row named *Federation and Scalability* shows if there is support to define federations and scale upwards or downwards the resources used by the Virtual Machines (VMs) or building clusters (i.e., grouping or ungrouping computational nodes) in the Cloud/Fog environment.

Table 6.2: Technical Comparison of Cloud/Fog Simulators.

Characteristics	**iFogSim**	**CloudSimSDN**	**YAFS**	**EmuFog**	**FogTorchπ**	**EdgeCloudSim**
Language	Java	Java	Python	Java	Java	Java
Topologies	Tree	Arbitrary	Arbitrary	Arbitrary	Arbitrary	Arbitrary
Fault Injection	No	No	Yes	Yes	No	No
Application Model	Modular	Service-chain	Modular	Docker-based	Modular	Modular
Mobility	No	No	Yes	No	No	Yes
Cost and Energy Model	Yes	Yes	Yes	Partial	Yes	Partial
Federation and Scalability	Yes	Yes	Yes	Partial	No	Partial

From Table 6.2 it is noteworthy that the most used programming language is Java, but newer tools (i.e., YAFS) use Python. About the topologies, the most useful is to have support for an arbitrary design, which will enable to recreate different simulation scenarios; iFogSim has a disadvantage by only allowing tree topologies. With iFogSim, the communication is only possible within the same branch of the tree, thus if a user wants to try, for instance, a customized placement policy, it would not be possible with iFogSim, since by placing application modules in different branches, the communication will not be carried out correctly (i.e., will not be included in the simulation's results).

Other essential features are the support for fault injection as well as adding or removing nodes and links arbitrarily. These features will allow to portrait more complex experiments and test out a wider variety of mechanisms to handle resilience and fault tolerance in more realistic environments. From the selected tools, only YAFS and EmuFog offer fault injection support.

Regarding application modeling, different approaches are taken. A common way that is close to real environments is using a modular approach where the application is defined as a set of modules (or microservices) that constitute the whole. This method is used by iFogSim, YAFS, FogTorchπ, and EdgeCloudSim. On the other hand, CloudSimSDN uses a service-chain model for applications. CloudSimSDN is more oriented to infrastructure, while the other tools are oriented to applications. Finally, EmuFog allows using real-life Docker-based applications in their emulation environment.

Mobility support is a key feature requirement for Cloud/Fog applications and services, considering they are usually attached to users that are moving between different access points at the Edge of the communication infrastructure. Out of the six simulators analyzed, only two offer native mobility support, YAFS and

EdgeCloudSim. A fork from iFogSim, called MyiFogSim, supports mobility and VM migration. This is one of the aspects with more room for improvement in the Cloud/Fog simulation field since the support is not only scarce, but it is also basic.

The cost of deploying services and applications is a critical feature for end-users and stakeholders. For this reason, the possibility to define a model or behavior regarding monetary or energy consumption during the simulation process has been incorporated into a variety of simulators. From the simulators under study, Emufog and EdgeCloudSim are the only ones with partial support of this characteristic. Particularly, in EmuFog only experiments considering the monetary cost have been carried out; on the other hand, EdgeCloudSim authors' mentioned the support of energy consumption models for mobile and Edge devices, as well as the Cloud datacenters, as a needed feature for the simulator.

Fog environments are dynamic by nature. This feature requires adaptive functions to enable asking for more resources or release them on-demand, as well as to deal with variations on the service infrastructure (e.g., data bursts, communication failures). Simulators as iFogSim, CloudSimSDN, and YAFS support the dynamism and on-demand requirements of Fog services/applications via VM elasticity and migration, federation polices, and clustering of computational nodes. Emufog has scalability support regarding the communication and topology infrastructure; nevertheless, it lacks of strategies to deal with on-demand requirements of services/applications inside computational nodes. Finally, EdgeCloudSim only supports Federation and Scalability between nodes of the same tier (only Cloud or only Fog), which means that it is not possible to achieve a proper orchestration along the Cloud to Fog continuum.

The reports obtained are also a technical feature under analysis. Table 6.3 lists the metrics reported by each simulator. The presence of a checkmark (✓) indicates that the simulator reports that metric, while a dash (–) says otherwise. It is noticeable that iFogSim, CloudSimSDN, and YAFS offer a more detailed report of the simulation, while EmuFog has the poorest. The metrics that are more often reported are those related to resource consumption (i.e., CPU, memory, bandwidth, and energy), while the metrics with the least support are those related to fault tolerance (i.e., failed tasks, waiting time, availability).

Table 6.3: Metrics reported by the Cloud/Fog Simulators.

Metrics	**iFogSim**	**CloudSimSDN**	**YAFS**	**EmuFog**	**FogTorchπ**	**EdgeCloudsim**
CPU consumption	✓	✓	✓	✓	✓	✓
Memory consumption	✓	✓	✓	✓	✓	✓
Bandwidth consumption	✓	✓	✓	✓	✓	–
Energy consumption	✓	✓	✓	–	–	–
Deployment cost	✓	✓	✓	–	✓	–
Latency	✓	–	✓	–	✓	✓
Execution time	✓	✓	–	–	–	–
CPU time	✓	✓	✓	–	–	–
Network time	✓	✓	✓	–	–	–
Failed tasks	✓	✓	–	–	–	✓
Waiting time	–	–	✓	–	–	–
Link availability	–	✓	–	–	–	–
Node availability	–	–	✓	–	–	–

With these findings in mind, it was time to move forward with the selection of the simulation tool. Both technical and non-technical features are considered in this process.

Based on the previous discussion and the outcomes of Tables 6.1, 6.2, and 6.3, three tools stand out: iFogSim, a Fog simulation tool with strong acceptance of the community (measured by its citations); CloudSimSDN, which not only includes the Fog features but also has a strong citation number; and YAFS, which although being relatively recent (low citation number), it includes the Fog characteristics and has a detailed documentation and metrics report.

FogTorchπ was discarded because of its low popularity (low number of citations while being available for a long time), and also because it is considered as a prototype tool by its authors [Brogi et al., 2017]. EdgeCloudSim lacks some critical requirements in the Cloud to IoT continuum, such as, execution of tasks on Fog/Mist devices and task migration between Cloud and Fog/Mist tiers; additionally, EdgeCloudSim has limited support of application performance metrics, which restricts the amount of information that can be gathered from experiments performed with this tool to resource consumption (e.g., CPU, memory), latency, and failed tasks. This issue drastically narrows its scope for experimental work. EmuFog was discarded since it is an emulation tool and not for simulation, and also for its even more limited metrics support.

CloudSimSDN is more suited for experiments aimed at evaluating the network infrastructure; particularly, the possibility to define the physical topology, the VN, and the workloads separately, making it useful to perform experiments related to SDN environments. iFogSim and YAFS are more inclined to experiments designed to assess the performance of an application in Cloud to IoT environments; however, iFogSim has reportedly shown starvation issues for complex and high resource demanding use cases [Perez Abreu et al., 2020]. iFogSim has better support regarding the simulation of algorithms and mechanisms that require measurements of resource consumption inside VMs; on the other hand, YAFS focuses on appraising the impact of load and placement of application modules in a communication infrastructure, which is enhanced by the possibility of including complex network theory and dynamic topologies in the simulations.

YAFS [Lera et al., 2019b] was ultimately selected as the simulation tool because of its strong support for Fog critical features and its capacity to introduce failures during the simulation [Perez Abreu et al., 2020], which allows evaluating the survivability of the Service Chains. The next section describes the characteristics of the experiments performed to validate the embedding mechanisms discussed in Chapter 5.

6.2 Describing the Evaluation Setup

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. The IBM CPLEX Optimizer version 12.9 [IBM, 2019] was used for the ILP model, and Python 2.7.16 was used for YAFS.

Figure 6.1 summarizes the evaluation procedure. Requests are generated automatically and randomly. The requested chains are evaluated according to the grammar presented in Chapter 4 to create the corresponding graphs that result as valid alternative chains that satisfy the user requirement.

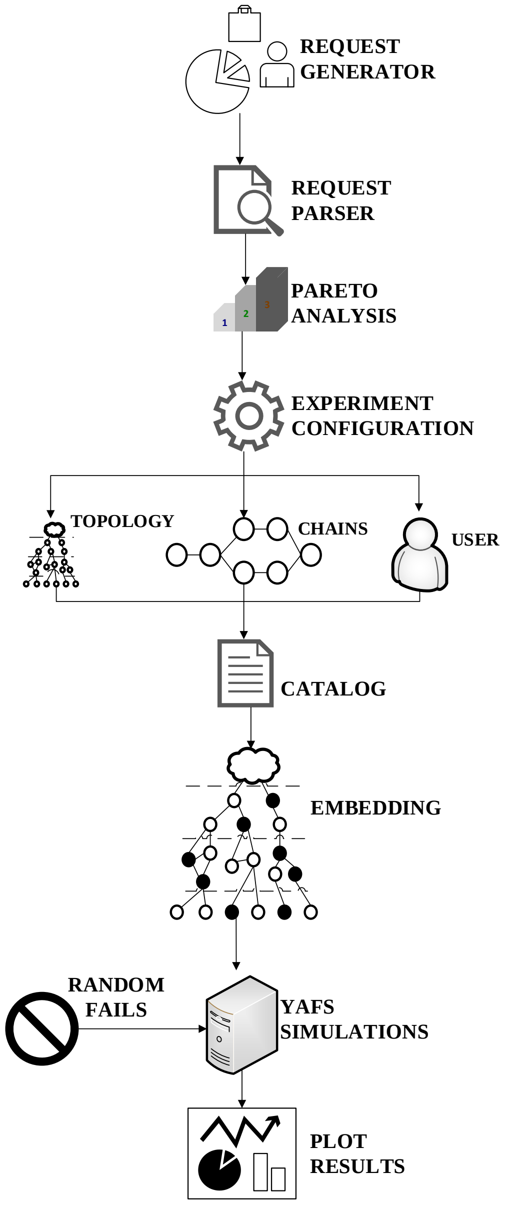


Figure 6.1: Experiment Setup Workflow.

Out of the different alternatives, the best ones are chosen (i.e., based on data rate, number of instances, and resources) using a Pareto analysis. The selected Service Chains, the topology and the information about the users (i.e., requests) are combined in the catalog that is used by the embedding mechanisms. Once the embedding process is carried out, the simulation is performed. During the simulation, random failures are introduced to measure the behavior of the embedding mechanisms under failing conditions. The plots resulting from the analysis of the raw results reported by YAFS are presented in the following section.

For this work, two partition graph methods (i.e., Fluid Communities and Newman-Girvan) were initially considered for the heuristic proposed for embedding SCs. The main idea was selecting the graph partition method that provides the best results regarding load balancing and resilience for the communities building process. Both partition methods were evaluated over different synthetic random network topologies (i.e., Barabasi-Albert, Complete, Lobster, PowerLaw, Star, and Tree), each with 50 nodes. The validation process consisted of a batch of experiments to compare metrics including performance [1] (higher values are better), communities length, average and standard deviation of the communities size; the number of communities for both methods was 5. The target was building communities similar in size so there were no tiny or monster communities as an outcome.

The results from this previous evaluation step are listed in Table 6.4 and a visual representation is displayed in Figure 6.2. Although the values are similar, FluidC shows a lower standard deviation for the communities length. It is also noticeable from the boxplots in Figure 6.2 that overall FluidC creates communities more balanced regarding their size. These observations led to the selection of FluidC for creating the communities. The final topology selection was the Barabasi-Albert to create a scale-free infrastructure. This decision was inspired by the fact of several natural and human-made system, such as the Internet, the World Wide Web, and Social Networks can be models using a scale-free network [Latora et al., 2017], besides this approach, it has been used before in similar studies [Velasquez et al., 2020; Lera et al., 2019a].

Table 6.4: Graph Partition Evaluation.

Method	**Topology**	**Perf**	**Comm. Size**	**Mean**	**SD**
FluidC	Tree	0.83	$[7, 13, 7, 13, 10]$	10.00	2.68
	Barabasi	0.82	$[10, 10, 10, 10, 10]$	10.00	0.00
	Lobster	0.83	$[7, 8, 12, 14, 9]$	10.00	2.60
	PowerLaw	0.82	$[11, 10, 11, 10, 8]$	10.00	1.09
	Star	0.18	$[1, 1, 47, 1, 1]$	10.20	18.40
	Complete	0.19	$[9, 8, 11, 14, 8]$	10.00	2.28
Girvan	Tree	0.83	$[11, 11, 9, 13, 6]$	10.00	2.36
	Barabasi	0.78	$[16, 12, 7, 13, 2]$	10.00	4.93
	Lobster	0.83	$[12, 13, 6, 7, 12]$	10.00	2.89
	PowerLaw	0.79	$[19, 6, 7, 11, 7]$	10.00	4.81
	Star	0.18	$[47, 1, 1, 1, 1]$	10.20	18.39
	Complete	0.84	$[46, 1, 1, 1, 1]$	10.00	18.00

After this analysis, a graph to model the substrate service provider infrastructure was generated following a random Barabasi-Albert method, according to the complex network theory [Jalili and Perc, 2017]. Forty-nine (49) nodes are spread between the IoT, Mist, and Fog, and an additional node (for a total of 50 nodes) represents the Cloud. The Cloud node is the node connected to the Fog nodes

[1] Ratio of the number of intra-community edges plus inter-community non-edges with the total number of potential edges.

with the highest betweenness centrality [2] in the graph; on the other hand, the nodes with the lower betweenness centrality correspond to the gateways in the IoT. The nodes directly connected to the gateways belong to the Mist, and the rest of the nodes constitute the Fog. In respect of the failures, the simulation time was set to 50000 time units, and there is a random failure set for each 2500 time unit. Thus, there are 20 failures during the simulation. This represents 40% of the nodes in the topology.

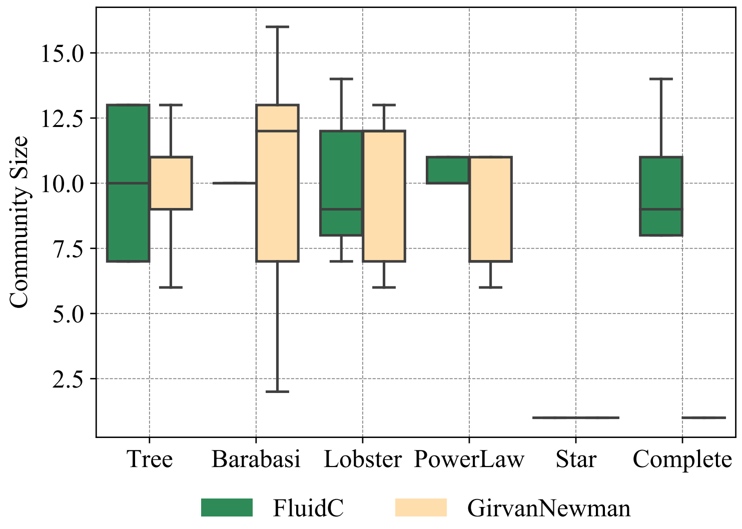


Figure 6.2: Nodes per Community for the FluidC and Girvan-Newman Methods.

The survivability factor for each node is assigned randomly according to some ranges that correspond to the layer to which the node belongs to. Thus, nodes closer to the edge have a lower survivability factor than nodes closer to the core. This reflects the fact that nodes in the IoT have a higher probability of failure while the nodes in the Cloud are less prone to failure, and an availability uptime of 99.999% of the times is required, following the *five nines* principle [Bauer and Adams, 2012].

Simulation parameters are listed in Table 6.5. The tier weight parameter guides the selection of the nodes in the embedding process so that the search space is explored vertically (see Tier vector in Section 5.1). YAFS' resource unit is used to specify VFs demands. This unit is defined as a vector that contains the capacity of different computational resources to be used by the VFs (e.g., number of cores for CPU, GB for memory, or TB for the hard disk). Regarding the node resources, the amount is randomly selected between 10 and 25 resource units for the Fog, Mist, and IoT tiers; however, this value is weighted so that the IoT nodes are closer to 10 resource units and the Fog nodes are closer to 25 resource units.

For the Service Chains, five different examples were used based on typical Smart Cities and IoT services. These chains are: (1) Web services, (2) Video streaming,

[2]Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

Table 6.5: Simulation Parameters.

