UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Guto Leoni Santos

An artificial Intelligence Powered Framework for Automatic Service Function Chain Placement in Distributed Scenarios

Recife

2023

Guto Leoni Santos

An artificial Intelligence Powered Framework for Automatic Service Function Chain Placement in Distributed Scenarios

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Inteligência Computacional

Orientador (a): Judith Kelner

Coorientador (a): Patricia Takako Endo

Recife

2023

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

S237a Santos, Guto Leoni
An artificial intelligence powered framework for automatic service function chain placement in distributed scenarios / Guto Leoni Santos – 2023.
186 f.: il., fig., tab.

Orientador: Judith Kelner.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2023.
Inclui referências e apêndices.

1. Inteligência computacional. 2. Network function virtualisation. 3. Service function chain. 4. Gerenciamento de rede. 5. Rede distribuída. 6. Aprendizado de máquina. I. Kelner, Judith (orientador). II. Título

006.31 CDD (23. ed.) UFPE - CCEN 2023 – 77

Guto Leoni Santos

“An Artificial Intelligence Powered Framework for Automatic Service Function Chain Placement in Distributed Scenarios”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 22/03/2023.

_____

Orientadora: Profa. Dra. Judith Kelner

BANCA EXAMINADORA

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática / UFPE

_____

Prof. Dr. Edmundo Roberto Mauro Madeira
Instituto de Computação / UNICAMP

_____

Prof. Dr. Antonio Alfredo Ferreira Loureiro
Departamento de Ciência da Computação / UFMG

_____

Prof. Dr. Carmelo Jose Albanez Bastos Filho
Escola Politécnica de Pernambuco / UPE

I dedicate this work to all my friends. The people who always supported me and made me get this far. Thank you, I love you.

ACKNOWLEDGEMENTS

I would first like to thank God for giving me the strength, patience and resilience to complete this work at such a complicated and difficult time.

I would like to thank my parents, Marcos and Cicera, for always doing everything so that I could study and work with what I love. I am very grateful to them for always encouraging me to study and follow my dreams. Mom, thank you so much for always understanding me and giving me the best advice, always respecting my limits and being there for everything. Dad, thank you so much for always being with me, through the best and worst times, for encouraging me to be strong and keep going, despite everything. I love you guys more than anything in this world. I would also like to thank Naially Sabrine, the best sister anyone could ask for. She, with her maximum sincerity and delicacy of a horse's kick, was always with me, supporting and helping me. I love you so much.

I would like to thank my supervisor, Judith Kelner, for all her patience, advice and guidance during this work. I repeat, we managed to do a good job in one of the most difficult moments that humanity has ever gone through, and all of this was only possible thanks to her, who opened so many doors for me. I would also like to thank Professor Djamel Sadok, who always wanted to guide me too and gave me so much advice with his usual good humour.

Again, for the third time, a special thanks to Patricia Takako Endo. This person I was very lucky to meet while still an undergraduate and to work with for so many years. The person who first opened doors for me and always believed in me so much. Know that you are a reference to me and that I will never know how to thank you for everything you have already done for me. Thanks also to Professor Theo Lynn, who helped me in so many moments in this work.

I would like to thank Elaine Santos, my psychologist. She was essential to maintain my mental health during my PhD. Always supporting me and teaching me to learn more about myself and my limits. Congratulations on being an incredible professional, and thank you for everything.

I would like to thank all my friends. Thank goodness, the list is long, so it would be impossible to mention them all here. However, know that you make all the difference in my life, and if I finished this PhD it was because you helped me to deal with my problems and conflicts. Although I did most of this PhD alone in my room because of a damn pandemic, I would like to thank my laboratory friends who saw my moments of difficulties up close for

a few moments: Elisson Rocha, Diego Bezerra, Daniel Bezerra (they are not brothers), Iago Richard, Assis Tiago and Marrone Dantas. A very special thank you to Leylane Ferreira, my best friend and sister who has always been with me. She knows everything I've been through, and she's always been there helping and supporting me. Thank you for everything! I would like to thanks to all those who have passed in my life before this work, I have not forgotten you, and you have helped me to get here (see acknowledgements of my TCC and my master dissertation).

Finally, I would like to thank the Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco for funding this work through grant IBPG-0059-1.03/19.

"E o caminho da felicidade ainda existe, é uma trilha estreita, é em meio à selva triste."

Racionais MC's (DEEZER, 2023)

ABSTRACT

Software Defined Network (SDN) and Network Function Virtualisation (NFV) are making networks programmable and consequently much more flexible and agile. To meet service level agreements, achieve greater utilisation of legacy networks, faster service deployment, and reduce expenditure, telecommunications operators are deploying increasingly complex Service Function Chains (SFCs). Besides the advantages from service virtualisation, it is expected that network performance and availability do not be affected by SFC usage. However, several factors that may compromise the SFC availability are added in a virtualised scenario such as software failures, misconfiguration, cyberattacks, and so on. In order to mitigate the impact of these factors, redundancy mechanisms can be used, i.e., to add redundant Virtual Network Functions (VNFs) in the servers to keep the SFC operation in case of failures. On the other hand, the network operators desire, of course, to allocate the SFCs optimising the resources utilisation in order to reduce Operational Expenditures (OPEX), which is a challenge since the replication mechanisms demand additional computational resources. In addition, the placement of SFCs in distributed scenarios can improve their availability, since an isolated failure would not impact the whole SFC operation. However, the placement in geo-distributed scenarios increases the management complexity, where different hardware and additional delay may compromise the SFC performance. Therefore, intelligent strategies are needed to optimise the SFC placement. This thesis presents the Sfc Placement framework focused on avallability for DistributEd scenaRios (SPIDER), a framework for SFC placement with focus on distributed scenarios and SFC availability. SPIDER is designed to make SFC placement in different distributed scenarios, i.e., scenarios with different hardware and software characteristics. To do that, SPIDER uses context information in order to define the SFC placement strategy. In addition, machine learning techniques are used to predict the traffic of allocated SFCs and reinforcement learning to select the servers for the SFC placement. We compare the performance of LSTM and GRU models to predict traffic using a real dataset of cellular network. In order to define the placement of an SFC request, we proposed a reinforcement learning based algorithm to select the suitable candidate node and define the redundancy strategy to meet availability requirements. We implemented a proof-of-concept of SPIDER in order to show the feasibility of the framework. We implemented the SFCs using containers and Kubernetes to manage them. We assess the framework by assessing the placement time for SFCs with

different numbers of VNFs. In order to evaluate the SFCs placed, we also evaluate the SFC delay for a centralized and a distributed scenario.

Keywords: network function virtualisation; service function chain; network management; distributed network; machine learning; deep learning; reinforcement learning.

RESUMO

Software Defined Networks (SDN) e *Network Function Virtualisation* (NFV) estão tornando as redes programáveis e, consequentemente, muito mais flexíveis e ágeis. Para cumprir acordos de nível de serviço, obter maior utilização de redes legadas, implantação de serviço mais rápida e reduzir despesas, as operadoras de telecomunicações estão implantando *Service Function Chains* (SFCs) cada vez mais complexas. Apesar dos benefícios das SFCs, o aumento da heterogeneidade e do dinamismo da computação em nuvem para a computação em borda apresenta desafios significativos de posicionamento de SFC, não menos importante, adicionando ou removendo funções de rede, mantendo a disponibilidade, qualidade de serviço e minimizando custos. Além das vantagens da virtualização de serviços, espera-se que o desempenho e a disponibilidade da rede não sejam afetados pelo uso de SFCs. No entanto, vários fatores que podem comprometer a disponibilidade da SFC são adicionados em um cenário virtualizado, como falhas de software, errors de configuração, ataques cibernéticos entre outros. Para mitigar o impacto desses fatores, mecanismos de redundância podem ser utilizados, ou seja, adicionar *Virtual Network Functions* (VNFs) redundantes nos servidores para manter o funcionamento da SFC em caso de falhas. Por outro lado, as operadoras de rede desejam, obviamente, alocar as SFCs otimizando a utilização dos recursos de forma a reduzir os Gastos Operacionais (OPEX), o que é um desafio visto que os mecanismos de replicação demandam recursos computacionais adicionais. Além disso, a alocação de SFCs em cenários distribuídos pode melhorar sua disponibilidade, pois uma falha isolada não impactaria no funcionamento da SFC como um todo. No entanto, a alocação em cenários geo-distribuídos aumenta a complexidade de gerenciamento, onde diferentes hardwares e atrasos adicionais podem comprometer o desempenho da SFC. Portanto, estratégias inteligentes são necessárias para otimizar o posicionamento do SFC. Esta tese apresenta o *Sfc Placement framework focused on avallability for DistributEd scenaRios* (SPIDER), um *framework* para posicionamento de SFC com foco em cenários distribuídos e disponibilidade dessa SFC. O SPIDER foi projetado para fazer a alocação da SFC em diferentes cenários distribuídos, ou seja, cenários com diferentes características de hardware e software. Para fazer isso, o SPIDER usa informações contextuais para definir a estratégia de alocação da SFC. Além disso, técnicas de aprendizado de máquina são usadas para prever o tráfego das SFCs alocadas e aprendizado de reforço para selecionar os servidores para a alocação da SFC. Comparamos o desempenho dos modelos LSTM e GRU para prever o tráfego usando um conjunto de dados real de uma rede celular. Para definir

o alocação de uma requisição SFC, propusemos um algoritmo baseado em aprendizado por reforço para selecionar o nó candidato adequado e definir a estratégia de redundância para atender aos requisitos de disponibilidade. Implementamos uma prova de conceito do SPIDER para mostrar a viabilidade do framework. Implementamos as SFCs utilizando containers e Kubernetes para gerenciá-los. Avaliamos o framework avaliando o tempo de colocação de SFCs com diferentes números de VNFs. Para avaliar os SFCs alocadas, também avaliamos o atraso das SFCs para um cenário centralizado e um distribuído.

Palavras-chave: *network function virtualisation*; *service function chain*; gerenciamento de rede; rede distribuída; aprendizado de máquina; aprendizado profundo; aprendizado por reforço.