Component	Parameter	Value
Cloud	Tier weight	0.6
	Availability (%)	99.999
	Resources (units)	∞
Fog	Tier weight	0.8
	Availability (%)	88 - 98
	Resources (units)	10 - 25
Mist	Tier weight	0.9
	Availability (%)	77 - 87
	Resources (units)	10 - 25
IoT	Tier weight	1
	Availability (%)	66 - 76
	Resources (units)	10 - 25
VFs	Requirements (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000
Failures	Time between failures (time units)	2500

(3) Voice-over-IP, (4) Online gaming, and (5) Generic IoT application. These chains can be used as the foundation of more complex Smart City-based applications, such as smart surveillance (which could be based on video streaming) or smart lighting (that can be based on a generic IoT for sensing and actuating chain). The Service Chains used, and their belonging Virtual Functions, are listed in Table 6.6. Similar base service chains were used in other works [ETSI GS NFV, 2013; Hmaity et al., 2017; Ahmed et al., 2019].

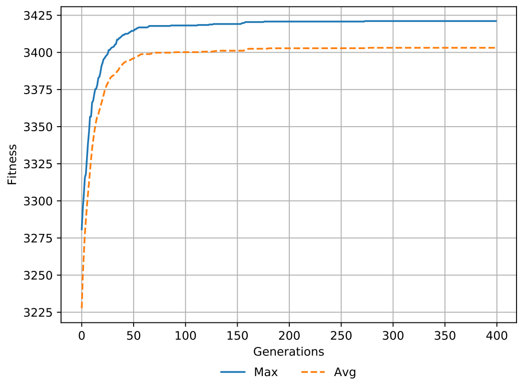
Table 6.6: Service Chains.

Service Chain	Chained VFs
Web services	NAT-FW-TM-WOC-IDPS
Video streaming	NAT-FW-TM-VOC-IDPS
Voice-over-IP	NAT-FW-TM-FW-NAT
Online gaming	NAT-FW-VOC-WOC-IDPS
Generic IoT application	GENSEN-DAGG-DACOM-DANA-DB

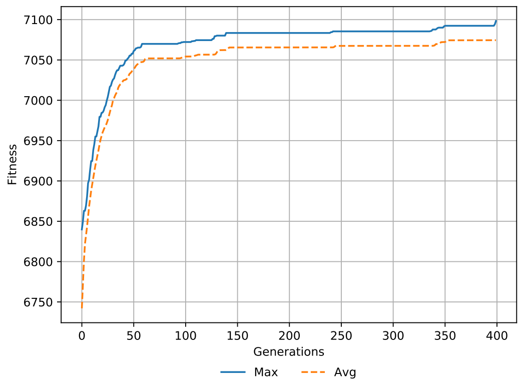
Three different redundancy models are used for the Service Chains [ETSI GS NFV-REL, 2016]: **End2End**, one replica for each VF in the chain; **vsReplicas**, some of the VFs in the chain have replicas; and **NoReplicas**, where no replica is used for any VF in the chain.

Four scenarios were defined by varying the network load with regards of the Service Chains: (1) **tiny**: 5 SCs, (2) **small**: 10 SCs, (3) **medium**: 15 SCs, and (4) **large**: 20 SCs. Similar loads were used in experimental cases before [Velasquez et al., 2020].

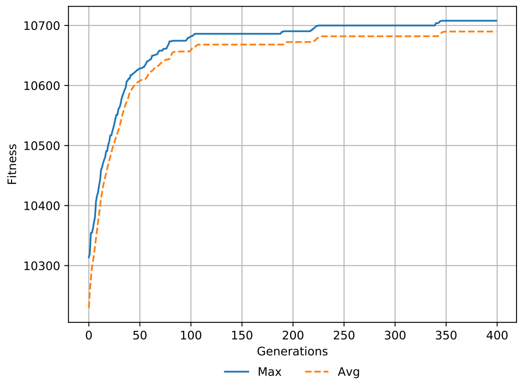
With respect of the parameters used for the GA, the weight used for each objective function was 0.33 since we considered the three objectives equally important. Regarding the size of the population and the number of generations, a study was made with the different workloads previously defined. Similar values were also used in previous work [Guerrero et al., 2019]. Figure 6.3 shows the evolution of the fitness value for the tiny, small, medium, and large scenarios when using the End2End replication method. It is noticeable that the fitness value improves as the generations augment, as expected, but the improvement is not as significant from generation 250, which indicates that the solution converged. The trend observed here is resemblant to the ones obtained for the other replication methods as depicted in Appendix A.



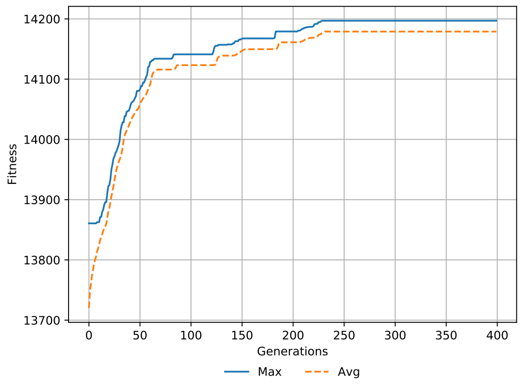
(a) Tiny Scenario.



(b) Small Scenario.



(c) Medium Scenario.



(d) Large Scenario.

Figure 6.3: Fitness Function Values by Generations for SCs using End2End Replication in the Scenarios Evaluated.

For comparison purposes, the ILP model, the GA approach, and the FCT heuristic are validated against the well-known First Fit (FF), as it is done in other works [Skarlat et al., 2017a,b; Velasquez et al., 2020]. All the scenario setup, as well as the source code, is available via a GitLab repository [Perez Abreu et al., 2020].

6.3 Results and Analysis

This section presents the results obtained from the simulations. The failures in the nodes were randomly generated before running the simulations to maintain the same scenario between repeated simulations. Correspondingly, the selection of the embedding locations is statically executed before the simulations; hence the VFs are placed in the same nodes during the different simulations. This way, the reports presented in this section are the average of 30 repeated simulations where the conditions are kept the same to minimize any statistical error. All the data and plots are available for download via the GitLab repository [Perez Abreu et al., 2020].

The three replication strategies used follow the models of redundancy discussed in Section 6.2: (1) noReplicas, where no replicas are used for all the SCs; (2) vsReplicas, where some of the VFs inside the SC have replicas (the replication ratio is two (2) VF replicas per chain); and (3) End2End, where all the VFs in the SC have replicas.

An important factor to take into consideration is the additional load added to the infrastructure with the different replication strategies (i.e., noReplicas, vs-Replicas, End2End). As more replicas are added, the resources of the nodes are depleted sooner, thus forcing the embedding of the VFs in different nodes (i.e., more nodes in the topology are used), even though the embedding mechanism remains the same. The fact that the embedding mechanisms try to find disjoint nodes for different instances of a given VF magnifies this behavior.

Table 6.7: Execution Time (in seconds).

Replication	**Scenario**	**Mechanism**			
		ILP	**GA**	**FCT**	**FF**
NoReplicas	Tiny	22.253	19.750	0.094	0.014
	Small	38.135	29.192	0.008	0.011
	Medium	50.369	36.349	0.009	0.013
	Large	66.535	52.759	0.007	0.011
vsReplicas	Tiny	27.468	23.488	0.007	0.009
	Small	51.582	42.603	0.007	0.014
	Medium	69.921	49.608	0.008	0.016
	Large	97.516	64.779	0.009	0.011
End2End	Tiny	38.041	35.058	0.008	0.009
	Small	67.793	58.726	0.009	0.009
	Medium	97.511	77.939	0.012	0.019
	Large	139.09	106.37	0.014	0.019

Table 6.7 shows the execution times, in seconds, for each mechanism, scenario, and replication strategy. As the load from the replicas and more SCs increases, so do the execution times. FCT showed the shortest times followed closely by FF. ILP showed times significantly larger, and GA showed times that might not be suitable for online scenarios. The execution times obtained for the mechanisms were as expected; at the top, the ILP requires more time to find the optimal solution to place the VFs. On the other hand, at the bottom, the FCT mechanism was able to find an acceptable placement solution on a negligible portion of time compared to the other two proposed mechanisms.

The execution time results of the proposed mechanisms allow concluding that FCT could be adopted by the *Resilience Manager* module (see Chapter 3) to improve the availability of SCs in the Cloud to IoT continuum. Results for experiments on failure ratio, node utilization, and response time are presented in the following subsections.

6.3.1 Failure Ratio

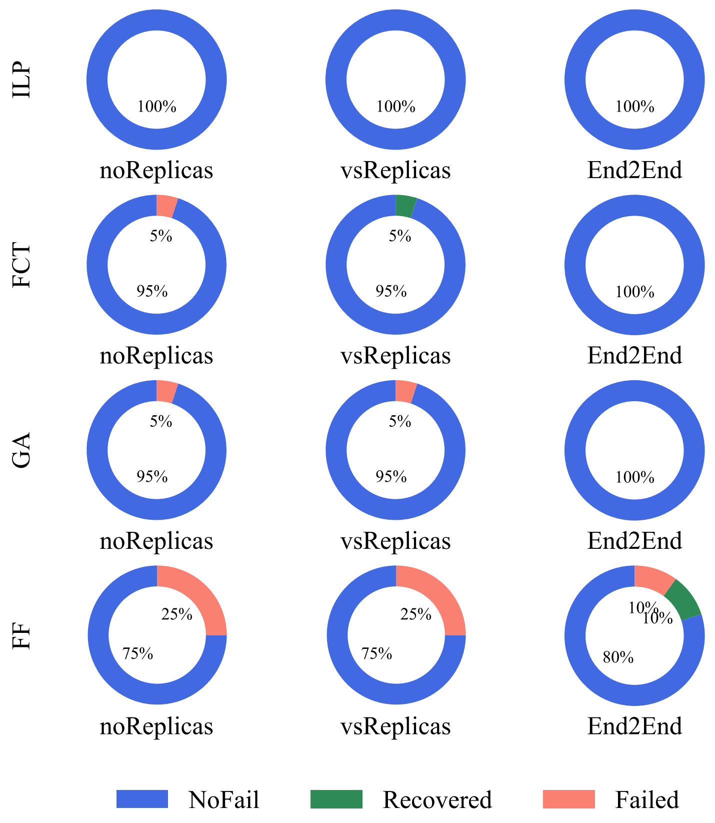
The first experiment is aimed at evaluating the resilience of the Service Chains embedded in the network infrastructure. Three outcomes are possible: NoFail (no VFs from that SC are affected by the node failures), Recovered (some VFs are affected but their replicas could be activated), and Failed (one or more VFs are affected but there is no replica or the replica(s) is also stricken by the failures). Figures 6.4 and 6.5 show the results regarding the failure ratio, where the rows represent the embedding mechanisms and the columns the replication methods.

The failure ratio refers to a node failure that affected a SC, preventing a successful communication among its VFs; because a VF was embedded in the failing node, or because the failing node was a critical part of the communication path. Figure 6.4a depicts the results for the tiny scenario, Figure 6.4b for the small scenario, Figure 6.5a for the medium scenario, and Figure 6.5b for the large scenario. As explained before, the load changes as more replicas are added to the infrastructure, influencing the embedding process.

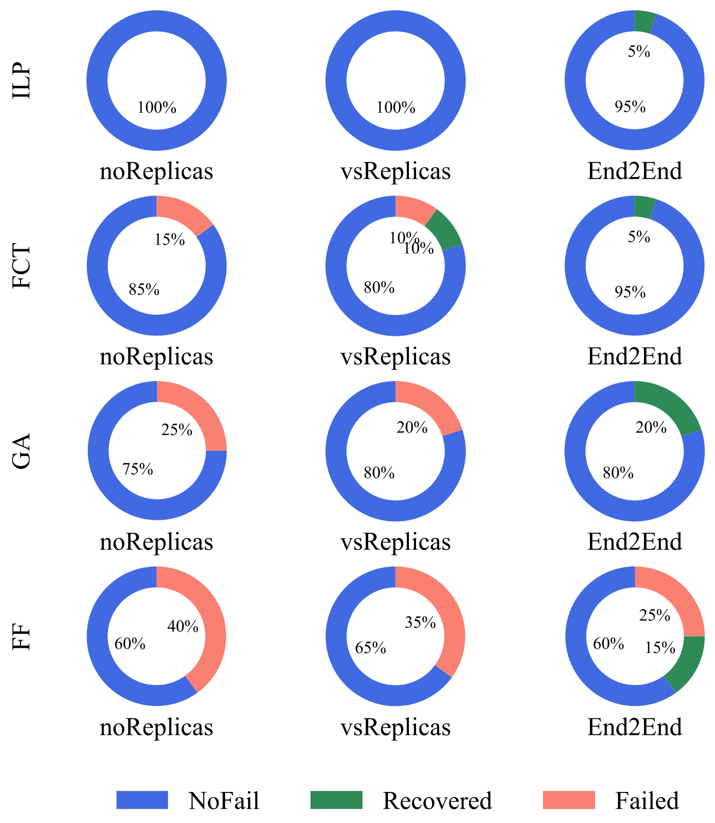
In the tiny scenario (see Figure 6.4a), the presence of failures affects fewer SC, since their VF are more spread in the infrastructure. ILP was the only mechanism without any failures for all the replication methods. When using noReplicas, FCT suffered from 5% of SC failures, and the same amount of recovery for the vsReplicas. GA was not able to recover failures in the noReplicas and vsReplicas methods. Both FCT and GA did not suffer any failures when using the End2End method. FF was the mechanism that showed more failures, with 25% of failing SCs in the noReplicas and vsReplicas. FF was also the only mechanism with failing SCs when using the End2End method in the tiny scenario, with 10% of recovering failures and 10% of the SCs being able to recover.

For the small scenario (see Figure 6.4b), as the load grows (i.e., more replicas) so does the number of failing SCs; since there are more VFs deployed in the nodes. Thus the probability of being deployed in a failing node is higher. ILP was the only mechanism with no unrecovered failing SCs, suffering from failures only when using the End2End method, but being able to recover. FCT suffered from failures when using the three different replication methods, being able to recover half of the affected SCs when using vsReplicas, and all of the failing SCs when using End2End. GA suffered from more failures than ILP and FCT for all the replication methods, being able to recover only for End2End. FF had the highest failure ratio of all the embedding mechanisms in the small scenario for all the replication methods. Specifically, FF was only able to recover when using the End2End replication method but still suffered from failures from which it could not recover (25%).

For the medium scenario (see Figure 6.5a), once again ILP was the only mechanism that did not suffer from irrecoverable failures, being able to activate replicas for the End2End method. FCT suffered irrecoverable failures for all the replication methods, being able to recover from some failures for the vsReplicas (10%) and End2End (15%). GA experienced more failures than ILP and FCT, having



(a) Tiny Scenario.



(b) Small Scenario.

Figure 6.4: Service Chains Failure Ratio - Tiny and Small Scenarios.

the capacity to recover for the vsReplicas (15%) and End2End (35%) but being unable to recover for the vsReplicas (45%) and End2End (10%). FF was the only mechanism incapable of recovering from failures for vsReplicas, showing the highest failure rate among all the mechanisms.