LIST OF FIGURES

Figure 1 – Stages of SFC orchestration. 30
Figure 2 – SFC placement example 31
Figure 3 – SPN Components 34
Figure 4 – Example of an SPN model to represent the availability of a generic component 34
Figure 5 – MAPE-K loop. 36
Figure 6 – Example of a LSTM block 39
Figure 7 – Actor-critic method 43
Figure 8 – A2C algorithm schema 45
Figure 9 – Example of SFC system operation 58
Figure 10 – Example of generic SFC request. 59
Figure 11 – SPIDER Framework overview. 61
Figure 12 – Data Monitor module and daemon. 69
Figure 13 – SPIDER core overview. 70
Figure 14 – Agent module description 72
Figure 15 – JSON examples for the Traffic Prediction module 73
Figure 16 – The Milan Metropolitan Area. 76
Figure 17 – Number of Internet activities of cells 1 and 1000 in the Telecom Italia dataset. 78
Figure 18 – Cell internet traffic prediction pipeline. 79
Figure 19 – Elbow method results. 80
Figure 20 – Overlay of the 12 clusters on a map of the metropolitan area of Milan. . . 82
Figure 21 – Mean RMSE of LSTM model for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12. 84
Figure 22 – Average RMSE of GRU model for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12. 86
Figure 23 – Box plot of the RMSE of the best LSTM configurations for Cluster 12. . . 87
Figure 24 – Boxplot of the RMSE of the best configurations of (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 5, (e) cluster 6, (f) cluster 7, (g) cluster 8, and (h) cluster 9. 88

Figure 25 – Comparison against ground truth Internet activity and the predictions of LSTM and GRU models for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12. 90
Figure 26 – Placement examples of VNFs of the same type 94
Figure 27 – Placement examples of SFCs with two VNF types 95
Figure 28 – Generation of SPN models based on the SFC placement 97
Figure 29 – Generation of SPN models with different VNFs types placed in shared servers 98
Figure 30 – Sample physical infrastructure and associated graph. 102
Figure 31 – SFC request example. 102
Figure 32 – SFC request represented as a directed graph. 103
Figure 33 – SFC placement represented as a graph matching. 104
Figure 34 – Selection of candidate nodes. 110
Figure 35 – SFC Request represented as an MDP. 113
Figure 36 – The availability difference impact on the reward. 118
Figure 37 – Example of candidate nodes selection in Cand-RL algorithm. 122
Figure 38 – AS graph with 50 nodes. 125
Figure 39 – Parametrization results of Cand-RL algorithm using a PPO agent. 128
Figure 40 – Acceptance rate results for different numbers of customers. 130
Figure 41 – Acceptance rate results for different availability requirements. 131
Figure 42 – Comparison about SFC aspects. 132
Figure 43 – Basic Kubernetes concepts. 136
Figure 44 – Traffic flow configuration. 141
Figure 45 – SFC request sequence diagram. 142
Figure 46 – The face detection application represented as an SFC. 142
Figure 47 – Network topology considered for the experiments. 144
Figure 48 – Scenarios considered in the experiments. 145
Figure 49 – Placement runtime results. 146
Figure 50 – VNF processing runtime results. 147
Figure 51 – SFC placement scenarios 175

LIST OF TABLES

Table 1 – Comparison among the related works (Y-yes, N-no, and *-not specified in the work) . . . 54
Table 2 – SPIDER Requirements. . . . 56
Table 3 – Repository scripts. . . . 61
Table 4 – Physical node data model. . . . 62
Table 5 – Physical link data model. . . . 64
Table 6 – Infrastructure data model. . . . 64
Table 7 – SFC request data model. . . . 65
Table 8 – VNF template data model. . . . 65
Table 9 – SFC traffic data model. . . . 66
Table 10 – API endpoints. . . . 67
Table 11 – Periods of the day. . . . 80
Table 12 – Grid Search parameters and levels. . . . 81
Table 13 – Parameters used to train the DL models. . . . 82
Table 14 – Comparison of the LSTM, GRU, Random Forest, and Decision Tree models. 89
Table 15 – Summary of parameters used in the model. . . . 100
Table 16 – Simulation parameters about the physical infrastructure. . . . 124
Table 17 – VNF types. . . . 126
Table 18 – Angular coefficient and variance results of the PPO algorithm for different parameter configurations. . . . 129
Table 19 – Network configuration of servers in Scenario 2. . . . 148
Table 20 – Communication SFC delay results (in seconds). . . . 148
Table 21 – Overall SFC delay results. . . . 149
Table 22 – Scientific papers produced related to this thesis. . . . 156
Table 23 – Parameters of different VNF types 174
Table 24 – Availability, downtime, and placement cost results for all scenarios 176
Table 25 – Runtime comparison of baseline SPN models and proposed algorithm 177

LIST OF ABBREVIATIONS AND ACRONYMS

A2C	Advantage Actor-Critic
AS	Authonous System
CAPEX	Capital Expenditure
CDR	Call Detail Record
CNN	Convolutional Neural Network
CSDL	Cloud Service Declarative Definition Language
CTMC	Continuous Time Markov Chain
DL	Deep Learning
DPI	Deep Packet Inspection
DRL	Deep Reinforcement Learning
ETSI	European Telecommunications Standards Institute
GRU	Gated Recurrent Unit
IaaS	Infrastructure as a Service
IDS	Intrusion Detection Systems
ILP	Integer Linear Programming
IoT	Internet of Things
ISP	Internet Service Provider
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MANO	Management and Orchestration
MDP	Markov Decision Process
MEC	Mobile Edge Computing
MILP	Mixed-Integer Linear Programming
ML	Machine Learning
MLP	Multi Layer Perceptron

MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
NAT	Network Address Translation
NDA	Non-disclosure Agreement
NFV	Network Function Virtualisation
NSD	Network Service Description
ONOS	Open Network Operating System
OPEX	Operational Expenditure
PPO	Proximal Policy Optimisation
QoS	Quality of Service
RBM	Restricted Boltzmann Machine
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SDN	Software Defined Network
SFC	Service Function Chain
SLA	Service Level Agreement
SMS	Short Message Service
SPN	Stochastic Petri Net
SVR	Support Vector Regression
TOs	telecommunications operators
TOSCA	Topology and Orchestration Specification for Cloud Applications
USDL	Unified Service Description Language
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
WSDL	Web Service Description Language

CONTENTS

1 INTRODUCTION 20
1.1 OBJECTIVES 25
1.2 CONTRIBUTIONS OF THE THESIS 25
1.3 ORGANISATION OF THE THESIS 27
2 BACKGROUND 29
2.1 SFC PLACEMENT 29
2.2 AVAILABILITY CONCEPTS 32
2.3 MAPE-K 35
2.4 DEEP LEARNING 37
2.4.1 Recurrent Neural Networks 38
2.4.1.1 Long Short-Term Memory Networks 38
2.4.1.2 Gated Recurrent Unit 40
2.5 REINFORCEMENT LEARNING 41
2.5.1 Advantage Actor-Critic 44
2.5.2 Proximal Policy Optimisation 45
2.6 CONCLUDING REMARKS 46
3 RELATED WORKS 47
3.1 FRAMEWORK FOR SFC PLACEMENT 47
3.2 COMPARISON 52
4 SPIDER 55
4.1 SPIDER REQUIREMENTS 55
4.2 SFC REQUEST EXAMPLE 58
4.3 SPIDER OVERVIEW 60
4.3.1 Repositories and Data Models 61
4.3.2 SPIDER API 66
4.3.3 SPIDER Core 68
4.3.4 Agent Module of SPIDER 72
5 TRAFFIC PREDICTION FOR SFC PLACEMENT 76
5.1 DATASET 76
5.2 TRAFFIC PREDICTION PIPELINE 77

5.3 CLUSTERING THE CELLS . . . 79
5.4 DL MODEL CONFIGURATION . . . 81
5.5 METRICS FOR EVALUATING DL MODELS . . . 83
5.6 RESULTS . . . 83
5.6.1 Statistical analysis . . . 85
5.6.2 Comparison of LSTM and GRU models. . . . 87
5.7 CONCLUDING REMARKS . . . 92
6 REINFORCEMENT LEARNING FOR SFC PLACEMENT . . . 93
6.1 MODELLING SFC PLACEMENT . . . 93
6.1.1 Generating SFC Availability models Automatically . . . 95
6.2 SYSTEM MODEL FOR SFC PLACEMENT PROBLEM . . . 99
6.2.1 Problem Definition . . . 105
6.3 THE CAND ALGORITHM FOR SELECTING CANDIDATE NODES . . . 107
6.3.1 An Illustrative Example of How the Cand Algorithm Works . . . 109
6.4 RL FOR SFC PLACEMENT . . . 111
6.4.1 Characteristics of SFC requests . . . 112
6.4.2 Environment State Representation . . . 114
6.4.3 Action representation . . . 115
6.4.4 Reward function . . . 116
6.5 THE CAND-RL ALGORITHM . . . 119
6.6 EVALUATION . . . 123
6.6.1 Simulation setup . . . 123
6.6.2 RL Agent Parametrization . . . 126
6.6.3 Scenario Variation . . . 129
6.7 CONCLUDING REMARKS . . . 134
7 SPIDER PROOF OF CONCEPT . . . 135
7.1 CONTAINER-BASED SFCS USING KUBERNETES . . . 135
7.2 PROTOTYPE EVALUATION . . . 142
7.2.1 Scenario Setup . . . 143
7.2.2 Placement Time . . . 144
7.2.3 VNF processing Runtime . . . 146
7.2.4 Communication SFC Delay . . . 147
7.2.5 Overall SFC delay . . . 149

7.3 CONCLUDING REMARKS 150
8 GENERAL CONSIDERATIONS 151
8.1 LIMITATIONS 153
8.2 SCIENTIFIC CONTRIBUTIONS 154
8.3 FUTURE WORKS 154
REFERENCES 157
APPENDIX A – SFC AVAILABILITY ANALYSIS 174
APPENDIX B – SFC REQUEST DETAILS 179
APPENDIX C – DAEMON CONFIGURATION 182
APPENDIX D – SFC PLACEMENT DECISION BY THE AGENT 185

1 INTRODUCTION

Historically, telecommunications operators (TOs) have deployed special purpose, network-specific, fixed-function hardware with standardised protocols. This hardware-centric approach often relied on fragmented, non-commodity hardware with physical installs for each appliance. As a result, innovation and competition was constrained by both the development of, access to, and maintenance of hardware development (JIM, 2015). The rapid emergence and adoption of the so-called *third IT platform* (GENS, 2013), a convergence of mobile technology, cloud computing, big data analytics, and social media, by both enterprises and consumers has resulted in dramatic shifts in bandwidth demand, network infrastructure requirements, and associated economic models that the traditional to approach cannot scale or adapt to easily. Against the backdrop increasing competition, TOs needed new ways to achieve greater legacy infrastructure utilisation, faster service deployment and new service time-to-market, improved greater agility and flexibility, new revenue streams, while at the same time reducing both Operational Expenditure (OPEX) and Capital Expenditure (CAPEX) (LYNN et al., 2018).

Software Defined Network (SDN) and Network Function Virtualisation (NFV) abstract the implementation of new network functions and decouple them from the hardware infrastructure and associated topological constraints, thus making networks programmable and as a result much more flexible and agile (RAY; KUMAR, 2021). The European Telecommunications Standards Institute (ETSI) defines NFV as the

> *"implementation of network functions in software that can run on a range of industry standard server hardware, and that can be moved to, or instantiated in, various locations in the network as required, without the need for installation of new equipment"* (INSTITUTE, 2018).

By virtualising network functions while retaining the same capabilities as the corresponding physical instances, multiple virtualised network functions can share physical hardware in the form of Virtual Machines (VMs), a concept that can be massively scaled to encompass large volumes of physical hardware (GROUP, 2013). As a result, TOs need no longer limit themselves to special-purpose hardware and can achieve the same, if not more, capacity with commercial off the shelf commoditised equipment thus reducing both CAPEX and OPEX. Consequently, TOs can benefit from reduced total cost of ownership and operational complexity, real-time bandwidth and network scalability, reduced service deployment and time to market, increased security, and improved visibility and integration (LYNN et al., 2018). Unsurprisingly, there are

a myriad of virtual network functions commonly used by TOs. These include firewalls, load balancing, Intrusion Detection Systems (IDS), Network Address Translation (NAT), amongst others (BHAMARE et al., 2016; KAUR; MANGAT; KUMAR, 2020).

A Service Function Chain (SFC) is defined as one or more Virtual Network Functions (VNFs) in an ordered sequence and subsequent steering of flows through them to provide end-to-end services (KAUR; MANGAT; KUMAR, 2020). The implementation of an SFC typically comprises four stages (MIRJALILY; ZHIQUAN, 2018):

1. Description – details the functional and non-functional properties of the network services including interfaces and constraints;

2. Composition – defines the order of network services to compose a functional service. Although these services can work independently, some network functions require a sequenced order to work properly;

3. Placement - determines in which physical nodes of the infrastructure the virtual functions will be deployed. This stage is essential in the pipeline because the physical resources must be managed optimally in terms of usage to avoid overloading some servers or wasting resources; and

4. Scheduling - defines the time an SFC will be deployed in the infrastructure and the time it will be removed, releasing resources to other deployed SFCs. The scheduling stage is important to minimise the whole execution time of the network services.

While SFCs have many advantages, they are not without challenges. In the same way that TOs must meet the Service Level Agreement (SLA) with their customers, SFCs must meet the policy constraints under which the SLA demands including network capacity and acceptable total latency for end users. These challenges are exacerbated by the increasing dynamism and heterogeneity of modern network environments. Thus there is a need for robust, well-tested, dynamic and automatic SFC models, and associated research (BHAMARE et al., 2016). Service reliability and availability have been the hallmark for trustworthy computing since the turn of the century (MUNDIE et al., 2002). They are the cornerstone of modern SLAs and failure to meet Quality of Service (QoS) can result in significant financial penalties to TOs as well as lost customers and reputational damage. In dynamic network environments, the dynamic addition or removal of network functions must be executed in such a way that it does not interfere with

the availability and quality of network services while at the same time minimising CAPEX and OPEX.

As highlighted by (XIE et al., 2016), there are many effort for SFC management and orchestration in centralised scenarios, such as traditional data centers. However, the SFC centralised solution can be not suitable in many situations such as multiple providers and multiple administrative domains. In addition, centralised infrastructures suffer from the scalability problem, which is a desired requirement for large network infrastructures. A natural improvement is to adopt a distributed architecture for the SFC placement, such as federated and/or Geo-distributed cloud. Considering the context of cellular networks, where a shorter communication delay is increasingly expected, the Mobile Edge Computing (MEC) paradigm can be applied in combination with the NFV paradigm (PANDEY et al., 2021). Therefore, the VNFs can be allocated in a distributed manner at the different levels of the network (from the edge to the cloud) meeting the delay and computation requirements. However, the management of this architecture and the SFC placement in these scenarios is a very complex task, due to scenario scale and constraints required by various applications (ESPOSITO, 2017). The complexity of SFC placement increases due to the different traffic conditions of different networks and the increased communication delay related to geographic distribution (ABU-LEBDEH et al., 2017).

Besides all challenges previously mentioned, with the adoption of NFV paradigm, obviously the availability of the SFC is expected to be guaranteed (MOUALLA; TURLETTI; SAUCEZ, 2018) (WANG et al., 2021a). Indeed, as the NFV paradigm relies on virtualisation for sharing physical network and computing resources, to guarantee the downtime avoid and failure recovery is important in order to respect SLA and the contracts established between consumers and providers (SOUALAH et al., 2017). Due to unplanned interruptions, the service providers can pay penalties because of SLA violations, while the costumers experience will degrade. For instance, unplanned interruptions can be translated into about $336,000 per hour in lost revenue for companies such as Microsoft and Amazon. Other more critical example is credit-card authorisation service, where a disruption can result in losses about $2.6 million per hour in transactions that can't be completed (ENDO et al., 2017).

Replication mechanism is usually used in order to mitigate failures and improve the availability of computer systems (CARPIO; BZIUK; JUKAN, 2017; ATTAR; RAISSI; KHALILI-DAMGHANI, 2017; KAYEDPOUR et al., 2017; ALAMDARI; SHARIFI, 2020). The VNFs from an SFC can be replicated in different nodes in order to decrease the failure probability of the whole SFC. However, the SFC placement problem is NP-hard (SUN et al., 2020) and the problem complexity

increases when the SFC must be deployed in a distributed scenario, due to the heterogeneity of computing resources and the common link delay (CAI et al., 2020); and ensuring the availability of the SFC, as adding replicas to increase availability can unnecessarily increase resource consumption, impacting the allocation cost and energy consumption (LIRA et al., 2019).

It is desired that the placement and orchestration of SFCs be done automatically or with minimum human intervention, which can be named as zero-touch service deployment and operation (RECSE; SZABO; NEMETH, 2020). However, the heterogeneity of NFV environment about the virtualisation stacks and the placement options need to be considered (LINGUAGLOSSA et al., 2019). These aspects increase the management complexity, resulting in a huge parameter space, "*rendering many traditional optimisation techniques either incompatible with the strict runtime constraints or entirely inapplicable due to hardware limitations*" (LANGE et al., 2020). In this way, the SFC placement problem can be modelled as a mathematical problem and solved by using computational techniques. Machine Learning (ML), heuristics, metaheuristics among other techniques can be applied for the autonomous network management, where several tasks can be done such as resource allocation, configuration optimisation, and traffic prediction (BOUTABA et al., 2018) (MASOUDI; GHAFFARI, 2016). These techniques can be applied for the network management in distributed scenarios, with a focus on service availability. These techniques can be used to define the replication configuration in order to mitigate the failures and downtime of distributed network infrastructure. For the point of view of customer, its SLA will be met, while for the point of view of network manager, the near optimal placement will be made, reducing CAPEX and OPEX.

Against this backdrop, in this thesis, we consider as the main research question: "*How to allocate and orchestrate SFCs in distributed scenarios, meeting availability requirements defined by the customers, in an automatic way?*". We propose the SFC Placement framework focused on avaIlability for DistributEd scenaRios (SPIDER), which considers the customers' requirements and a variety of information about the substrate network in order to choose the best SFC placement, focusing on how to optimise the SFC availability. SPIDER monitors information about the network environment, in order to decide the best strategy for the SFC placement. Then, when a customer sends an SFC request, SPIDER retrieves the relevant information, takes into account the customer requirements and decide the more appropriated SFC placement. As mentioned above, different techniques can be used for SFC placement and orchestration, as well as implemented in SPIDER. However, for the experiments and the implementation of the SPIDER in this work, we explored the usage of artificial intelligence

techniques for automatic SFC placement. Specifically, we used Deep Learning (DL) models and Reinforcement Learning (RL) agents combined to process the context information monitored from the network infrastructure to define the SFC placement. In our studies, we implemented an RL-based algorithm to define the SFC placement that meets the availability requirement defined by the customer, while the placement cost and energy consumption are minimized, in order to reduce OPEX. In addition, the SPIDER takes into account the computational and network requirements for the VNF and virtual links of the SFC.

We considered the following research questions to guide the development of this thesis:

- **How to allocate SFCs in distributed scenarios efficiently?** To answer this question, we performed a literature review to identify works that propose solutions for allocating SFCs in distributed scenarios. Although the focus of this thesis is to consider the availability of SFC, we will not limit our searches to just that topic, since other solutions, with different placement objectives, can contribute to the development of the framework proposed in this thesis.