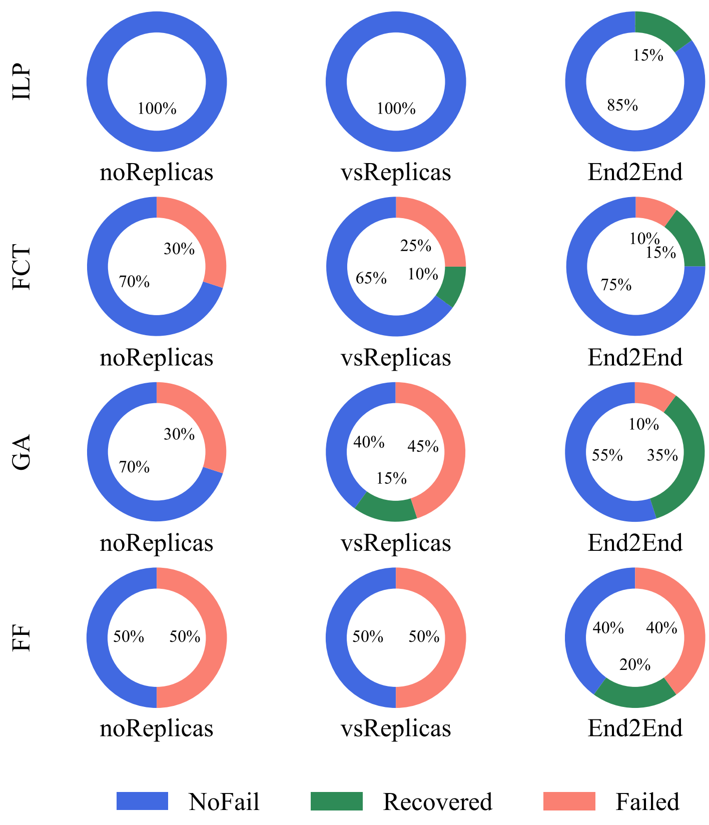
With the heaviest load in the large scenario (see Figure 6.5b), higher failure rates are reported by each embedding mechanism, since more VFs are deployed in the substrate infrastructure increasing the failure probability. ILP suffered from irrecoverable failures in all the replication methods, unlike previous scenarios, but it was able to recover from some failures (25%) for End2End. The ratio of failures that could not be recover increased for FCT for all the replication methods, while the ratio of recovered failures remained relatively stable. GA suffered from failures of around 70% of the SCs for all the replication methods, but was able to recover from most failures (60% out of 80%) for End2End. Once again, FF was the only mechanism incapable of recovering from any failure from vsReplicas, showing the highest failure rate between the embedding mechanisms for all the replication methods.

In general, the behavior observed in the assessment of the mechanisms regarding the failure ratio was as expected. The noReplicas method, for all mechanisms, could not recover from failures since there were no replicas to activate. It is also noticeable the trend that ILP showed the lowest number of failures, followed by FCT, then GA, and finally FF.

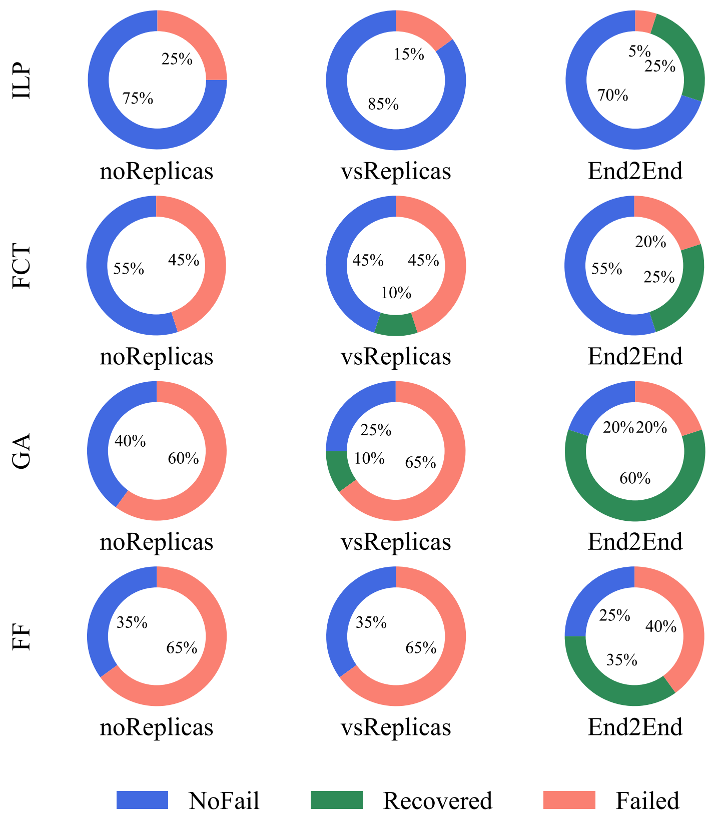
For the vsReplicas, FF could not recover from failures in any scenarios. For the large scenario, the ILP mechanism was also unable to recover from some failures, but the number of failing SCs was significantly lower than with the other mechanisms. FCT was able to recover from some failures in all the scenarios, as well as GA. Particularly, for the large scenario using ILP (see Figure 6.5b) the amount of failing SCs is lower for vsReplicas than noReplicas. This is due to some replicas taking place in nodes with higher availability factor, but that ultimately failed, and being placed in nodes with less availability that turned out not failing during the simulation.

With End2End, since there are replicas for all the VF, all the mechanisms were able to recover some SCs affected by the failures. For ILP in the medium scenario, although all the SCs were able to complete (i.e., recovered after node failures), it is noticeable that for the End2End method, 15% of the SCs were affected by failures. This is caused by the extra load included by the replicas. Using this method, more load is added to the nodes, forcing the embedding of some VFs in different nodes that were affected by failures. Nonetheless, the failing SCs were able to activate the replicas for the corresponding VFs. Using End2End, the amount of SC affected by failures is more significant for all the mechanisms, given the higher number of VF instances (i.e., primary and backup ones) that are embedded. GA was the mechanisms that recovered more SCs in End2End for all the scenarios. However, ILP and FCT did not need to activate as many replicas since these mechanisms were more effective selecting nodes that did not fail during the simulation (i.e., less affected SCs).

In Figures 6.4 and 6.5, ILP was always the most effective mechanism regarding



(a) Medium scenario.



(b) Large scenario.

Figure 6.5: Service Chains Failure Ratio - Medium and Large Scenarios.

the failures, having a higher success rate for all the scenarios. In the case of FCT, it is noticeable that there are fewer SCs affected when using the End2End method. For this case, it is important to remember that FCT bases its embedding decision on the available resources by tier. Thus, by changing the load on the network, the embedding decision process is affected, thus changing the nodes selected for the embedding. For GA also the amount of affected SCs is lower when using the End2End method since GA also combines the encouragement of utilizing lower tiers, as well as, nodes with high availability (higher tiers); thus compromising the embedding of the VFs throughout the entire Cloud to IoT landscape. In any case, FCT showed a behavior close to ILP followed by GA. As the load grows, the rate of affected SCs with GA also increases, being closer to FF although still slightly more efficient in failure recovery. This suggests that GA would require a larger number of generations to converge to a better solution when using heavy loads.

For all the methods and scenarios, FF was the mechanism that showed more failures and less capacity to recover; ILP was the mechanism with better results. It is noteworthy that as the load grows, the number of failing Service Chains increase; and the load also affects the embedding process. Thus the same VF instance can be embedded in different nodes even though the same mechanism is used.

6.3.2 Node Utilization

The next experiment had the objective of evaluating the load balancing of the different mechanisms. Figures 6.6 shows the results for the noReplicas, Figure 6.7 for vsReplicas, and 6.8 for End2End methods. For each replication method, the top plot shows the number of nodes used, while the bottom plot depicts the amount of VFs embedded on the busiest node; this is, the node with more VFs.

Since ILP seeks the optimal node, it will saturate it, thus using fewer nodes overall, in spite of the replication method. As the scenarios grow, the number of nodes used also grows, showing an increasing trend for all the mechanisms; and maintaining the fact that ILP uses the lowest number of nodes between all. However, for ILP the results regarding the VFs on the busiest nodes are the highest among all the mechanisms, suggesting a saturation on the nodes. This might lead to more failures, both in the node as in the communication links.

With regards to the nodes with the highest load, there are different trends among the mechanisms. In the case of FF, all the nodes are treated equally (i.e., no node is prioritized). The VFs are embedded in any case when there are enough resources for it. On average, for these simulation parameters, 8 VFs will fill a node; thus, this is the value observed for the busiest node in all the scenarios. GA also showed a static trend on the number of VFs on the busiest node, but a higher number of nodes are used among all the mechanisms. This means a disparity in the mapping of VFs to nodes, embedding around 10 VFs on the busiest node, but less VFs in other nodes. GA also showed the highest number

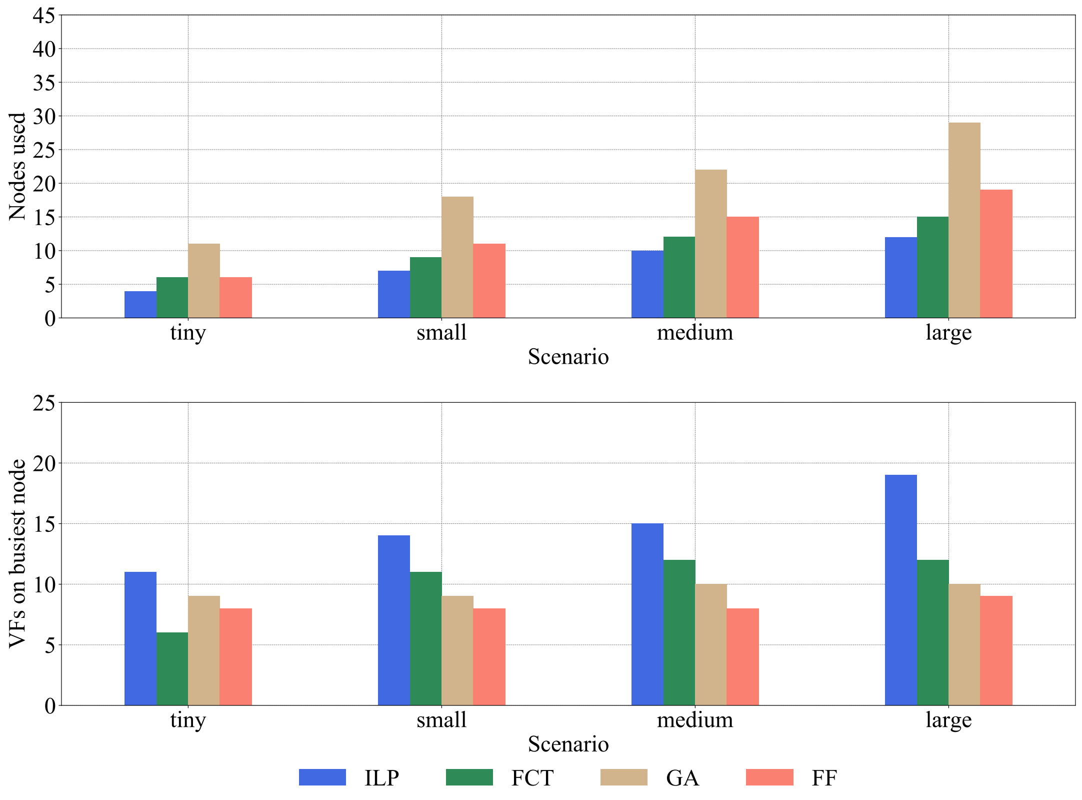


Figure 6.6: Node Utilization - noReplicas.

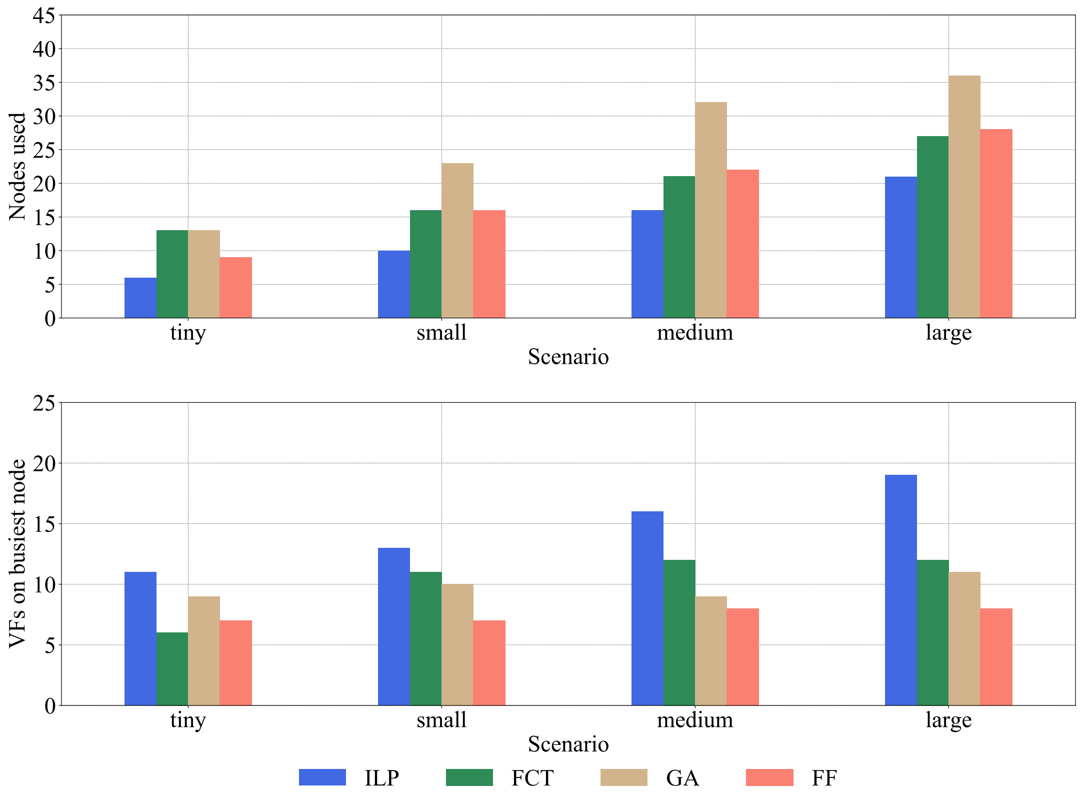


Figure 6.7: Node Utilization - vsReplica.

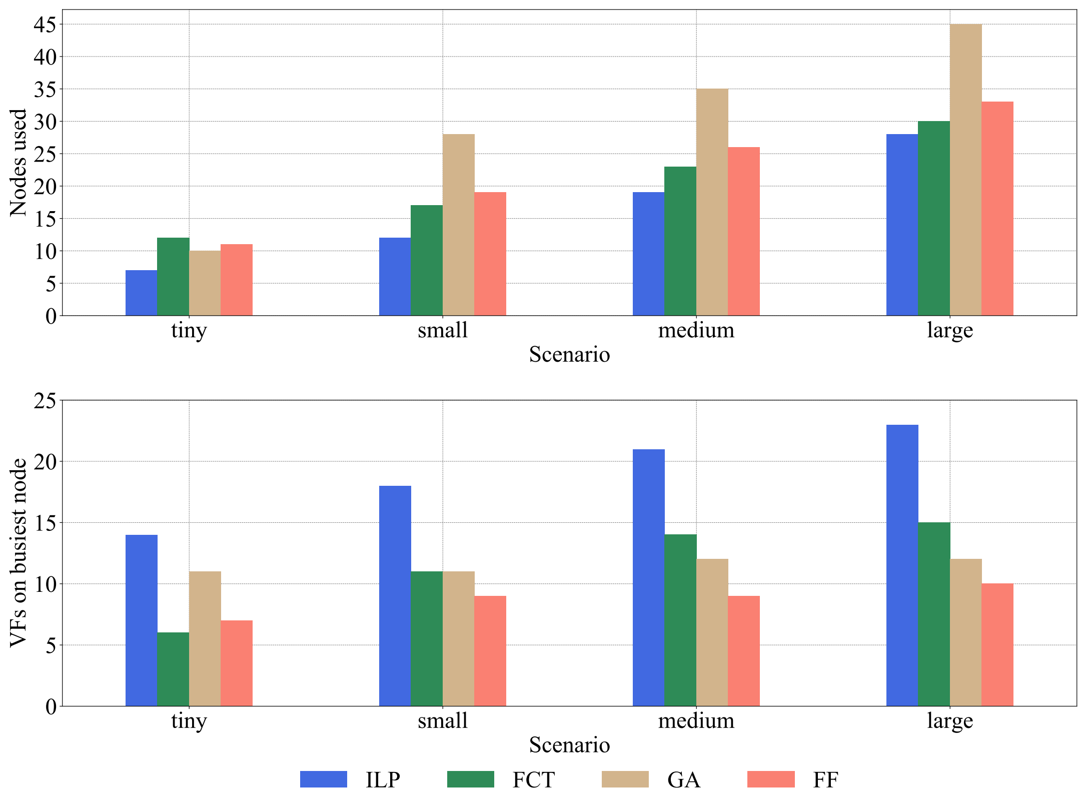


Figure 6.8: Node Utilization - End2End.

of nodes used among all the mechanisms, impacting the response time of the SCs, since more inter-node transmissions are needed to complete the communication of the VFs that are embedded in different and sometimes distant nodes.