- **How can contextual information about the infrastructure be organised and stored to enable quick consultation and updating?** Since we will have to deal with contextual information about the infrastructure to define better SFC placement strategies, we defined data models that are simple and extensible to be used in the proposed framework. The data models the physical and logic components considered in the SPIDER: physical nodes, physical nodes, physical links, VNFs, virtual links, SFC, among others. We used NoSQL schema in order to be easy to update the data schema, and due to the good performance of NoSQL databases.

- **How can computational algorithms be used to provide intelligent placement strategies, with a focus on SFC availability?** To process the contextual information of the infrastructure and create SFC placement strategies automatically, we use computational algorithms. This is due to the fact that such algorithms are able to learn how to perform such tasks, without the need to define deterministic placement strategies previously.

- **How effective can an SFC availability-focused solution be?** SFC availability can be achieved in different ways, however other metrics need to be taken into account. Thus, to evaluate the performance of the framework in the role of defining placement

strategies, as well as in the allocation process, we carried out experiments with simulation and prototype.

1.1 OBJECTIVES

Despite the benefits of the NFV paradigm, it cannot be assumed that the usage of SFC will not impact the network service performance and availability. Solutions for the SFC placement and orchestration need to consider the main customer requirements as well as the current network status, in order to provide a placement solution that does not violate the SLA. In addition, in order to reduce OPEX, it is expected that SFC placement will be done in an automated manner, with as little human intervention as possible.

Against this backdrop, the main objective of this thesis is to propose a framework, called SPIDER, which considers a set of context information about the infrastructure combined with computational algorithms to define an SFC placement solution. The definition of the solution takes into account the availability requirement of the user and the current information about the infrastructure. Then, a placement solution is created by the SPIDER in order to meet the availability requirements without violating computational requirements, while considering placement cost and the energy consumption of the SFC placement.

In order to validate the solution proposed, we evaluate the SPIDER by its modules as well as the whole SPIDER in different studies. The specific objectives of this thesis proposal are:

- To define algorithms which can be used for the SFC placement task;
- To specify data models to represent the context information as internal information of the SPIDER;
- To define automatic placement solution for the SFC placement with focus on the SFC availability; and
- To assess the SPIDER performance in different application scenarios.

1.2 CONTRIBUTIONS OF THE THESIS

The main contributions of this thesis are:

- We proposed the SPIDER framework for the SFC placement (Chapter 4). We present the framework architecture, defining the main modules and how they interact with each other. We detail how we model the infrastructure data and the SFC data are modelled in the framework. We also present the monitoring module to collect data and store it into the database. We detail how the context information is processed by an agent module by using computational algorithms to define the SFC placement. Finally, we present how the placement decision is processed and the SFC is placed in the infrastructure.

- In order to define the future status of the physical links, we predict the traffic about the SFCs already placed in the infrastructure. We proposed DL models to perform the traffic prediction (Chapter 5). We use real datasets about cellular networks to represent the traffic of SFCs. We propose different model architectures and compare their performance in order to identify the model which provides the lowest error prediction.

- We propose the Cand-RL algorithm, which uses RL to select the suitable candidate node and define the redundancy strategy to meet availability requirements (Chapter 6). The main advantage of using RL to create the placement solution is to find the near optimal solution automatically, thus dispensing with the need of human experience or labelled training datasets (SUN et al., 2020). Since RL is based on a trial-and-error learning approach (QIANG; ZHONGLI, 2011), we create an agent that learns how to create placement strategies to meet availability requirements by interacting with the network equipment and taking into account placement cost and energy consumption. The Cand-RL algorithm is used in the SPIDER in the module that defines the SFC placement based on the infrastructure status and the SFC request.

- In order to show the feasibility of SPIDER, we propose a proof-of-concept and implement it in a real scenario using virtual machines (Chapter 7). Based on the work presented in (SANTOS et al., 2020b), we implement a container-based SFC by using Kubernetes. We show how the VNFs can be created and connected using Kubernetes. We also perform experiments in order to assess the SPIDER performance in terms of placement time as well as the delay of the SFCs created in the infrastructure, comparing a centralized and a distributed scenario.

1.3 ORGANISATION OF THE THESIS

This thesis proposal handle the challenges of SFC placement in distributed scenarios and present the SPIDER, which is a framework for automatic placement based on context information and artificial intelligence algorithms. The remainder of this document is organised as follows:

Chapter 2 presents some basic concepts needed for the reader understand the proposal of this document. Concepts about SFC placement, availability modelling, DL, and RL are explained.

Chapter 3 presents the related works and a brief qualitative comparison between our proposal and other works from the literature.

Chapter 4 presents the SPIDER, which is the main contribution of the thesis. We present the data models, detailing the main context information considered in the framework. Afterwards, we detail the SPIDER API, describing the endpoints and how the SPIDER functions can be accessed. We also describe the main modules of SPIDER, and how they interact to process an SFC request, collecting context information from the infrastructure and using ML models to define the best SFC placement.

In Chapter 5, we present DL models proposed for the traffic prediction. We used these models in the framework to map the virtual links into physical links, with the purpose to compose the virtual path that connect the VNFs. We used real data about cellular networks to train the models, and combined a cluster approach to create group of cells with similar statistics in order to reduce the number of models created. We also compared the proposed models with traditional ML models in order to show the gain of use DL models.

Chapter 6 presents a solution for SFC placement based on RL taking into account SFC availability, operational costs, and energy consumption. This solution is used in SPIDER to define the SFC placement based on the context information collected from the infrastructure. We present the Cand-RL algorithm, which uses RL based on Proximal Policy Optimisation (PPO) to select the suitable candidate node and define the redundancy strategy to meet availability requirements. We compare Cand-RL against two greedy algorithms in a variety of simulated scenarios. The results show that the Cand-RL outperforms the greedy algorithms, achieving a higher acceptance rate and a good balance between availability, placement cost, and energy consumption.

Chapter 7 presents the proof-of-concept of SPIDER. We show how we implement the SPI-

DER using Kubernetes, to deploy the VNFs as containers. We also use Kubernetes mechanisms to connect the VNFs and to forward the traffic among them. We also conduct experiments to assess the SPIDER placement runtime and SFCs delay in two different networks.

Chapter 8 presents the general consideration of this thesis proposal, the scientific contributions and the next steps for the conclusion of the thesis.

2 BACKGROUND

This chapter describes some basics concepts needed to better understand the proposal of this thesis. Firstly, in Section 2.1, we describe the SFC paradigm, detailing the main steps in the SFC orchestration process. After, in Section 2.2 we describe the availability concepts and how the system availability can be calculated through Stochastic Petri Net (SPN) models. Afterwards, we present concepts about DL in Section 2.4, presenting more details about recurrent networks, which is important to understand a module of the framework proposed. Finally, we present concepts about RL in Section 2.5, focusing in the actor-critic paradigm, which is used to create other important module of the framework.

2.1 SFC PLACEMENT

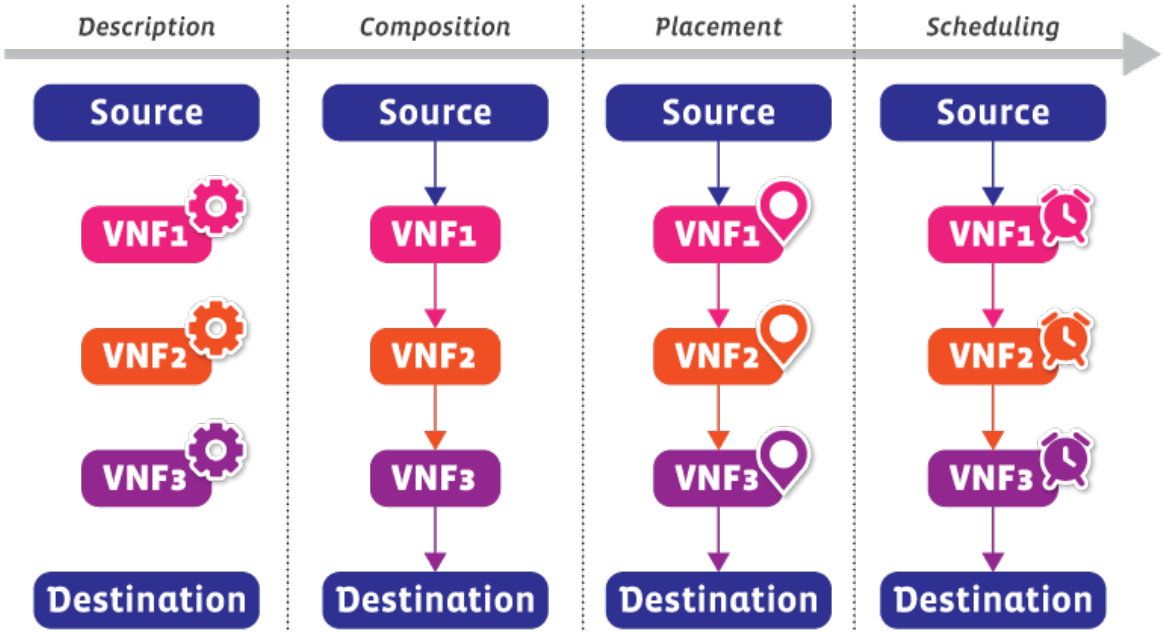
The NFV paradigm was proposed by the ETSI to achieve flexible scaling, redundancy and lower total cost of operation (GUPTA et al., 2017). It emerged to meet the challenge of exponential growth resulting from user demand for new diverse, agile, and high-quality mobile-enabled services (MIRJALILY; ZHIQUAN, 2018). In the NFV paradigm, network functions are deployed as virtual functions in commodity servers rather than special-purpose hardware (CHAI et al., 2019). This results in both lower CAPEX and OPEX for the service provider and offers new ways to design and deploy different network services types (KHEZRI et al., 2019). NFV providers usually offer the network services as part of a SFC, a set of VNFs have to be processed in a predefined order (XIAO et al., 2019a).

As mentioned by Mirjalily et al. (MIRJALILY; ZHIQUAN, 2018), there are four stages needed to deliver an SFC. The stages are identified for handling possible issues related to automatic SFC deployment in network infrastructures: description, composition, placement, and scheduling. Figure 1 shows an example with an SFC composed of three generic functions (VNF1, VNF2, and VNF3) as well as their source and destination.

The first stage is the description which is responsible for describing each VNF and detailing its functional and nonfunctional properties. This stage is important for network providers to specify the functionalities and requirements of each VNF needed to make the SFC completely operational.

There are several alternatives that can be applied for VNF description. Both the Web

Figure 1 – Stages of SFC orchestration.



Source: the author (2023).

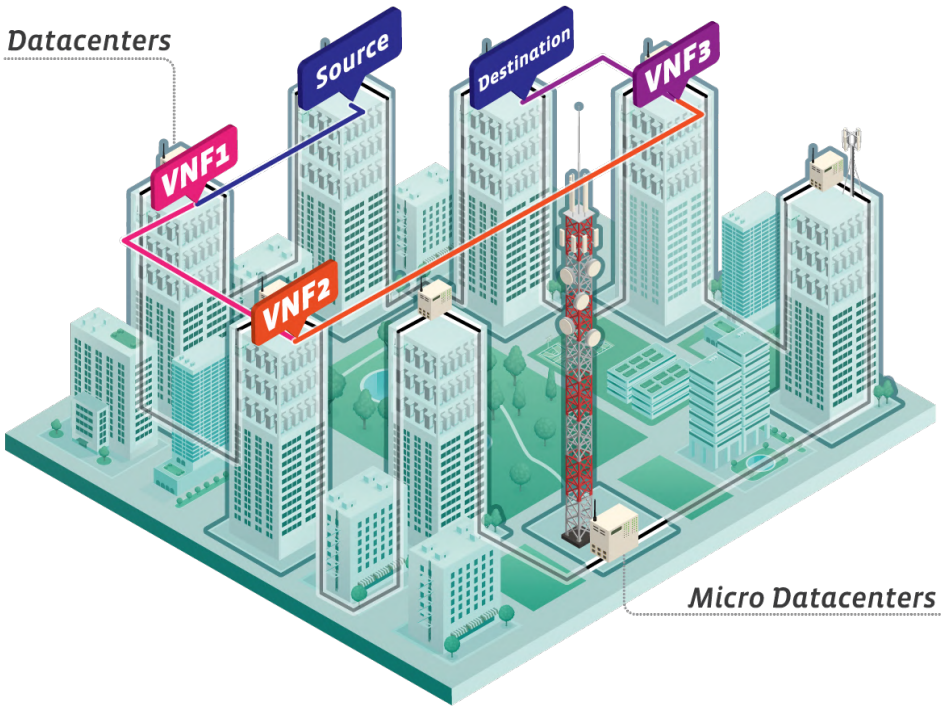
Service Description Language (WSDL) (BOUBENDIR et al., 2017) and Unified Service Description Language (USDL) (CARDOSO et al., 2010; SUN; DONG; ASHRAF, 2012) are efforts for web services description, while Cloud Service Declarative Definition Language (CSDL) and its variations (GHAZOUANI; SLIMANI, 2017) focus on the description of cloud services. As highlighted by the authors in (MEHRAGHDAM; KARL, 2016), the YANG data model can be used for VNF description. An additional data model that can be used is the Topology and Orchestration Specification for Cloud Applications (TOSCA), that provides a data model and templates to orchestrate and manage application services with NFV in cloud environments (KATSAROS et al., 2014).

The second stage in SFC deployment is the composition. This step defines how the VNFs will be ordered in an SFC. Although most of these network functions can run in an independent way, their order must be defined carefully and is determined by the service they seek to offer. For example, considering an Internet of Things (IoT) scenario, where one can have large data to transmit, the first VNF of an SFC can perform data compression, in order to reduce the bandwidth consumption in the network (REN et al., 2019).

The third stage is the SFC placement. It establishes where each VNF from a given chain will be hosted. The main goal of this phase is to ensure an efficient allocation, taking into

account several constraints from the infrastructure and customers (MIRJALILY; ZHIQUAN, 2018). However, some authors present other concerns for this stage. The authors in (KOUAH et al., 2018) divided the SFC placement into two sub steps: VNF placement and SFC chaining. They see placement as one step that consists in assigning VNFs to the physical infrastructure, whereas chaining is defined as interconnecting VNFs in order to deploy an SFC. In other words, the SFC placement defines the servers where the VNFs are deployed, and the SFC chaining defines the physical links that will connect these servers. A different definition is presented in (CHEN; LIAO, 2019), where the SFC placement can be considered in two distinct moments: when an SFC request arrives in the system and when it is needed to migrate an already deployed one. Finally, SFC migration is an important operation invoked in order to overcome service interruption. A placement example of the SFC considered in Figure 1 is showed in Figure 2. The VNFs are allocated across different distributed data centers and connected through virtual links.

Figure 2 – SFC placement example



Source: the author (2023).

The fourth step is the SFC scheduling. After composing and defining which servers the VFNs will be allocated, the scheduling step determines when that allocation will happen. The main focus of this step is to minimise the total execution time of the SFCs allocated (MIRJALILY; ZHIQUAN, 2018). As highlighted by Alameddine et al. (ALAMEDDINE et al., 2019), the scheduling

step is still a field under investigation, and received little attention by researchers. For instance, the authors in (RIERA et al., 2014) formulated the scheduling problem as a flexible job-shop problem. The authors in (MIJUMBI et al., 2015) proposed a Tabu search based algorithm to solve the scheduling problem. The work presented in (QU; ASSI; SHABAN, 2016), formulated the scheduling problem as an Mixed-Integer Linear Programming (MILP) and solved it using a genetic algorithm.

In this thesis, we focus on the SFC availability, and in the next section we describe the main concepts about availability and how to model it.

2.2 AVAILABILITY CONCEPTS

Dependability is related to the ability of systems to avoid failures and interruptions in a given period of time without negatively affecting the user experience (AVIZIENIS et al., 2004). It is a critical aspect in modern systems, both technically and commercially, from the design of systems (both hardware and software) to their implementation and operation (ANDRADE et al., 2017). It is often used as an umbrella term for reliability, availability, security, confidentiality, integrity, and maintainability (COSTA et al., 2016). Although these terms are closely related, they differ from each other. For example, reliability refers to the probability that a system will deliver a service properly until a certain time without any failure (DÂMASO; ROSA; MACIEL, 2017). On the other hand, availability is defined as the percentage of time that the service has worked properly during the total operation time, and as such assumes that the system can be repaired during its operation (ARAUJO et al., 2018). In this proposal, we will focus on the availability evaluation.

The availability of a system component can be calculated using Equation 2.1 (FAN et al., 2017):

$$A = \frac{Uptime}{TotalTime} = \frac{Uptime}{Uptime + Downtime}, \tag{2.1}$$

which is defined by the system uptime, i.e., how much time the system remains operational divided by the total operation time (the sum of the uptime and the downtime of system). The system uptime can also be defined as the Mean Time to Failure (MTTF), the time when the system is not at fault and working properly. Conversely, system downtime can be defined as the Mean Time to Repair (MTTR), that is, the mean time the system fails and is under repair.

Thus, availability can be defined as:

$$A = \frac{MTTF}{MTTF + MTTR}. \tag{2.2}$$

Numerous techniques exist for evaluating the availability of complex systems, such as stochastic Petri nets (MOLLOY, 1982a), continuous Markov chains (BOLCH et al., 2006), and fault trees (DÂMASO; ROSA; MACIEL, 2014). In this proposal, we make use of the SPN which is a mathematical formalism to define the relationships of system components.

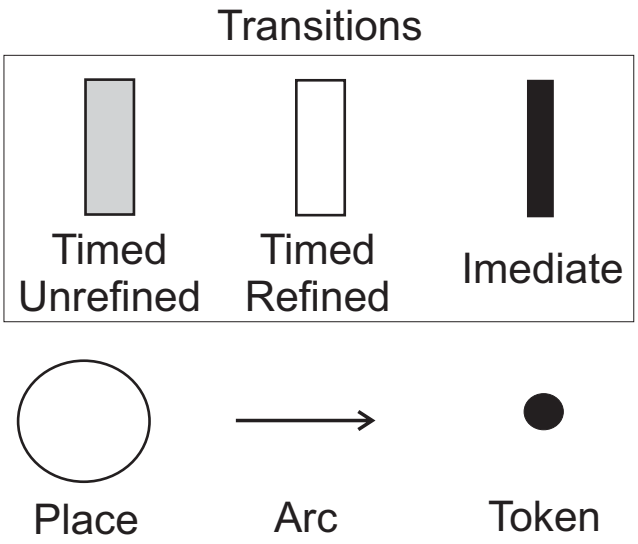
Petri Nets are a graphical and mathematical modelling tool used to represent different types of systems. They are a robust tool to describe systems characterized mainly by their concurrency, asynchrony, distribution, parallelism, non-determinism, and/or stochastic, such as distributed computing, telecommunication, control systems, workflow management (CHEN; HA, 2018).

Since Petri's seminal work, many representations and extensions have been proposed, allowing more concise descriptions and representing system features not observed in early models (MURATA, 1989). SPN models are a special case of Petri Nets and the proposals regarding performance evaluation sought an equivalence between SPN and Continuous Continuous Time Markov Chain (CTMC) (GERMAN, 2000). In order to obtain an equivalence between a PN and a CTMC, it was necessary to introduce temporal specifications such that the future evolution of the model, given present marking, was independent of the marking history. Therefore, SPNs can be translated to CTMC, which may then be solved to reach the desired performance or dependability results (MOLLOY, 1982b; MARSAN et al., 1994; TRIVEDI, 2001).

Considering the Figure 3, the rectangles represent SPN transitions. The black rectangle represents immediate transitions, and fire when it is enabled, without wait any period of time. The white transitions are called stochastic, and they fire following a stochastic process, according to a probability distribution function. The gray rectangles are unrefined transitions and are used to represent that no event occurred and nothing was collected. The white circles represent places, while the small black circles are tokens; a set of tokens assigned to a given place is called markup. The arcs (directed edges) are used to connect places to transitions.