ILP and FCT prioritize the nodes according to their availability factor and the tier to where they belong. Furthermore, the acceptance rate is also considered in the embedding process. Thus, smaller VFs are prioritized, favoring them during the embedding, resulting in smaller VFs grouped in the same node, with the additional advantage of lower internal fragmentation in the nodes. This behavior leads to an increasing trend for ILP, with more VFs in the busiest node. FCT had a hybrid behavior by selecting a subset of candidates (i.e., communities) and balancing the load amongst them. Hence, it shows a stable amount of VFs on the busiest node in all the scenarios, getting to balance the load, but also using more nodes than ILP to achieve this. This also reinforces the hypothesis of using a mechanism for load balancing during the embedding process, for instance, using communities; particularly communities similar in size, while also considering the intra-community and inter-community connectivity.

6.3.3 Response Time

The following experiment was designed to evaluate the performance of the SCs from an end-user perspective. The results for this experiment are presented below in Figures 6.9 to 6.14, for the tiny and large scenarios, and all replication methods. The results for the small and medium scenarios present the same trends and are shown in Appendix B to ease the readability of this section. The plots show the average response time (in milliseconds) from successful end-to-end communications. The lack of a bar indicates that there was a node failure during the simulation that affected the corresponding SC, preventing any successful end-to-end communication (i.e., no data was collected). The line on the top of the plot indicates the deadline for each SC (also in milliseconds).

For the tiny scenario using noReplicas (see Figure 6.9), FCT reported on average the best response time, with an improvement of around 30% lower time in the best case against the other mechanisms. FF was the only mechanism that failed to fulfill a complete end-to-end communication. GA was the mechanism with the highest response time, followed closely by ILP.

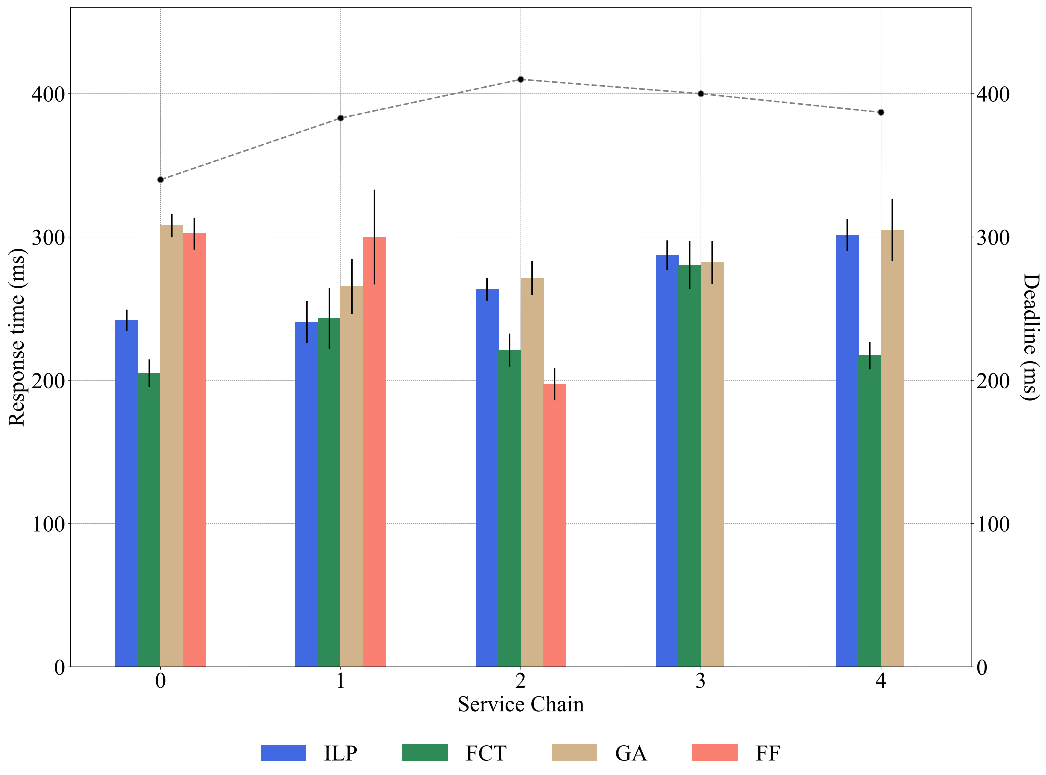


Figure 6.9: SC Response Time - Tiny Scenario - noReplicas.

For the vsReplica method (see Figure 6.10), once more FF was the only mechanism unable to complete end-to-end communication. FCT showed the best results, with response times of up to 30% lower than GA and ILP. In the case of the End2End replication method for the tiny scenario (see Figure 6.11), the worst results were obtained by ILP and GA. However, these mechanisms were able to complete all their end-to-end communications, unlike FF that showed lower response times for the SCs capable of completing all their end-to-end communications. All the mechanisms completed the SCs within their deadlines for all the replication methods in the tiny scenario, for those SCs that achieved successful end-to-end communications.

For the scenario with the heaviest load (i.e., the large one), the difference in re-

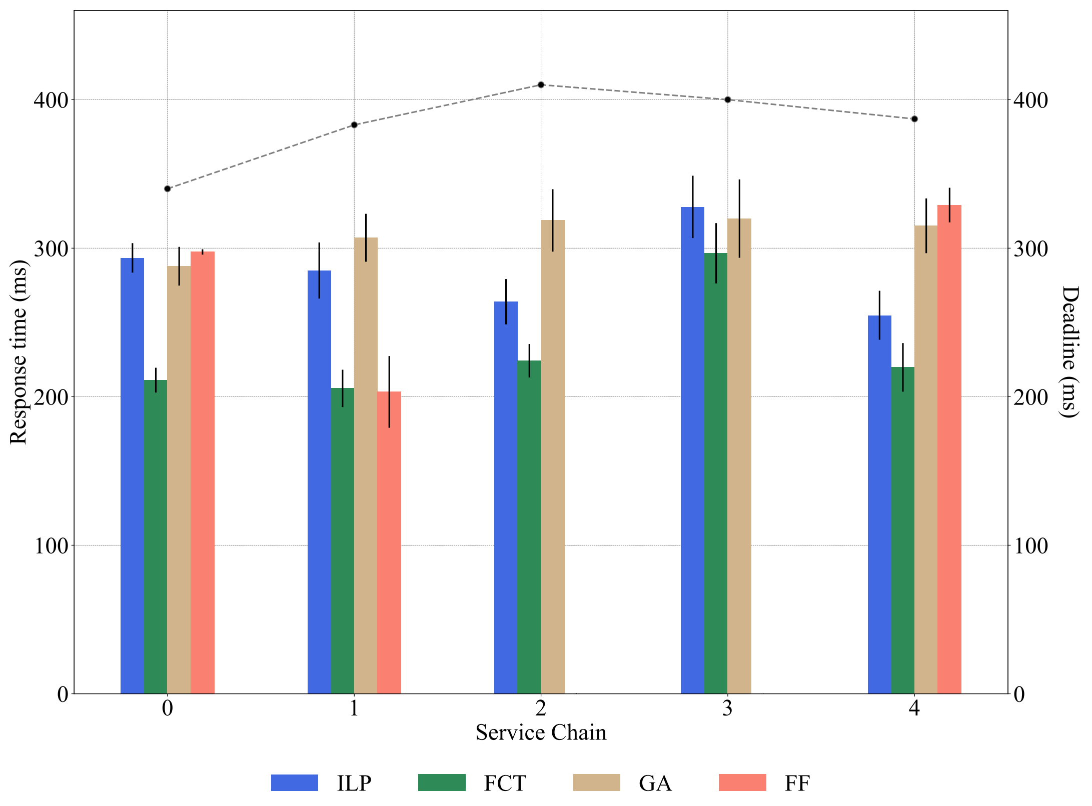


Figure 6.10: SC Response Time - Tiny Scenario - vsReplicas.

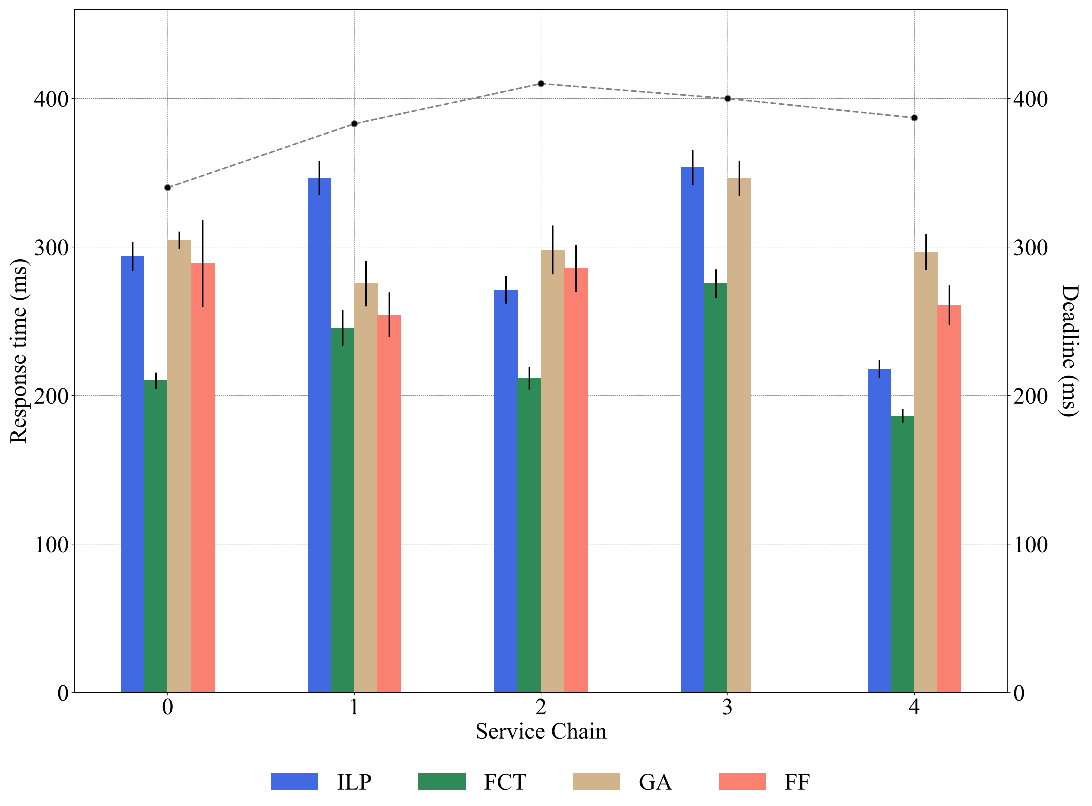


Figure 6.11: SC Response Time - Tiny Scenario - End2End.

sponse time perceived with the different mechanisms are more significant. When using noReplicas (see Figure 6.12), GA and FF were not capable of successfully completing end-to-end communications for all the SCs, with 10% of the SCs failing with GA and 15% with FF. ILP and GA showed times significantly larger than FCT for most of the Service Chains.

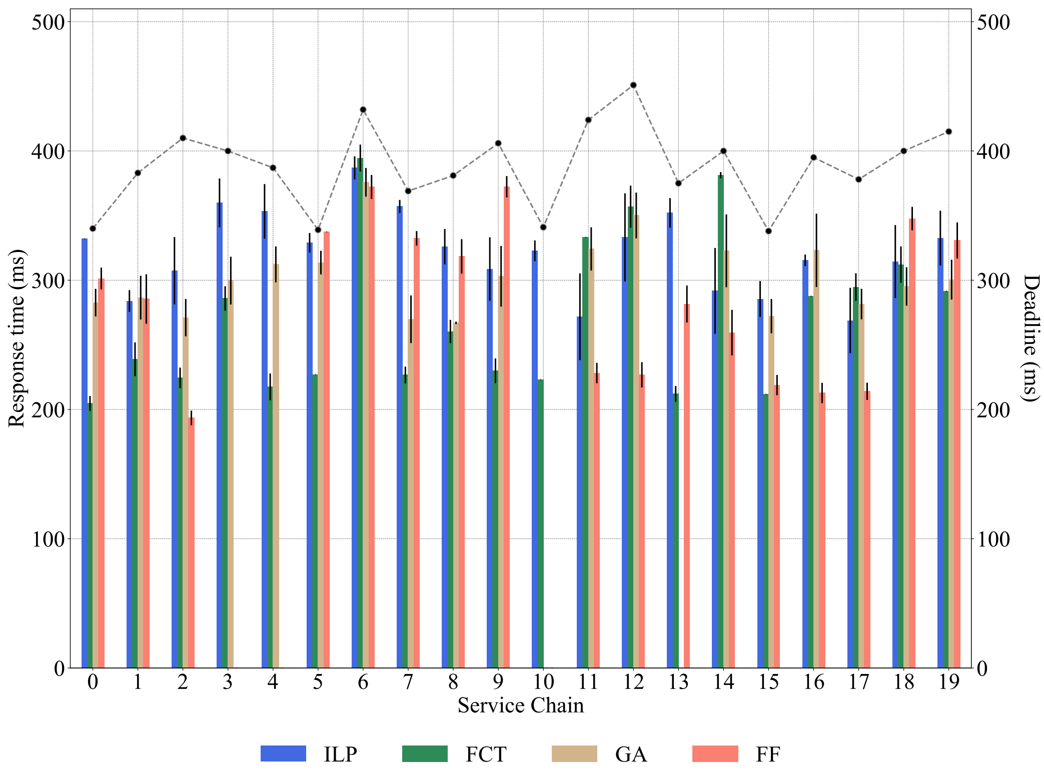


Figure 6.12: SC Response Time - Large Scenario - noReplicas.

For vsReplicas (see Figure 6.13), FCT was the only mechanism able to show results for all the SCs, since ILP, GA, and FF had SCs affected by failures at the beginning of the simulation (5% for ILP, 5% for GA, and 15% for FF), preventing them from achieving any successful and-to-end communication. On average, the lowest response times were reported by FCT, and the highest by ILP. Figure 6.14 shows the results for End2End. For this replication method, GA and FF were unable to complete successful end-to-end communications for 20% and 10% of the SCs, respectively. The lowest response times on average were obtained by FCT, with a 57% improvement respecting ILP and GA, and 30% against FF in the best case (see SC 9 in Figure 6.14). For the SCs that achieved successful end-to-end communications, all the mechanisms completed the SCs within their deadlines for all the replication methods in the large scenario.

The results obtained for these experiments allow to point out the mechanisms that more frequently failed to fulfill a complete successful end-to-end communication for at least one SC were GA and FF. This is due to them suffering from more failures than ILP and FCT (node failures that prevented a successful communication, as seen in Figures 6.4 and 6.5). Thus, there is a higher

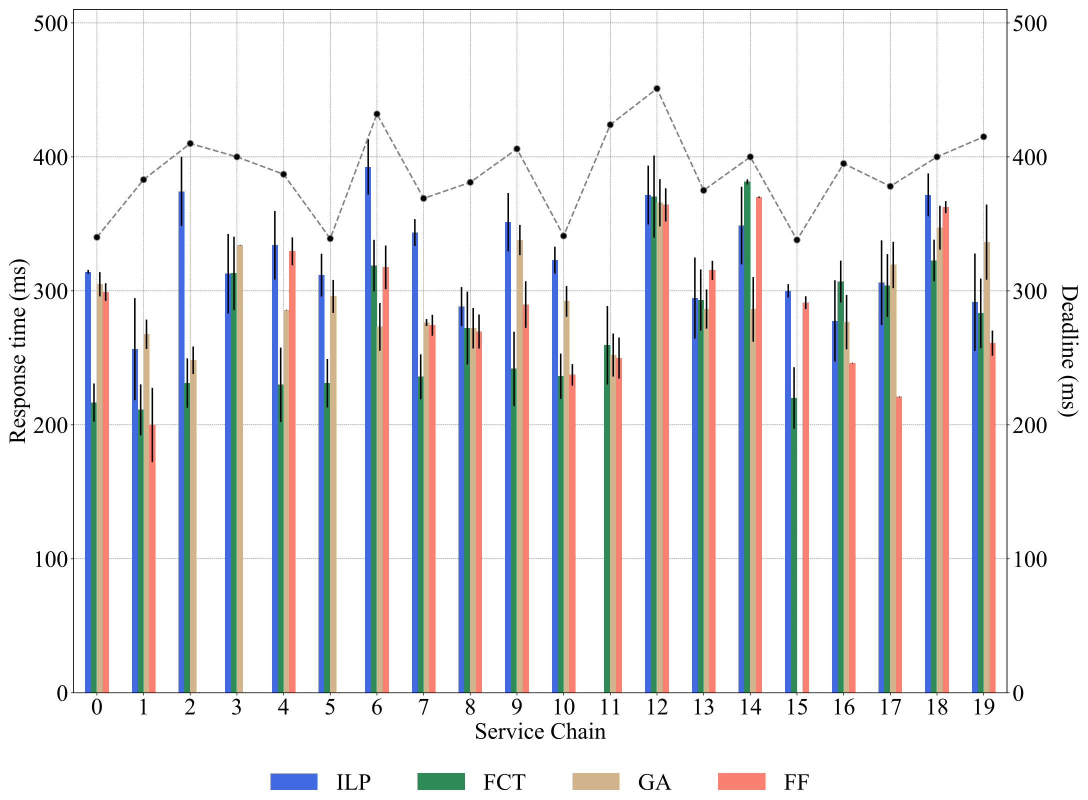


Figure 6.13: SC Response Time - Large Scenario - vsReplicas.