The data flow is the main aspect of the SPN behaviour. Tokens are created and consumed according to the transition firing conditions (GERMAN, 2000). The immediate transitions have higher priority than other transitions, but the user may define the priority among the existing immediate transitions. There are also guard functions (represented by boolean expressions) that

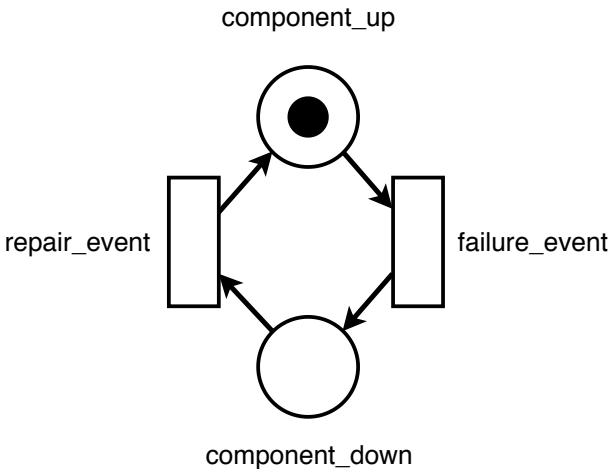
Figure 3 – SPN Components



Source: the author (2023).

control the transitions, declaring specific conditions regarding the model marking. Therefore, if the guard function produces a *true* value, it is enabled and will fire, otherwise, it is considered disabled (MARSAN et al., 1994).

Figure 4 – Example of an SPN model to represent the availability of a generic component



Source: the author (2023).

Figure 4 presents an example of an SPN that represents a generic component availability. If there is a token at place *component_up*, it means that the component is operating properly. If the transition *failure_event* fires, it will represent that the component is in failure; this event consumes one token from place *component_up* and produces on token at place *component_down*, representing that the component is unavailable. This transition is modelled according to a stochastic process (usually following an exponential distribution) defined by

the MTTF parameter. The transition *repair_event* also follows a stochastic process and represents the repair event of the component, defined by the MTTR parameter. Thus, when this transition fires, one token is consumed from place *component_down* and generated at place *component_up*, turning the component available.

In this example, the availability of the component is the probability of having, at least, one token at place *component_up*.

SPN can be used to model different SFC configurations and assess their availability. Usually, SFCs are deployed as sequential functions (CAI et al., 2020), and can be considered a series system, where the failure of any function makes the whole SFC unavailable. Adding redundant functions (in a parallel configuration) can increase the availability of an SFC, however, these additional components also increase the system costs, a key consideration in SFC placement. ML algorithms are an option for planning SFC placement that takes into account the actual network resources as well as others metrics of interest (LI et al., 2019), including availability. We will introduce two different ML algorithms later in this chapter.

In the next subsection, we will present concepts about the MAPE-K, a framework which served as inspiration for the creation of SPIDER.

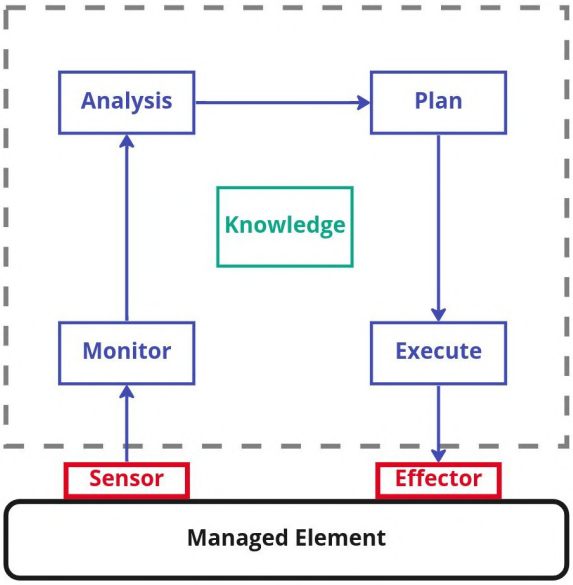
2.3 MAPE-K

MAPE-K (COMPUTING et al., 2005) is a control loop mechanism introduced in by IBM in 2005 created for the implementation of autonomic systems. Figure 5 illustrates the basic MAPE-K loop, which can be divided in four steps.

The first step is the monitor function, which is responsible to collect, aggregate, filter, and report details collected from a managed element. The information collected can comprise topology information, metrics, configuration property settings, and so on. It is important to highlight that these components can be software (databases, APIs, piece of software packages, etc) or hardware (servers, routers, and other IT equipment), and it must have an interface that allows it to be monitored, which is named sensor. The information collected is stored in the knowledge source, which is a collection of data that is shared with other steps of the loop.

The second step of MAPE-K loop is the analyse function, which is responsible to observe and analyse situations and determine if some change needs to be made. This function usually models complex situations such as time-series forecasting and queuing models. Then, the monitored information from the system components can be used to learn about the environment

Figure 5 – MAPE-K loop.



Source: adapted from (ASLANPOUR; GHOBAEI-ARANI; TOOSI, 2017).

and help predict future situations.

The plan function is the third step of the loop, which defines the actions needed to achieve the goals and objectives of the system manager. In the original document, IBM defines that the planning mechanism uses policy information to guide its work. Therefore, tanking into account the information monitored by the monitor step and processed by the analyse step, the plan function defines actions according to the policy previously defined by the system manager. These actions can be a single command or starts a complex workflow.

The last step of the MAPE-K loop is the execute function, which is responsible to schedule and perform the changes needed to the system, according to the actions defined in the plan function. In this way, this step can change the system context, updating the components of the systems based on the actions defined in the plan function. The mechanisms used to change the behaviour of the resources are aggregated into an effector. In addition to the changes, the execute function can also update the knowledge source.

These four steps work together to provide the autonomous loop control for an IT system. Therefore, each step sends and receives information to/from other steps, exchanging appropriate knowledge and data. All these steps are usually executed automatically, i.e., run without a human intervention. However, some of the steps can be executed with the human intervention, for instance, the system manager can set up the loop only to monitor and analyse automatically and send the information generated to a console or to another system.

To create the SPIDER modules that make decisions, we use different machine learning algorithms. In the next subsections, we present the concepts of deep learning (more precisely recurrent neural networks) and reinforcement learning.

2.4 DEEP LEARNING

ML, at a high level, "enables an algorithm to make predictions, classifications or decisions based on data, without being explicitly programmed" (ZHANG; PATRAS; HADDADI, 2019). In the last years, deep neural networks, DL, have increased in prominence in research and practice. DL is a sub-branch of ML that addresses the limitations of single-layer neural networks by using multiple layers to transform their input into higher-dimensional representations and then into the output. The emergence of DL is largely driven by increased computational power through heterogeneous processors such as GPUs, the availability of large data sets for training, and advances in optimization algorithms (KRAUS; FEUERRIEGEL; OZTEKIN, 2020).

In contrast to traditional ML and neural networks, DL can cater for high dimensionality in data thus enabling DL networks to model highly complex non-linear relationships between variables (KRAUS; FEUERRIEGEL; OZTEKIN, 2020). As such, it is particularly suitable to the mobile and wireless networking domain which is characterized by massive volumes of high velocity unlabeled heterogeneous data (ZHANG; PATRAS; HADDADI, 2019). In addition, DL can significantly reduce operational and capital expenditure by reducing or eliminate the time and effort required by valuable and scarce human resources in feature extraction (ZHANG; PATRAS; HADDADI, 2019). However, DL is not without its limitations. It hides its internal logic to the user thereby sacrificing accuracy for interpretability with practical and ethical consequences (GUIDOTTI et al., 2018). Other limitations include vulnerabilities to adversarial and privacy attacks (ANSARI et al., 2020), computational demands unsuitable to small-form computing in edge networks, and the time taken to find optimal configurations, particularly for highly-parameterised data and multi-step network prediction (MA; GUO; ZHANG, 2020; ZHANG; PATRAS; HADDADI, 2019).

Common DL architectures include, Restricted Boltzmann Machine (RBM), auto-encoders, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) (ZHANG; PATRAS; HADDADI, 2019). These can be differentiated by the data structures that they target, and their respective tuning parameters (KRAUS; FEUERRIEGEL; OZTEKIN, 2020). For example, Multi Layer Perceptron (MLP) targets feature vectors of fixed length and are tuned by the activation

function setting and the number of layers and units (KRAUS; FEUERRIEGEL; OZTEKIN, 2020). In contrast, CNNs target high-dimensional data with local dependencies and are tuned by the number and width of convolutional kernels or filters (KRAUS; FEUERRIEGEL; OZTEKIN, 2020). Because both MLP and CNN assume that all inputs are independent of each other, they are not suited to modelling sequential data, where sequential correlations exist between samples. RNNs specifically target sequential data, like time series data flows from mobile networks. As such, we focus on the use of RNNs in this proposal.

2.4.1 Recurrent Neural Networks

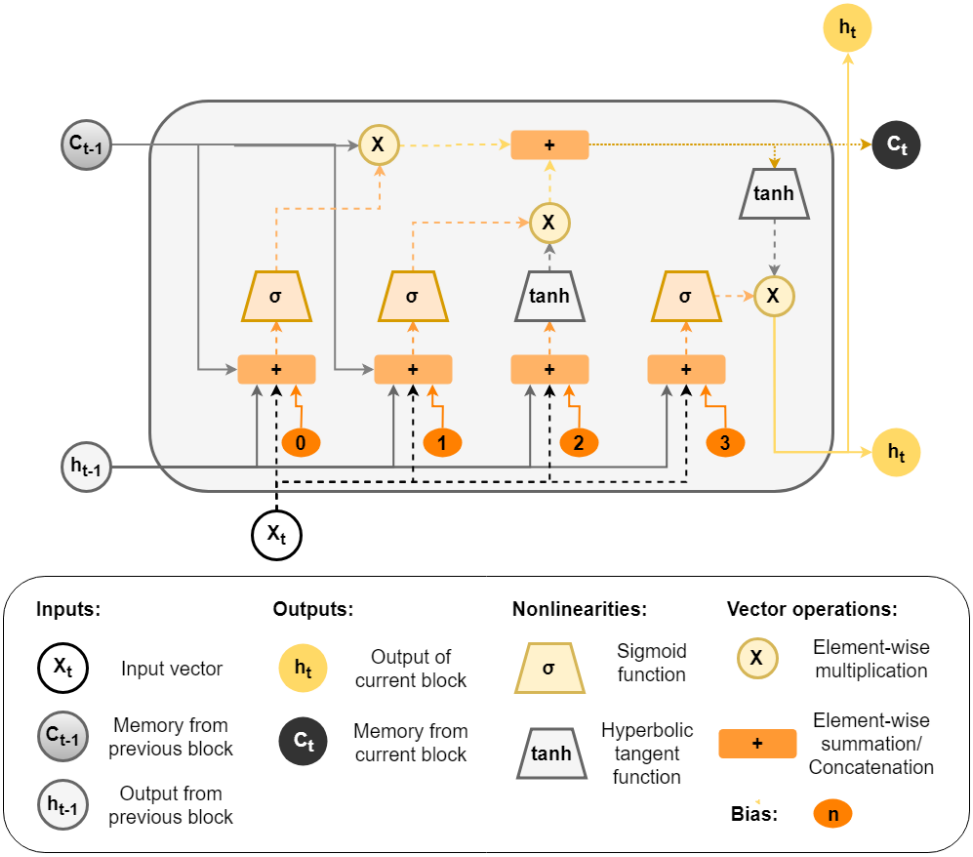
As discussed earlier, traditional ML, MLPs and CNNs, typically target input vectors with fixed dimensions. The formalization for sequential data is fundamentally different and thus MLP and CNN are not suitable for time series data. RNN architectures were specifically designed to model sequential data by producing output via recurrent connections (cells) between hidden units (ZHANG; PATRAS; HADDADI, 2019). These recurrent connections are the *memory* that stores the previous data allowing RNNs to learn the temporal dynamicity of the sequential data (ORDÓÑEZ; ROGGEN, 2016). In RNNs, the output of the current timestamp is influenced by the output of the previous timestamps, which is critical when the sequence of events or data is important to determine the outcome of a problem. Despite being designed to model sequential data, early RNNs suffer from long time dependencies resulting from both vanishing and exploding gradient problems (HOCHREITER, 1998) that negatively impacted training using the Back-Propagation Through Time (BPTT) algorithm. To overcome this limitation, a variation of traditional RNNs was proposed, Long Short-Term Memory (LSTM), which introduces the concept of gates to mitigate gradient problems (ORDÓÑEZ; ROGGEN, 2016; JOZEFOWICZ; ZAREMBA; SUTSKEVER, 2015; GREFF et al., 2017).

2.4.1.1 Long Short-Term Memory Networks

Figure 6 presents the basic schema of an LSTM unit (adapted from (YAN, 2016a)).

The LSTM unit state updates through specific gate operations: write (input gate), read (output gate), or reset (forget gate). These operations consist of component-wise multiplications and apply different functions in the input data as shown in the following equations

Figure 6 – Example of a LSTM block



Source: adapted from (YAN, 2016b).

(ORDÓÑEZ; ROGGEN, 2016):

$$i_t = \sigma_i(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.3}$$

$$f_t = \sigma_f(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.4}$$

$$c_t = f_t c_{t-1} + i_t \sigma_c(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.5}$$

$$o_t = \sigma_o(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.6}$$

$$h_t = o_t \sigma_h(c_t) \tag{2.7}$$

where i, o, f, and c are, respectively, the input gate output, the output gate output, the forget gate output, and the unit activation vector. h_t is the hidden value of the unit (i.e. the memory state) and has the same size of the previous vectors. The non-linear functions of the input,

forget, and output gates are represented by σ_i, σ_f, and σ_o, respectively. The weight matrices of the unit state (c), input (i), output (o), and forgot (f) gates are represented by W_{xi}, W_{hi}, W_{ci}, W_{xf}, W_{hf}, W_{cf}, W_{xc}, W_{hc}, W_{xo}, W_{ho}, and W_{co}, where x is the input and h the hidden value of LSTM unit. Finally, b_i, b_f, b_c, and b_o are, respectively, the bias of input gate, forget gate, cell, and output gate (ORDÓÑEZ; ROGGEN, 2016). While LSTM addresses gradient problems, critics have noted that the LSTM architecture is *ad hoc*, has a substantial number of components whose purpose is not immediately apparent, and that it is characterized by long training times (JOZEFOWICZ; ZAREMBA; SUTSKEVER, 2015; LIN; HSUEH; LIE, 2016).

Previous works showed that LSTM models provides better results for traffic prediction for long-term dependencies due to the ability to keep relevant information from the input data in the "additional gates" incorporated into the cells (ABDULJABBAR; DIA; TSAI, 2021). As shown in (AZARI et al., 2019), LSTM models can outperform traditional approaches for traffic prediction. Therefore, in this study, we compare LSTM with other traditional ML approaches.

2.4.1.2 Gated Recurrent Unit

Gated Recurrent Unit (GRU) is a variation of LSTM which only uses two gates, an update gate and a reset gate. Indeed the update gate in a GRU replaces the input and forget gates used in LSTM and decides what input data will be kept (CHUNG et al., 2014). Furthermore and unlike LSTM, GRU exposes its memory content at each step balancing between the previous and new memory content (CHUNG et al., 2015).

The GRU activation, h_t^j, is represented in Equation 2.8. Considering the input data as a time series, at the timestamp t, h_t^j is the linear interpolation between the previous unit data (h_{t-1}^j) and the current data ($\tilde{h}_t^j$).

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.8}$$

where z_t^j is the output gate, and defines what should be forgotten and what should be kept in the GRU unit. The output gate is defined in Equation 2.9 (CHUNG et al., 2015):

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1}) \tag{2.9}$$

where the current and previous weight matrices are W_z and U_z, respectively. In a simplified way, this procedure taking a linear sum between the previous hidden states h_{t-1} and the current

input x_t and applies the sigmoid function (σ).

The new memory unit is calculated as described in Equation 2.10:

$$\tilde{h}_t = than(Wx_t + r_t \odot Uh_{t-1}) \tag{2.10}$$

where $\odot$ is the element-wise multiplication, r_t refers to reset gate, and its output can be calculated as defined in Equation 2.11:

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2.11}$$

Researches suggests that GRUs are easier to generalize with faster training times while achieving comparable performance outcomes (JOZEFOWICZ; ZAREMBA; SUTSKEVER, 2015; CHUNG et al., 2015; KAISER; SUTSKEVER, 2015; TROFIMOVICH, 2016). As such, we also propose a GRU for comparison against an LSTM in this study.

2.5 REINFORCEMENT LEARNING

RL is a learning paradigm that has featured more prominently in recent literature. It refers to the use of autonomous software agents that learn to perform a task by trial and error without any human intervention (GOODFELLOW et al., 2016). In other words, in RL, the learning process happens through the iterative interaction of the agent with the environment. As such, unlike DL, RL does not use a fixed data set for training, validation and testing (TROIA; ALVIZU; MAIER, 2019). Instead, RL uses the feedback from the environment in which the agent is inserted. In the learning process, the agent tries to maximise a reward by observing the consequences of its actions (ARULKUMARAN et al., 2017).

The tasks performed by RL agent can be classified as episodic tasks or continuing tasks (RAVICHANDIRAN, 2018). In episodic tasks, there are one or more final states. When the agent achieves a given state, the task restarts. To use an analogy, the agent is like a driver in a racing car video game where each race is an episode. The agent starts each race and plays until the end of a race, concluding an episode. When the race is over, the agent can starts again from the initial state; each episode is independent of the others. In continuing tasks, there is no terminal state. The agent actuates in the environment achieving rewards indefinitely, more like a robot that continuously responds to commands and completes tasks.

RL is supported by a formalism called the Markov Decision Process (MDP), which is composed of (ARULKUMARAN et al., 2017):

- A finite set of states $\mathcal{S}$, plus a distribution of starting states $p(s_0)$.
- A finite set of actions $\mathcal{A}$.
- A set of dynamic transition $\mathcal{P}(s_{t+1}|s_t, a_t)$ that maps an action to a state at time *t* onto a distribution of states at time $t+1$.
- A reward function $\mathcal{R}(s_t, a_t, s_{t+1})$.
- A discount factor $\gamma \in [0, 1]$, where lower values means more immediate rewards.

In general, a policy $\pi(a_t|s_t)$ defines the agent behavior. At instant *t*, the policy maps an action a_t into a state s_t. A reward r_t is calculated, and transitions to the next state s_{t+1} following a state transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. Considering an episodic MDP, this process continues until the terminal state is reached. For each step, the reward is accumulated from the environment resulting into the returned value $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The main goal of RL is to find the optimal policy π^* (Equation 2.12) that results in the maximum expected return (reward) for all states.

$$\pi^* = \underset{\pi}{\texttt{argmax}}\, \mathbb{E}[R|\pi] \tag{2.12}$$

A widely-used class of algorithms in the literature are value-based methods (OSBAND et al., 2016; MOUSAVI; SCHUKAT; HOWLEY, 2017; PAN et al., 2018). These algorithms try to extract the near optimal policy based on the value function, which is defined in Equation 2.13. The value (V) is the expected long-term reward achieved by the policy (π) from a state s.

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{2.13}$$

The Q-learning is a family of algorithms which learn how to optimize the quality of an action (Q value). Equation 2.14 defines the Q value of an action a in a given state s following a policy π at time t.