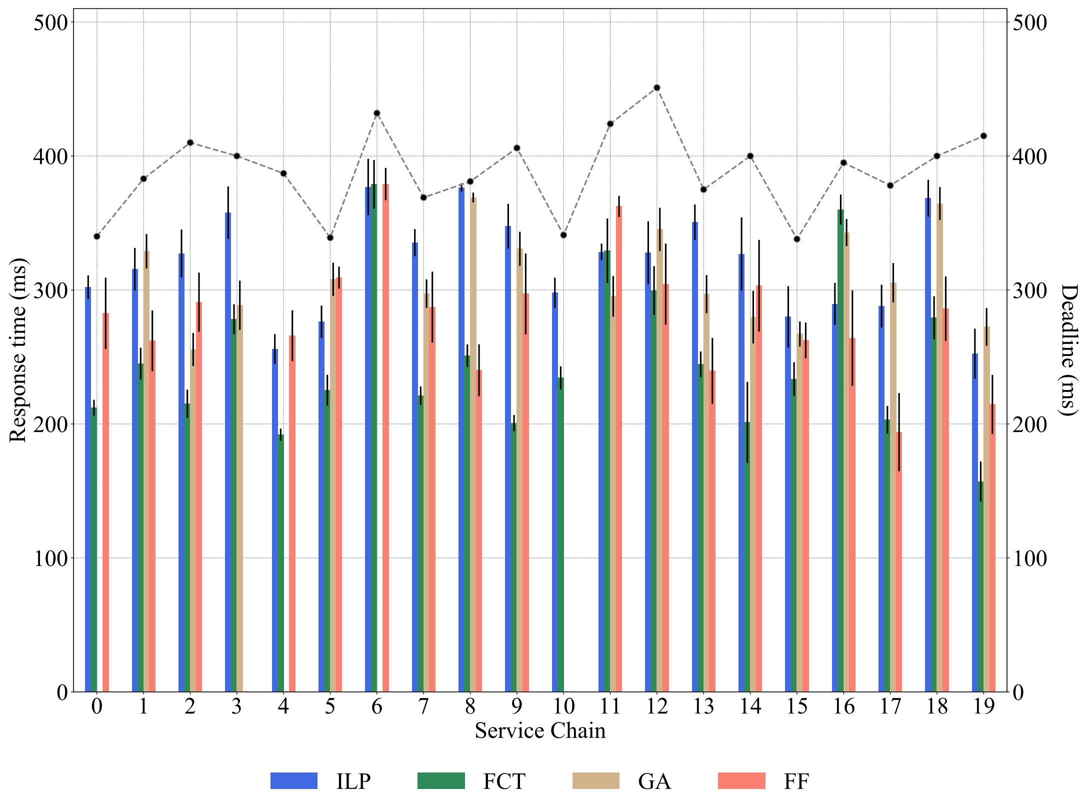


Figure 6.14: SC Response Time - Large Scenario - End2End.

probability that the failure that affects the SCs occurs at the beginning of its workflow. Particularly, in Figure 6.12 is noticeable that with FF 3 SCs were not able to complete any end-to-end communication (i.e., SCs 3, 4, 10) and 2 with GA (i.e., SCs 10, 13). For all the scenarios, FF failed to complete a successful communication for all the SCs, showing at least 1 affected SC.

As the load grows, so does the number of affected SCs for GA, reinforcing the hypothesis that more generations are needed in heavier scenarios to converge to a more efficient solution. On average, all the chains were able to complete their end-to-end communications within their deadline. In most cases, FCT showed the best response times. FCT distributes the VFs among the different tiers of the landscape, and by bringing some VFs closer to the user, the response time is improved. On the other hand, since ILP seeks the node with the highest availability, and these nodes are on the highest tiers of the infrastructure, the VFs are embedded farther away from the user, impacting the response time. FF showed the most stable response times (i.e., having fewer differences among the SCs). For the scenarios with lower load (i.e., tiny), ILP showed better response times than GA, but as the load grows (i.e., large scenario) this trend shifts. This is due to the saturation of the busiest node with heavier loads when using ILP.

6.3.4 Discussion

Overall, ILP showed the best results regarding the resilience and survivability, and the number of nodes used. The outcomes on the resilience and survivability experiment were expected, considering that the model finds an optimal solution for embedding the VFs and their replicas. However, this embedding mechanism requires more time to reach a result; thus, it is not suitable for more complex scenarios that demand quick response times for the embedding. As for the number of nodes, the fact that the ILP model concentrates the VFs in the same optimal nodes leads to a bottleneck for these nodes and their respective links, affecting the performance and response times of the SCs that go through them.

On the other hand, FCT achieved a better load balancing for the nodes by spreading the VFs along the different tier nodes of the infrastructure, which are grouped into communities. Furthermore, by implementing a vertical search inside the communities considering the amount of resources available by tier, and exploiting the possibility of embedding the VFs closer to the edge of the network, FCT obtained better response times, ultimately increasing the QoS for final users while offering a close to the optimal ratio of recovery from failures.

GA showed a recovery ratio close to FCT especially for scenarios with lower load; displaying a more significant gap among these mechanisms as the load grows. GA was also the mechanism with the higher number of nodes used while maintaining a low number of VFs on the busiest node, leading to a higher service spread in the underlying network.

FF was the least effective mechanism of the group, showing the lowest recovery rate from failures. Additionally, some of the Service Chains were not able to complete a successful end-to-end communication. This proves that even though this is a simple mechanism, it is not necessarily adequate to be applied in the Cloud-Fog-Mist-IoT service infrastructure.

Considering the results obtained in the experiments performed and the discussion presented above, it is possible to conclude that FCT exhibits a balance between recovery from failures and response time of applications, making it suitable for the Cloud to IoT landscape modeled in this research.

6.4 Summary

The evaluation of the embedding mechanisms designed and implemented was presented in this chapter. After a detailed analysis of different simulation tools for Cloud/Fog environments, YAFS was selected for the evaluation because of its strong support for Fog features and its capacity for fault injection. This last feature was critical to validate the behavior of the resilient embedding mechanisms under failures. Three replication methods were used for the experiments: no replicas (noReplicas), some VFs had replicas (vsReplicas), and all the VFs had replicas (End2End).

The ILP resulted in the embedding of VFs in mostly Fog and Mist nodes, and also in the saturation of nodes that were selected as optimal (based on their availability factor). This saturation is counterproductive, potentially causing more failures in both the nodes and their communication links. ILP was the mechanism with less unrecovered failures and an overall lower failure ratio.

The GA was able to recover from all failures in the scenarios with lower load (i.e., tiny and small) but had some SCs affected by failures in the scenarios with heavier load (i.e., medium and large). GA was also the mechanism that used the larger number of nodes, generating the larger service spread in the network; however, this did not result in a lower failure ratio.

FCT showed a failure ratio close to ILP without saturating the nodes. Since FCT performs a vertical search to select the node in which the VF will be embedded, and by taking into consideration the amount of free resources in each layer, FCT was able to get lower response times for the SCs.

For all the analyzed scenarios and replication methods FF showed less capacity to recover from failures, with the highest failure rate among all the mechanisms. Along with GA, FF was not able to get any successful end-to-end communication for some of the SCs, although for FF this happened in all the scenarios, for GA it only happened in the medium and large scenarios.

FCT got to balance the load among the nodes while improving the response time of the SCs and experiencing fewer failures than FF and GA.

The contributions of this chapter are as follows:

- An evaluation of Cloud to IoT simulation tools, describing their functional and non-functional characteristics, to help in the selection of the proper simulation tool according to the experimental goals; and

- An assessment of the performance of the proposed embedding mechanisms using simulation. The appraisal included measurements of failure rate, node utilization, and response time of the Service Chains embedded.

After a successful validation and assessment of the embedding mechanisms designed and implemented in this work, the next chapter recapitulates the work performed during the PhD, as well as the outcome contributions. Additionally, a discussion of possible future paths on improving resilience in the Cloud to IoT continuum is presented.

Chapter 7

Conclusions and Future Work

Contents

New solutions to support the daily activities of citizens, like smart traffic control, health care, public safety, among others, are being incorporated in the paradigm that intends to interconnect objects in the scope of a city, also known as a Smart City. Cloud computing, Fog, and IoT seem to be some of the most critical technologies that are going to constitute the foundation for this reality. However, there are still open issues that need to be tackled to bring flawless interoperability between these paradigms to reality.

One of these open issues is the necessity of maintaining the availability of the services/functions hosted in the ICT infrastructure that interconnects smart objects and components within the IoT. Particularly, resilience is a challenge since it requires keeping track of a highly changing environment from the infrastructure point of view and also from the requirements of the applications. This represents the necessity of implementing new mechanisms to enable the dynamic and distributed management orchestration of the infrastructure supporting the Cloud to IoT continuum.

7.1 Synthesis of the Thesis

This work is focused on providing solutions to improve the resilience for virtualized services/functions within the context of Smart City applications. The work is divided as follows.

Chapter 2 presented the research context, defining resilience in the Cloud to IoT. The chapter also introduced basic concepts on the different technologies that support the Smart City paradigm, including IoT, Cloud and Fog computing, and virtualization. Furthermore, the chapter provides a revision about the state of the art on resilience for virtualized services/functions in the Cloud to IoT continuum, identifying open issues and research paths to explore in order to improve the resilience of smart services.

Next, in Chapter 3 a novel architecture with resilience features for the IoT environments was proposed. The architecture takes into consideration the communication infrastructure requirements of the IoT, besides it uses the advantages of the Cloud and Fog paradigms to add ubiquity and scalability to the environment. The interaction between components in the IoT middleware combined with the proper technologies allows fulfilling an efficient communication process between smart objects and final users. Particularly, the proposed architecture was designed taking into consideration the Smart City scenario, and the requirements needed for its computing and communication platform. The architecture is mainly focused on enhancing the resilience of virtualized smart services.

Additionally, Chapter 3 presented an ontology to describe the IoT infrastructure, to handle its heterogeneity improving the interoperability of different mechanisms working in the architecture. The ontology comprises as main classes the

IoT infrastructure, its devices, communication interfaces and links, and the performance metrics related to them. This information will ease the managing of the infrastructure in order to guarantee its proper work for user services and applications. The ontology was successfully validated with different procedures, including consistency evaluation and queries to corroborate its correctness.

After offering a solution for resilient management and administration of the Cloud to IoT continuum to support Smart City applications, it was identified that the *Resilience Manager* from the proposed architecture needed to provide robustness and survivability to the smart virtualized services, identified as Virtual Functions and organized in a structure called Service Chain. To fulfill this goal, Chapter 4 presented a framework for the composition and embedding of Service Chains. The framework is divided into two main modules, one for the *Request Analysis* and one for the *Request Embedding*.

Chapter 4 offers the description of the first module, including a formal grammar that allows the customized specification of SC requests and the replicas of their constituent VFs. The grammar allows to specify the tier in which the VF should be embedded, the number of replicas, and the order of the VFs inside the SC. In the case that an arbitrary ordering is allowed and multiple SCs equally satisfy the same request, a Pareto analysis is applied to select the resulting SC to embed, based on their data rate, resource usage, and the number of VFs instances. Chapter 4 closes with a discussion about how to deal with embedding from a general point of view before moving forward to the specific details of the *Request Embedding* module.

Chapter 5 handles the second module of the framework, which is in charge of the embedding of the SCs. Three embedding mechanisms are proposed for the VFs and their replicas. The first mechanism is based in ILP and aims at increasing the acceptance rate while also maximizing the availability of the nodes on which the VFs will be embedded. The model also guarantees that the replicas of a single VF are embedded in disjoint nodes (if possible), so in the case of a failure the replica can be activated. The second mechanism is based on GAs and uses a weighted sum approach to combine multiple objectives in the same fitness function. The objectives include maximizing the availability of the nodes selected for the embedding, prioritizing the use of disjoint nodes for the replicas, and weighting the different tiers of the Cloud to IoT continuum to distribute the VFs along the entire landscape instead of concentrating them in the upper tiers of the network, where the nodes with higher availability value are gathered, and thus improving the response time of the SCs. The third and final mechanism is a heuristic based on graph partitions to create balanced *communities* in which the SCs will be embedded. The heuristic first tries to select the tier with more available resources and then selects the node with higher availability from that tier for each community. This way the load is spread along the network landscape.

All the mechanisms are appraised via simulations. Chapter 6 begins with an analysis of different Cloud/Fog simulation tools that presents their different characteristics so that a researcher can easily select the one more suited for their study. Then the evaluation setup is characterized so the experiments described in this chapter can be recreated. Finally, the evaluation of the embedding mechanisms is presented, including experiments on SCs failure rate, infrastructure node utilization, and SCs response time. Results show that the ILP mechanism had the lower failure rate, while also incurring in node saturation. The GA showed results close to the optimal ILP for lower loads, but the difference got bigger for larger loads, implying the need to use more generations (and thus more computing power and execution time) to reach a better solution. The heuristic, called FCT, got results close to the ILP optimal for all the scenarios, but was the mechanism with better results for the response time, showing to be able to improve not only the resilience but also the performance of the SCs.

7.2 Contributions

The research work conducted in the context of this thesis was driven by the objectives described in Chapter 1. With these objectives in mind, this thesis led to the contributions described below:

- **A resilient architecture for the Cloud to IoT continuum, including the characterization of its modules and their interaction.** The proposed architecture takes into consideration the computing and communication requirements of the IoT, besides it uses the advantages of the Cloud and Fog paradigms to add ubiquity and scalability to the environment. The interaction between components in the *IoT Middleware* combined with the proper technologies allows fulfilling an efficient computing and communication process between the Smart Objects and the end-users. The architecture is accompanied by an ontology to describe the IoT infrastructure, which provides a generic and uniform mechanism to identify smart objects within the computing and network infrastructure, allowing the interoperability among modules of the architecture and also of solutions offered by different service providers, enabling the use of a federated approach;
- **A framework for the composition and embedding of Service Chains.** The framework not only defines the actions needed to define a SC and to embed it but also includes original solutions for both tasks. Regarding the composition of SCs and their description, a context-free grammar to characterize and verify the correctness of SCs is provided. The grammar allows the specification of the number of replicas for each VF and the tier in which the VF should be embedded. Also, the SCs could be given a specific order or not, in the case of disordered SCs, a solution to select the most adequate SC based on a Pareto analysis is also offered;
- **A conceptual and technical review of simulation tools for the Cloud to IoT continuum.** From a set of simulators specially designed to model the particularities of the Cloud, Fog, Mist, and IoT, a revision of their technical and non-technical characteristics is provided. This analysis will help the research community in the selection of the proper simulation

tool, a critical step in the evaluation process for the design and validation of novel mechanisms for this landscape; and

- **For the embedding of the Service Chains**, three different mechanisms are also presented. **A mathematical model based on ILP to optimize the embedding of VFs** that belong to a Service Chain, using replicas to increase their availability; **an embedding approach based in a Genetic Algorithm that uses the availability of the nodes to compare different solutions,** as well as information on how disjoint are the replicas of the VFs; and **a heuristic based on the Fluid Communities algorithm, using a multi-tier approach**, to maximize the availability of the Service Chains while improving their performance.

All the embedding mechanisms proposed were evaluated using YAFS and the source code of the experiment performed is available to the research community via a GitLab repository [Perez Abreu et al., 2020]. These proposals, as well as the literature review performed, are published in nine international conference papers, five journal articles, and one book chapter.

7.3 Future Work

Several options have been defined as future work. For instance, regarding the architecture proposed, there is the need to design new mechanisms for other modules including path planning capable of dealing with the dynamism of the environment (e.g., urban traffic congestion and mobile users) allowing to change the route of the data paths in real-time without consequences in the QoS required by the components involved in the communication. Moreover, mechanisms to guarantee the continuity of the service have to be provided for the aforementioned use-cases. Protection strategies based on overlapping topologies and smart migration of services, as well as recovery approaches to reroute the data after a failure using virtualization techniques for network devices, including the complete view of the system (applications, Cloud, Fog, Mist, and IoT infrastructure) have to be developed.

Regarding the SC composition framework, it would be interesting to extend it to have more flexibility in the design of the SCs; for example, having the option to define parallel VF for load balancing. Additionally, other metrics for the Pareto analysis could be included, and analyze the impact of the resulting SCs on the costs for the service providers, and the QoS perceived by the user.

Another future path for research is defining a resilience scale to characterize the resilience level required by each SC. Using this scale, and for a given SC, a study could be made to analyze the chain structure to determine its key VFs and automatically suggest the number of replicas for each VF considering a set of user requirements. In this context, it would be interesting to apply machine learning techniques in the study of the request SCs using data collected from the communication infrastructure provider.

Unquestionably, it would be interesting to add more dynamism to the scenarios to study the repercussions of the changes in the availability of the nodes for the embedding mechanisms. Additionally, proposing a learning mechanism that allows taking into consideration previous failures and updating the availability factor of each node.

Finally, the implementation and performance measure of the proposed resilience mechanisms in the context of a MANO module for Kubernetes or Docker solutions could be a strong contribution of this research to the industry.

Bibliography

Abade, B. (2018). *Context-Aware Improved Experiences in Smart Environment.* Msc thesis, University of Coimbra, DEEC.

Abade, B., Perez Abreu, D., and Curado, M. (2018). A Non-Intrusive Approach for Indoor Occupancy Detection in Smart Environments. *MDPI - Sensors*, 18(11):1–18.

Ahmadian, N., Lim, G. J., Cho, J., and Bora, S. (2020). A quantitative approach for assessment and improvement of network resilience. *Reliability Engineering and System Safety*, 200(1):106977.

Ahmed, T., Alleg, A., and Marie-Magdelaine, N. (2019). An architecture framework for virtualization of iot network. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 183–187, Paris, France. IEEE.

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2007). *Compilers: Principles, Techniques, and Tools.* Pearson Education, Inc, 2 edition.

Aidi, S., Zhani, M. F., and Elkhatib, Y. (2018). On improving service chains survivability through efficient backup provisioning. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 108–115, Rome, Italy. IEEE.

Akusok, A., Bjork, K. M., Miche, Y., and Lendasse, A. (2015). High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications. *IEEE Access*, 3(1):1011–1025.

Al-Turjman, F. M., Hassanein, H. S., Alsalih, W. M., and Ibnkahla, M. (2011). Optimized relay placement for wireless sensor networks federation in environmental applications. *Wireless Communications and Mobile Computing*, 11(12):1677–1688.

Alam Khan, M. M., Shahriar, N., Ahmed, R., and Boutaba, R. (2016). Multi-path link embedding for survivability in virtual networks. *IEEE Transactions on Network and Service Management*, 13(2):253–266.

Atzori, L., Iera, A., Morabito, G., and Nitti, M. (2012). The Social Internet of Things (SIoT)–when social networks meet the internet of things: Concept, architecture and network characterization. *Computer Networks*, 56(16):3594–3608.

Bandyopadhyay, S., Sengupta, M., Maiti, S., and Dutta, S. (2011). A Survey of Middleware for Internet of Things. In *Recent Trends in Wireless and Mobile Networks*, chapter 9, pages 288–296. Springer Berlin Heidelberg.

Barcelo, M., Correa, A., Llorca, J., Tulino, A. M., Vicario, J. L., and Morell, A. (2016). Iot-cloud service optimization in next generation smart environments. *EEE Journal on Selected Areas in Communications*, 34(12):4077–4090.

Bassi, A., Bauer, M., Fiedler, M., Kramp, T., Van Kranenburg, R., Lange, S., and Meissner, S. (2013). *Enabling things to talk*. Springer, 1 edition.

Bauer, E. and Adams, R. (2012). *Reliability and availability of cloud computing*. John Wiley & Sons, 1 edition.

Bays, L. R., Gaspary, L. P., Ahmed, R., and Boutaba, R. (2016). Virtual network embedding in software-defined networks. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 10–18, Istanbul, Turkey. IEEE.

Beck, M. T. and Botero, J. F. (2017). Scalable and coordinated allocation of service function chains. *Computer Communications*, 102(1):78–88.

Beheshti, N. and Zhang, Y. (2012). Fast failover for control traffic in Software-defined Networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2665–2670, Anaheim, USA. IEEE.

Ben Jemaa, F., Pujolle, G., and Pariente, M. (2016). Cloudlet and NFV-based carrier Wi-Fi architecture for a wider range of services. *Annals of Telecommunications*, 71(1):617–624.

Bichot, C.-E. and Siarry, P. (2011). *Graph partitioning*. Wiley, 1 edition.

Brogi, A., Forti, S., and Ibrahim, A. (2017). How to Best Deploy Your Fog Applications, Probably. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 105–114, Madrid, Spain. IEEE.

Carpio, F., Dhahri, S., and Jukan, A. (2017). Vnf placement with replication for loac balancing in nfv networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, Paris, France. IEEE.

Chen, J., Liu, H., and Jia, H. (2020). Cross-layer resource allocation in wireless-enabled nfv. *IEEE Wireless Communications Letters*, 9(6):879–883.

Chen, M., Hao, Y., Li, Y., Lai, C. F., and Wu, D. (2015). On the computation offloading at ad hoc cloudlet: architecture and service modes. *IEEE Communications Magazine*, 53(6):18–24.

Chiang, M. and Zhang, T. (2016). Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864.

Cholda, P., Mykkeltveit, A., Helvik, B., Wittner, O., and Jajszczyk, A. (2007). A survey of resilience differentiation frameworks in communication networks. *IEEE Communications Surveys & Tutorials*, 9(4):32–55.

Cholda, P., Tapolcai, J., Cinkler, T., Wajda, K., and Jajszczyk, A. (2009). Quality of resilience as a network reliability characterization tool. *IEEE Network*, 23(2):11–19.

Chowdhury, N. M. K. and Boutaba, R. (2010). A survey of network virtualization. *Computer Networks*, 54(5):862–876.

Chowdhury, N. M. M. K., Rahman, M. R., and Boutaba, R. (2009). Virtual network embedding with coordinated node and link mapping. In *IEEE INFOCOM 2009*, pages 783–791, Rio de Janeiro, Brazil. IEEE.

Chowdhury, S. R., Salahuddin, M. A., Limam, N., and Boutaba, R. (2019). Re-architecting nfv ecosystem with microservices: State of the art and research challenges. *IEEE Network*, 33(3):168–176.

Curado, M., Madeira, H., da Cunha, P. R., Cabral, B., Perez Abreu, D., Barata, J., Roque, L., and Immich, R. (2019). Internet of Things. In: Cyber Resilience of Systems and Networks. In *Cyber Resilience of Systems and Networks*, chapter 2, pages 381–401. Springer International Publishing.

da Fonseca, N. L. S. and Boutaba, R. (2015). *Cloud Services, Networking, and Management*. Wiley Online Library, 1 edition.

Dastjerdi, A. V. and Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116.

Datta, S., Bonnet, C., and Nikaein, N. (2014). An IoT gateway centric architecture to provide novel M2M services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 514–519, Seoul, South Korea. IEEE.

De Turck, F., Kang, J.-M., Choo, H., Kim, M.-S., Choi, B.-Y., Badonnel, R., and Hong, J. W.-K. (2017). Softwarization of networks, clouds, and internet of things. *International Journal of Network Management*, 27(2):e1951.

Deo, N. (2017). *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 1 edition.

ElDefrawy, K. and Kaczmarek, T. (2016). Byzantine fault tolerant software-defined networking (sdn) controllers. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, pages 208–213, Atlanta, USA. IEEE.

ETSI GS NFV (2013). Network functions virtualisation (nfv); use cases. Technical Report ETSI GS NFV 001 v1.1.1, European Telecommunications Standards Institute.

ETSI GS NFV-REL (2016). Network functions virtualisation (nfv); reliability; report on models and features for end-to-end reliability. Technical Report ETSI GS NFV-REL 00. v1.1.1, European Telecommunications Standards Institute.

Fernandes, J., Perez Abreu, D., Velasquez, K., Mateus, M., ao Carrilho, J., Silva, M., Monteiro, E., and Martins, A. (2017a). Building a smart city iot platform - the suscity approach. In *48nd Spanish Congress on Acoustics and the Iberian Encounter on Acoustics (TECNIACUSTICA)*, pages 1–9, Coruña, Spain. Sociedad Española de Acústica (SEA).

Fernandes, J., Perez Abreu, D., Velasquez, K., Monteiro, E., and Martins, A. (2017b). An architecture to support affordable internet of things applications: The suscity project case study. In *8th Congresso Luso-Moçambicano de Engenharia V Congresso de Engenharia de Moçambique (CLME2017 - V CEM)*, pages 1055–1056, Maputo, Moçambique. INEGI/FEUP.

Fernandez-Anez, V. (2016). Stakeholders approach to smart cities: A survey on smart city definitions. In *Smart Cities*, pages 157–167, Málaga, Spain. Springer International Publishing.

Fischer, A., Botero, J. F., Beck, M. T., de Meer, H., and Hesselbach, X. (2013). Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials*, 15(4):1888–1906.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3):75–174.

Galis, A., Clayman, S., Mamatas, L., Rubio Loyola, J., Manzalini, A., Kuklinski, S., Serrat, J., and Zahariadis, T. (2014). Softwarization of future networks and services -programmable enabled networks as next generation software defined networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, Trento, Italy. IEEE.

Gomes, R. L., Bittencourt, L. F., Madeira, E. R., Cerqueira, E., and Gerla, M. (2016). Bandwidth-aware allocation of resilient Virtual Software Defined Networks. *Computer Networks*, 100(5):179–194.

Groups, G. (2019). CloudSim - Google community group. `https://groups.google.com/forum/#!forum/cloudsim`. Last visited: 2019-05-10.

Grover, W. D. (2003). *Mesh-based Survivable Transport Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking.* Prentice Hall PTR, 1 edition.

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

Gupta, A., Habib], M. F., Mandal, U., Chowdhury, P., Tornatore, M., and Mukherjee, B. (2018). On service-chaining strategies using virtual network functions in operator networks. *Computer Networks*, 133:1–16.

Han, X., Cao, X., Lloyd, E. L., and Shen, C. C. (2010). Fault-tolerant relay node placement in heterogeneous wireless sensor networks. *IEEE Transactions on Mobile Computing*, 9(5):643–656.

Hao, Y., Linke, G., Ruidong, L., Asaeda, H., and Yuguang, F. (2014). DataClouds: Enabling Community-Based Data-Centric Services Over the Internet of Things. *Internet of Things Journal*, 1(5):472–482.

He, M., Alba, A. M., Basta, A., Blenk, A., and Kellerer, W. (2019). Flexibility in softwarized networks: Classifications and research challenges. *IEEE Communications Surveys & Tutorials*, 21(3):2600–2636.

Hmaity, A., Savi, M., Musumeci, F., Tornatore, M., and Pattavina, A. (2017). Protection strategies for virtual network functions placement and service chains provisioning. *Networks*, 70(4):373–387.

Hou, L., Zhao, S., Xiong, X., Zheng, K., Chatzimisios, P., Hossain, M. S., and Xiang, W. (2016). Internet of things cloud: Architecture and implementation. *IEEE Communications Magazine*, 54(12):32–39.

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

IBM Industry Solutions (2013). IBM Smarter Cities - Creating opportunities through leadership and innovation. IBM - `https://goo.gl/7aLzQM`. Last visited: 2019-05-19.

IETF (2015). Routing Over Low power and Lossy networks Working Group. `http://datatracker.ietf.org/wg/roll/charter`. Last visited: 2016-01-30.

Internet of Things Architecture (2010). IoT-A. `http://www.iot-a.eu`. Last visited: 2016-05-10.

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N. S., and Mahmoudi, C. (2018). Fog computing conceptual model. Technical Report NIST Special Publication 500-325, National Institute of Standard and Technology.

IoT-Eclipse (2015a). IoT Eclipse - Kura. `https://eclipse.org/kura`. Last visited: 2020-04-21.

IoT-Eclipse (2015b). IoT Eclipse - Open Source for IoT. `http://iot.eclipse.org`. Last visited: 2017-05-15.

ITU-T (2009). E.800: Definitions of terms related to quality of service. Technical Report E.800, ITU.

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

Jin, J., Gubbi, J., Tie, L., and Palaniswami, M. (2012). Network architecture and QoS issues in the internet of things for a smart city. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 956–961, Gold Coast, Australia. IEEE.

Jiong, J., Gubbi, J., Marusic, S., and Palaniswami, M. (2014). An Information Framework for Creating a Smart City Through Internet of Things. *Internet of Things*, 1(2):112–121.

Keller, M., Robbert, C., and Karl, H. (2014). Template embedding: Using application architecture to allocate resources in distributed clouds. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 187–395, London, UK. IEEE.

Kirkpatrick, K. (2013). Software-Defined Networking. *Communications of the ACM*, 56(9):16–19.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Latora, V., Nicosia, V., and Russo, G. (2017). *Complex Networks: Principles, Methods and Applications*. Cambridge University Press, 1 edition.

Le, Q., Ngo-Quynh, T., and Magedanz, T. (2014). RPL-based multipath Routing Protocols for Internet of Things on Wireless Sensor Networks. In *2014 International Conference on Advanced Technologies for Communications (ATC 2014)*, pages 424–429, Hanoi, Vietnam. IEEE.

Lee, G., Jung-Soo, P., Kong, N., Crespi, N., and Chong, I. (2012). The Internet of Things - Concept and Problem Statement. Technical Report draft-lee-iot-problem-statement-05, IETF.

Lera, I. and Guerrero, C. (2019). YAFS Documentation - Release 3.0. `http://bit.do/yafsdoc`. Last visited: 2019-03-11.

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.

LinkSmart (2015). LinkSmart Middleware Platform. `http://iot.eclipse.org`. Last visited: 2017-04-15.

Liu, W., Li, H., and Huang, O. (2014). Service function chaining (sfc) general use cases. Technical Report draft-liu-sfc-use-cases-08, IETF.

Ma, N., Zhang, J., and Huang, T. (2017). A model based on genetic algorithm for service chain resource allocation in nfv. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 607–611, Chengdu, China. IEEE.

Mahmud, R., Ramamohanarao, K., and Buyya, R. (2018). Latency-aware application module management for fog computing environments. *ACM Trans. Internet Technol.*, 19(1):1–21.

Malik, A., Aziz, B., Adda, M., and Ke, C.-H. (2020). Smart routing: Towards proactive fault handling of software-defined networks. *Computer Networks*, 170(1):1389–1286.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computing — the business perspective. *Decision Support Systems*, 51(1):176–189.

Matias, J., Garay, J., Toledo, N., Unzilla, J., and Jacob, E. (2015). Toward an SDN-enabled NFV architecture. *Communications Magazine*, 53(4):187–193.

Mehboob, U., Qadir, J., Ali, S., and Vasilakos, A. (2016). Genetic algorithms in wireless networking: techniques, applications, and issues. *Soft Computing*, 20(6):2467–2501.

Mehraghdam, S., Keller, M., and Karl, H. (2014). Specifying and placing chains of virtual network functions. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 7–13, Luxembourg, Luxembourg. IEEE.

Montori, F., Bedogni, L., and Bononi, L. (2018). A collaborative internet of things architecture for smart cities and environmental monitoring. *IEEE Internet of Things Journal*, 5(2):592–605.

Mukherjee, M., Shu, L., and Wang, D. (2018). Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys & Tutorials*, 20(3):1826–1857.

Mulligan, G. (2007). The 6lowpan architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, pages 78–82, New York, USA. ACM.

Nakamura, H. (2016). Network functions virtualisation (nfv); reliability; report on models and features for end-to-end reliability. Technical Report DGS/NFV-REL003, ETSI Industry Specification Group (ISG).

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

Noghin, V. D. (2018). Edgeworth-pareto principle. In *Reduction of the Pareto Set: An Axiomatic Approach*, chapter 1, pages 1–22. Springer International Publishing.

OPENIoT (2015). OPENIoT - Open Source Cloud Solution for the Internet of Things. `http://openiot.eu`. Last visited: 2017-01-30.

Pahl, C. and Lee, B. (2015). Containers and clusters for edge cloud architectures – a technology review. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 379–386, Rome, Italy. IEEE.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys & Tutorials*, 15(3):1389–1406.

Papadimitriou, D. and Mannie, E. (2006). Analysis of Generalized Multi-Protocol Label Switching (GMPLS)-based Recovery Mechanisms (including Protection and Restoration). Technical Report RFC:4428, IETF.

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.

Pavković, B., Theoleyre, F., and Duda, A. (2011). Multipath opportunistic rpl routing over ieee 802.15.4. In *Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 179–186, New York, NY, USA. ACM.

Perez Abreu, D. and Velasquez, K. (2016). SusCity Ontology. `https://eden.dei.uc.pt/~dabreu/ontologies/iot_infrastructure.ttl`. Last visited: 2016-04-20.

Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2015). Resilience in iot infrastructure for smart cities. In *2015 Cloudification of the Internet of Things (CIoT)*, pages 1–4, Paris, France. IEEE.

Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2017a). 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.

Perez Abreu, D., Velasquez, K., Curado, M., and Monteiro, E. (2017b). A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, 72(1):19–30.

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.

Perez Abreu, D., Velasquez, K., Miranda Assis, M. R., 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.

Perez Abreu, D., Velasquez, K., Paquete, L., Curado, M., and Monteiro, E. (2020). Resilient embedding - GitLab Repository. `https://git.dei.uc.pt/dabreu/ResilientServiceChains.git`. Last visited: 2020-07-29.

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.

Perez Abreu, D., Velasquez, K., Pinto, A. M., Curado, M., and Monteiro, E. (2017c). 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.

Peter Mell, T. G. (2011). The NIST Definition of Cloud Computing. Technical Report SP 800-145, National Institute of Standards and Technology - NIST.

Pham, T. A. Q., Sanner, J.-M., Morin, C., and Hadjadj-Aoul, Y. (2020). Virtual network function–forwarding graph embedding: A genetic algorithm approach. *International Journal of Communication Systems*, 33(10):e4098.

Pioro, M. and Medhi, D. (2004). Restoration and protection design of resilient networks. In *Routing, Flow, and Capacity Design in Communication and Computer Networks*, chapter 9, pages 353–401. Elsevier.

Pöhls, H. C., Angelakis, V., Suppan, S., Fischer, K., Oikonomou, G., Tragos, E. Z., Rodriguez, R. D., and Mouroutis, T. (2014). RERUM: Building a reliable IoT upon privacy- and security- enabled smart objects. In *Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 122–127, Istanbul, Turkey. IEEE.

Pujolle, G. (2020). *Software Networks: Virtualization, SDN, 5G and Security.* Willey, 2 edition.

Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M., and Wu, D. O. (2020). Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, pages 1–28.

Quinn, P. and Nadeau, T. (2015). Problem statement for service function chaining. Technical Report RFC:7498, IETF.

Quittek, J. (2014). Network functions virtualisation (nfv); management and orchestration. Technical Report DGS/NFV-MAN001, ETSI Industry Specification Group (ISG).

Rahman, M. R. and Boutaba, R. (2013). Svne: Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management*, 10(2):105–118.

Rehman, A. U., Aguiar, R. L., and Barraca, J. P. (2019). Fault-tolerance in the scope of software-defined networking (sdn). *IEEE Access*, 7(1):124474 – 124490.

Reitblatt, M., Canini, M., Guha, A., and Foster, N. (2013). FatTire: Declarative fault tolerance for software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 109–114, New York, USA. ACM.

Renner, G. and Ekárt, A. (2003). Genetic algorithms in computer aided design. *Computer-Aided Design*, 35(8):709–726.

RERUM (2013). REliable, Resilient and secUre IoT for sMart city applications. `https://ict-rerum.eu`. Last visited: 2020-05-18.

ResearchGate (2019). CloudSim - ResearchGate community group. https://www.researchgate.net/topic/CloudSim. Accessed: 2019-02-10.

Ros, F. J. and Ruiz, P. M. (2014). Five nines of southbound reliability in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, pages 31–36, New York, USA. ACM.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23.

Schöller, M. and Khan, N. (2015). Network functions virtualisation (nfv); resiliency requirements. Technical Report DGS/NFV-REL001, ETSI Industry Specification Group (ISG).

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.

Sköldström, P., Sonkoly, B., Gulyás, A., Németh, F., Kind, M., Westphal, F.-J., John, W., Garay, J., Jacob, E., Jocha, D., Elek, J., Szabó, R., Tavernier, W., Agapiou, G., Manzalini, A., Rost, M., Sarrar, N., and Schmid, S. (2014). Towards Unified Programmability of Cloud and Carrier Infrastructure. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 55–60, London, UK. IEEE.

Sousa, B. (2013). *Multihomming Aware Optimization Mechanism*. PhD Thesis, University of Coimbra, LCT.

Souza, V. B., Masip-Bruin, X., Marín-Tordera, E., Ramírez, W., and Sánchez-López, S. (2017). Proactive vs reactive failure recovery assessment in combined fog-to-cloud (f2c) systems. In *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–5, Lund, Sweden. IEEE.

Souza, V. B. C., Ramírez, W., Masip-Bruin, X., Marín-Tordera, E., Ren, G., and Tashakor, G. (2016). Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–5, Kuala Lumpur, Malaysia. IEEE.

Stanford University (2015). Protégé. `http://protege.stanford.edu`. Last visited: 2016-02-28.

Stephens, B., Cox, A. L., and Rixner, S. (2013). Plinko: Building Provably Resilient Forwarding Tables. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pages 26:1–26:7, New York, USA. ACM.

Sterbenz, J. P. G., Çetinkaya, E. K., Hameed, M. A., Jabbar, A., Qian, S., and Rohrer, J. P. (2013). Evaluation of network resilience, survivability, and disruption tolerance: analysis, topology generation, simulation, and experimentation. *Telecommunication Systems*, 52(2):705–736.

Sun, L., Dong, H., and Ashraf, J. (2012). Survey of service description languages and their issues in cloud computing. In *2012 Eighth International Conference on Semantics, Knowledge and Grids*, pages 128–135, Beijing, China. IEEE.

SusCity-Project (2015). FCT - SusCity Project. `http://goo.gl/4WlgpC`. Last visited: 2020-09-19.

TUWIEN. Vienna University of Technology (2015). European Smart Cities. `http://www.smart-cities.eu`. Last visited: 2020-03-20.

Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H., and Smith, L. (2005). Intel virtualization technology. *Computer*, 38(5):48–56.

University of OXFORD Information Systems Group (2015). HermiT OWL Reasoner. `http://www.hermit-reasoner.com`. Last visited: 2016-02-29.

Velasquez, K., Perez Abreu, D., Assis, M. R., Senna, C., Aranha, D. F., Bittencourt, L. F., Laranjeiro, N., Curado, M., Vieira, M., Monteiro, E., et al. (2018). Fog orchestration for the internet of everything: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 9(1):1–23.

Velasquez, K., Perez Abreu, D., Curado, M., and Monteiro, E. (2015). Towards latency mitigation in emergency scenarios. In *2015 Cloudification of the Internet of Things (CIoT)*, pages 1–4, Paris, France. IEEE.

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.

Velasquez, K., Perez Abreu, D., Gonçalves, D., Bittencourt, L., 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.

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*, pages 64–72, Paris, France. IEEE.

Vugrin, E. D., Warren, D. E., Ehlen, M. A., and Camphouse, R. C. (2010). *A Framework for Assessing the Resilience of Infrastructure and Economic Systems*, chapter 5, pages 77–116. Springer, 1 edition.

W3C (2013). SPARQL. `https://www.w3.org/TR/rdf-sparql-query`. Last visited: 2016-03-20.

Wei, W., De, S., Toenjes, R., Reetz, E., and Moessner, K. (2012). A Comprehensive Ontology for Knowledge Representation in the Internet of Things. In *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1793–1798, Liverpol, England. IEEE.

Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., and Alexander, R. (2012). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Technical Report RFC:6550, IETF.

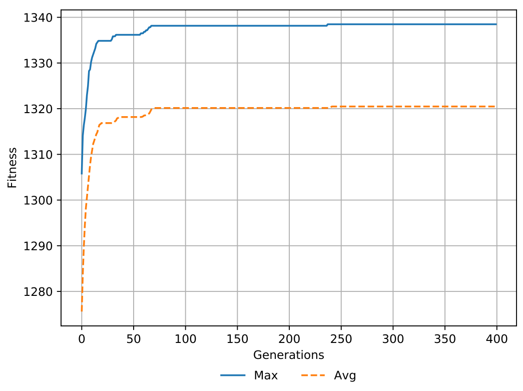
Wright, S., Hu, Y. C., and Reid, A. (2015). Network functions virtualisation (nfv); infrastructure overview. Technical Report DGS/NFV-INF001, ETSI Industry Specification Group (ISG).

Yi, B., Wang, X., Li, K., k. Das, S., and Huang, M. (2018). A comprehensive survey of network function virtualization. *Computer Networks*, 133(1):212–262.

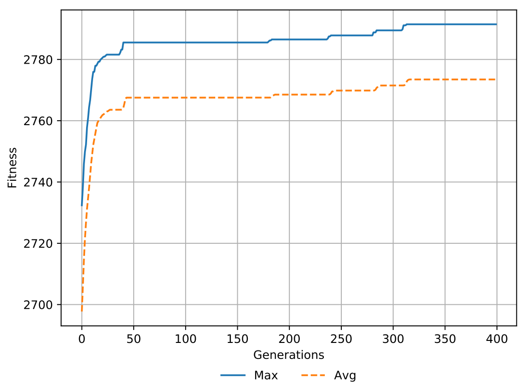
Appendixes

A Results - Fitness Function Values

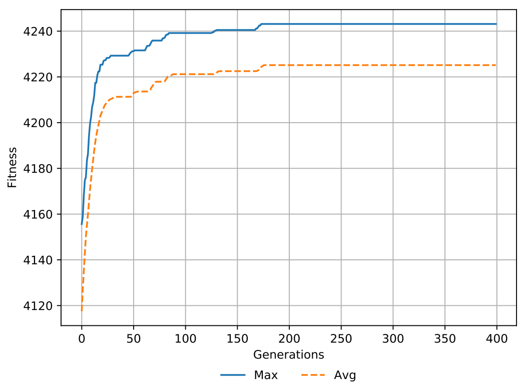
This Appendix presents the evolution of the fitness values for the tiny, small, medium, and large scenarios when the replication methods noReplicas and vsReplicas are used.



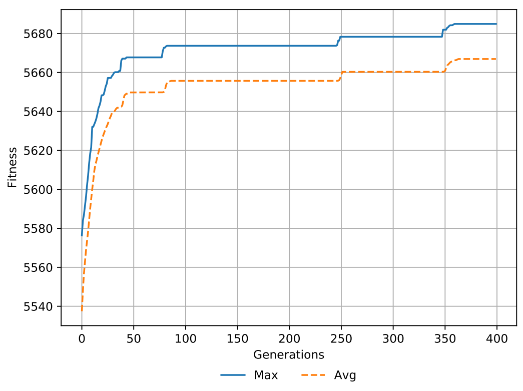
(a) Tiny Scenario.



(b) Small Scenario.

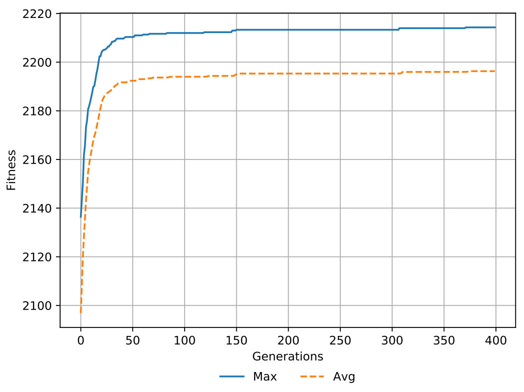


(c) Medium Scenario.

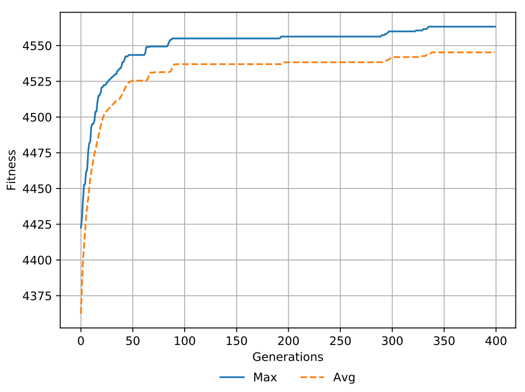


(d) Large Scenario.

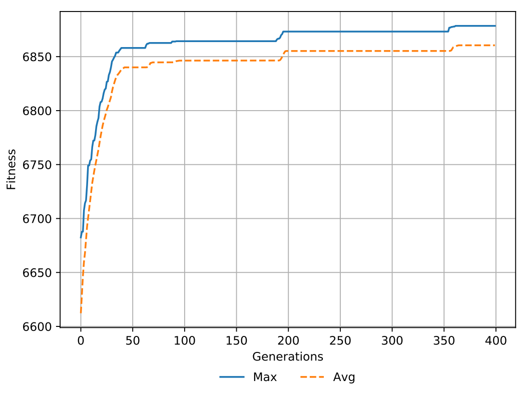
Figure A.1: Fitness Function Values by Generations for SC using noReplicas Replication in the Scenarios Evaluated.



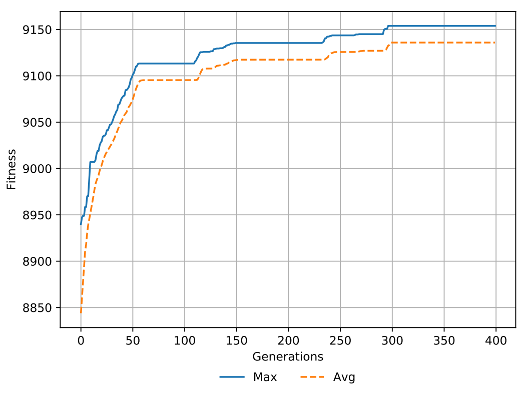
(a) Tiny Scenario.



(b) Small Scenario.



(c) Medium Scenario.



(d) Large Scenario.

Figure A.2: Fitness Function Values by Generations for SC using vsReplicas Replication in the Scenarios Evaluated.

B Results - Response Time

This Appendix presents the results of the response time experiments for the small and medium scenarios using the three replication methods, namely noReplicas, vsReplicas, and End2End.

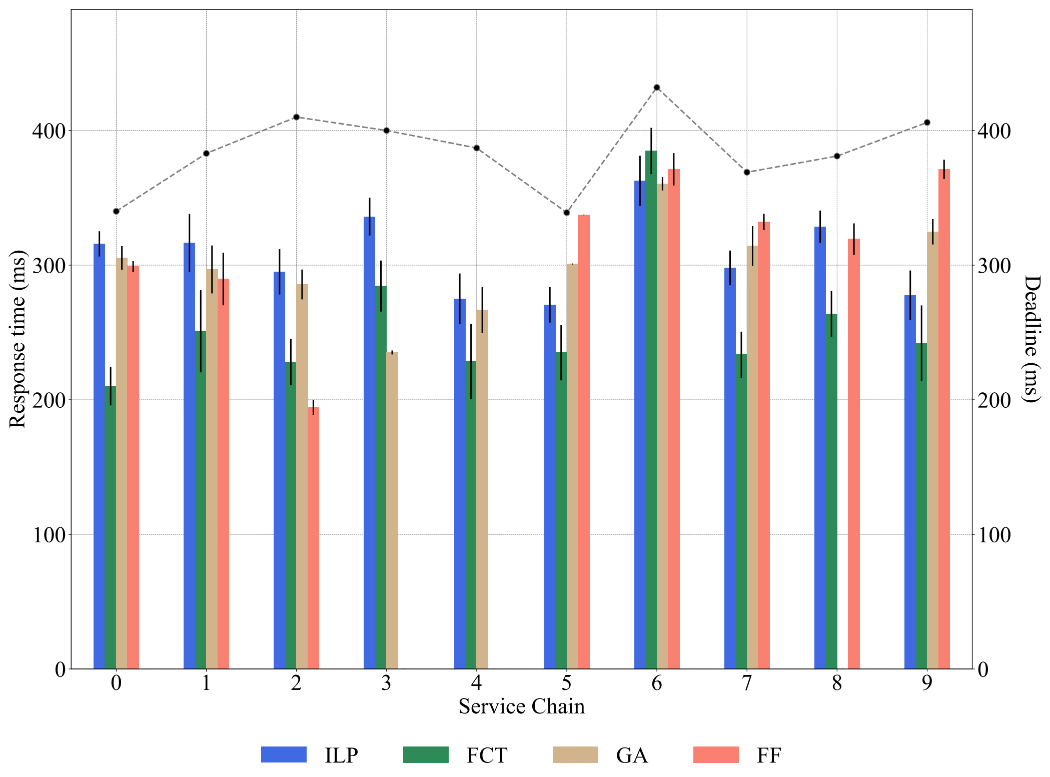


Figure B.1: SC Response Time - Small Scenario - noReplicas.

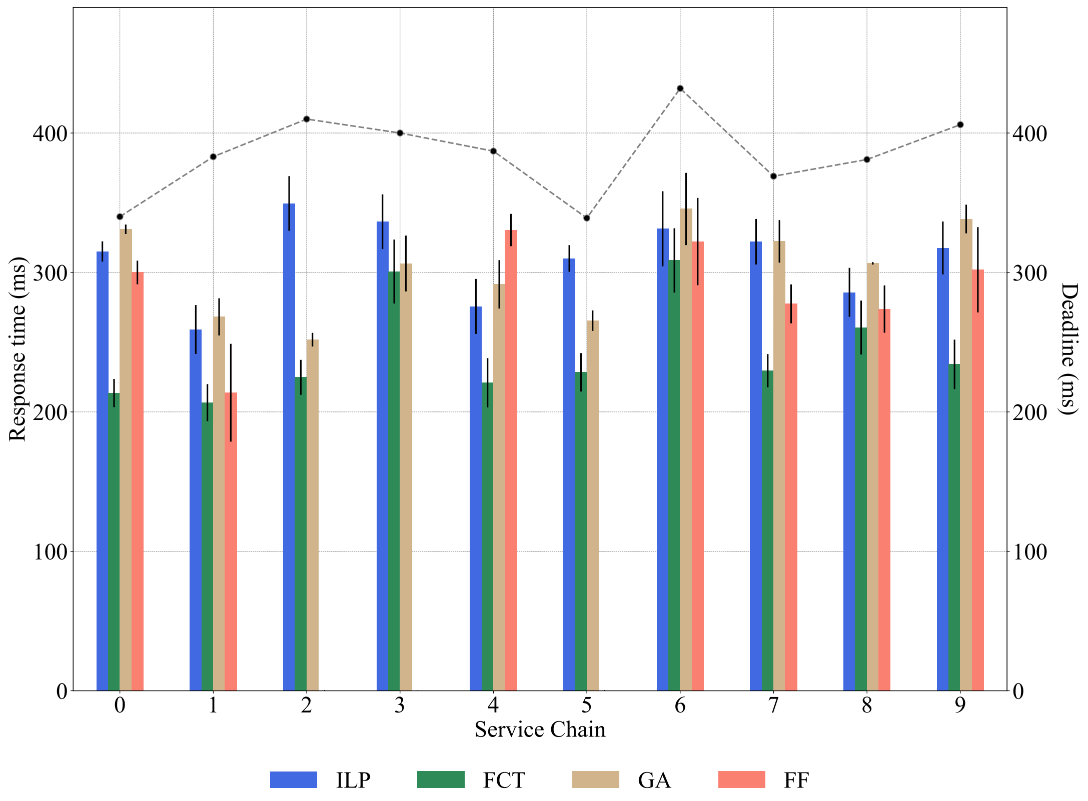


Figure B.2: SC Response Time - Small Scenario - vsReplicas.

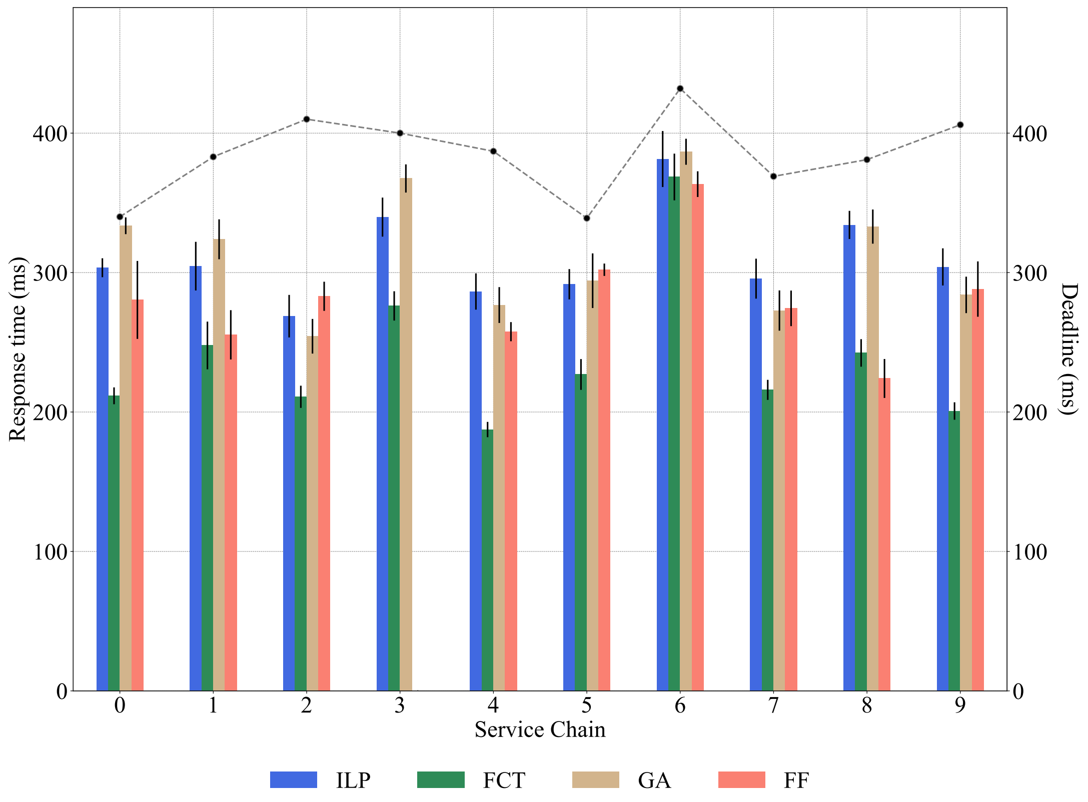


Figure B.3: SC Response Time - Small Scenario - End2End.

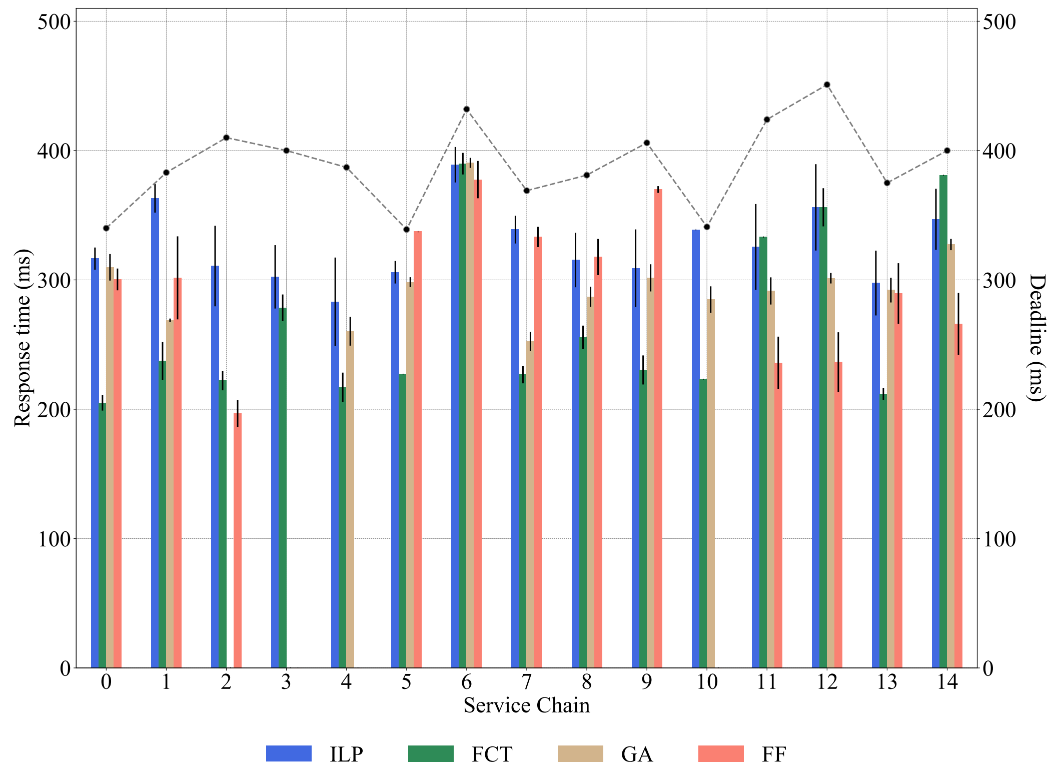


Figure B.4: SC Response Time - Medium Scenario - noReplicas.

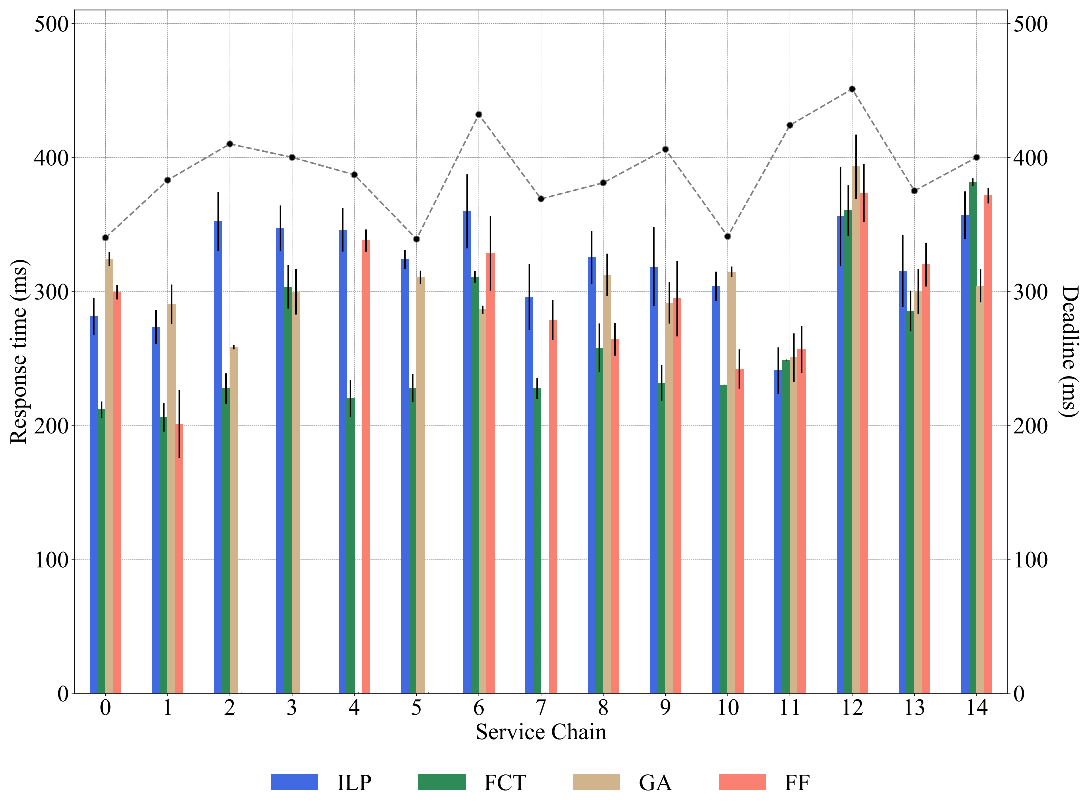


Figure B.5: SC Response Time - Medium Scenario - vsReplicas.

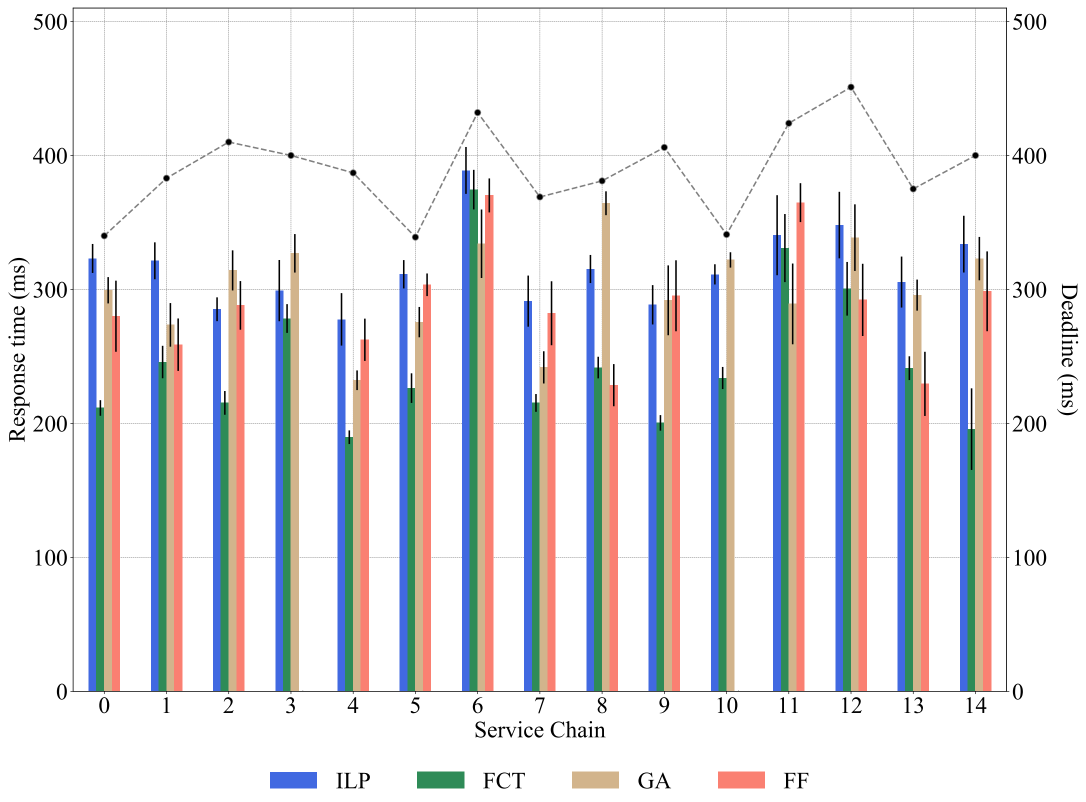


Figure B.6: SC Response Time - Medium Scenario - End2End.