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.14}$$

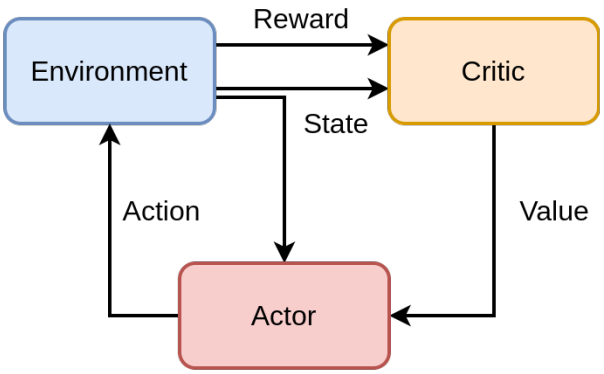
The best policy is derived from the Q value of each state, e.g., select the action which returns the maximum expected reward for a given state. The Q value is learned in an interactive way during agent training. The Bellman Equation (Equation 2.15) updates the Q value during the training (HASSELT, 2010).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \underset{a}{max} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \tag{2.15}$$

The new estimation of Q value is the sum of the old estimation and the error. The error is defined by the reward achieved, r_{t+1}, plus the difference between the new Q value obtained, $\underset{a}{max} Q_t(s_{t+1}, a)$ minus the old value, $Q_t(s_t, a_t)$. γ is the discount factor and is a value between 0 and 1 t that defines the importance of the immediate reward. α is the learning rate that defines how much the Q values updated.

Another method to find the policy π is called **Actor Critic** (SUTTON; BARTO, 2018). Figure 7 shows the architecture of a traditional Actor Critic method. The actor is responsible to choose the action which guides the agent; the critic learns the quality of the actions from actor based on the reward from the environment i.e. the Q value.

Figure 7 – Actor-critic method



Source: adapted from (WANG; VELSWAMY; HUANG, 2017).

The actor and critic are two independent functions as shown in Equation 2.16 and Equation 2.17, respectively. These functions have parameters, θ for actor function and w for the critic function, which are learned during the training. These functions are usually defined as neural networks, but other functions can be used.

$$actor = \pi(s, a, \theta) \tag{2.16}$$

$$critic = q(s, a, w) \tag{2.17}$$

For each time step t, the state is forwarded to the actor and critic. The actual policy takes the state and outputs an action a, which results in a new stat S_{t+1} and a reward r_{t+1}. Afterwards, the critic calculates the Q value (q_w) and the actor updates its parameters using the θ variation as calculated in Equation 2.18. After updating its parameters, the actor produces a new action and the critic updates its parameters using w variation which is calculated as shown in Equation 2.19. The actor and critic functions have different learning rates, α and β, respectively.

$$\Delta\theta = \alpha\nabla_\theta(\ln \pi_\theta(s,a))q_w(s,a) \tag{2.18}$$

$$\Delta w = \beta(r(s,a) + \gamma q_w(s_{t+1},a_{t+1}) - q_w(s_t,a_t))\nabla_w q_w(s_t,a_t) \tag{2.19}$$

The parameters of actor and critic functions are updated, respectively, as shown in Equations 2.20 and 2.21.

$$\theta \leftarrow \theta + \Delta\theta \tag{2.20}$$

$$w \leftarrow w + \Delta w \tag{2.21}$$

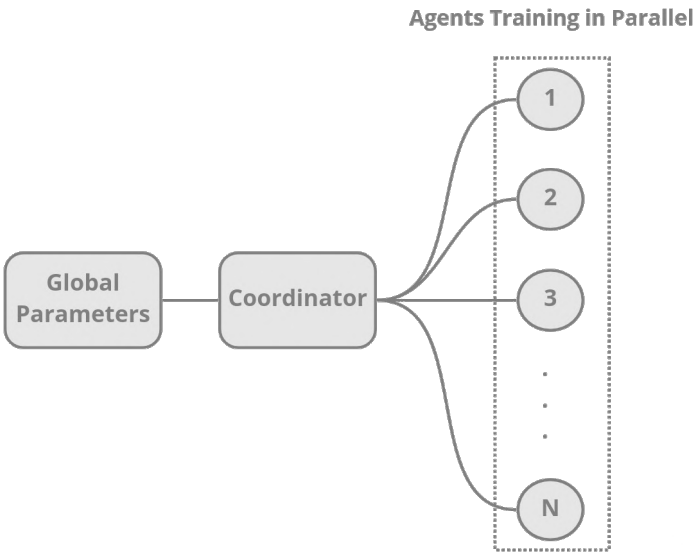
2.5.1 Advantage Actor-Critic

The Advantage Actor-Critic (A2C) method is variant of the actor-critic method which includes an advantage concept derived from the Q value as shown in Equation 2.22. While the state value V defines how good is to be at a state (Equation 2.13), in effect the expected return from a policy (SUTTON; BARTO, 2018), the advantage A defines how good an action is when compared with other actions. In A2C, the RL agent uses the advantage value instead of the Q value during training, which reduces the high variance of the policy networks and stabilizes the model training (PENG et al., 2018).

$$\begin{aligned} Q(s,a) &= V(s) + A(s,a) \\ A(s,a) &= Q(s,a) - V(s) \\ A(s,a) &= r + \gamma V(s_{t+1}) - V(s) \end{aligned} \tag{2.22}$$

In the A2C algorithm, multiple agents explore different parts of the environment in parallel thus increasing exploration efficiency compared to a single agent (MNIH et al., 2016). The parameters of all the agents are used to update a set of global parameters. These global parameters are the actor parameters (see Equations 2.16 and 2.17) and they are used by all parallel agents during the training in a set of environments, as shown in Figure 8. A coordinator controls the update of global parameters, which happens after each agent finishes its exploration based on the values of all agents. Then, in the next iteration, all agents use the same global parameters to explore the environment. This parameter synchronisation makes the training more cohesive and potentially faster.

Figure 8 – A2C algorithm schema



Source: adapted from (SIMONINI, 2018).

2.5.2 Proximal Policy Optimisation

PPO combines the idea of having many agents from A2C with the idea of exploring the policy region in the optimisation problem during the training (SCHULMAN et al., 2017). Instead of using the log of π to update the policy parameters (Equation 2.18), PPO uses the ratio between the probability of action under the current policy (π_θ) divided by the probability of action under the previous policy ($\pi_{\theta old}$). Equation 2.23 describes the ratio.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{2.23}$$

The policy loss in PPO can be defined as Equation 2.24.

$$L(\theta) = \hat{\mathbb{E}}_t = \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \hat{A}_t \right] \tag{2.24}$$

If the variation probability from the previous policy compared to the current policy is too high, it can cause an excessive policy update resulting in a large policy gradient step. In order to mitigate the exploding gradient problem resulting from a large step, PPO uses the clipped surrogate objective function (Equation 2.25).

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.25}$$

As per Equation 2.25, we have two probability ratios, one non clipped and one clipped between $1-\epsilon$ and $1+\epsilon$, where ϵ is a hyper parameter which defines the clip range. The minimum between these ratios is taken as the final objective in the PPO algorithm, which is the lower (pessimistic) bound of the unclipped objective.

In this work, we used the PPO algorithm since it was shown to outperform several other "on-policy" gradient methods and also provide relatively stable training behaviour (SCHULMAN et al., 2017).

2.6 CONCLUDING REMARKS

In this chapter, we presented the main concepts necessary to understand the main contributions of this thesis. We started the chapter presenting the main concepts about SFC placement in Section 2.1, describing the main stages and how they interact with each other. Afterwards, we explained the basic concepts about availability and how to estimate it using SPN models in Section 2.2. These concepts are needed to understand how we estimated the availability of a SFC placement. Then in Section 2.3 we presented the MAPE-K framework, which as used as reference to propose the SPIDER. We presented the concepts about ML in Sections 2.4 and 2.5. In Section 2.4 we presented the concepts about recurrent neural networks, with focus on LSTM and GRU. Finally, in Section 2.5 we presented the concepts about RL, describing the mains concepts about the A2C method and the PPO algorithm. In the next chapter, we describe the main related works found in the literature.

3 RELATED WORKS

In this chapter, we present some works that also proposed frameworks for the SFC placement. Firstly, we provide a brief description about the related works found in the literature, and after we compare these works with the proposal of this thesis. In Section 3.1 we provide a brief description of the frameworks for SFC placement found in the literature. Section 3.2 presents a comparison between the frameworks and highlights how our work differs from the another ones. The results obtained in this chapter were the result of a systematic review that was published in (SANTOS et al., 2022).

3.1 FRAMEWORK FOR SFC PLACEMENT

It is important to highlight that we do not consider works that only proposed algorithms for the SFC placement, but do not present a complete solution for the SFC placement. For instance, several works proposed algorithms or heuristics for SFC placement (BHAMARE et al., 2017; BHAMARE et al., 2018; CAI et al., 2020; MARTÍN-PÉREZ; BERNARDOS, 2018; SUBRAMANYA; HARUTYUNYAN; RIGGIO, 2020), but do not detail relevant aspects of SFC placement such as how the information from the environment is treated in their solution, what the communication protocols are used to communicate with the infrastructure, how the placement is done in a real infrastructure (Management and Orchestration (MANO) tool or Virtual Infrastructure Manager (VIM) tool used). We also do not consider works that have no focus on distributed scenarios, which is out of scope of this thesis proposal.

In (GUPTA et al., 2017), Gupta et al. propose the cost optimised latency aware placement (COLAP) which is a framework that implements a randomised selection of clouds for optimum cost in SFC placement and a heuristic for placement based on the latency prediction through Support Vector Regression (SVR). The COLAP framework consists in two phases. In the first phase, a set of clouds to deploy the SFC is selected using random search taking into account the cost (The cost is composed of cloud resources and inter-cloud links) and latency as constraints. For each iteration, the set of clouds that provides the lowest cost and meet the latency requirements is stored. After many iterations, the algorithm is able to select the best cloud to deploy the SFC (using random search). In the second phase, the latency is predicted using SVR. This prediction can be used for both initial SFC placement and scheduling. The

model takes as input a feature vector that describes two clouds and predicts the latency between them.

The work presented in (SHEN; ZHANG, 2018) proposes a framework to scale the components of an SFC without compromise its performance. This framework is based on the monitoring of traffic, which is used to decide the scale strategy for the SFCs. Then, the framework tries to estimate the amount of demanded resources in order to scale up/down the VNFs present in a server. This estimation is done based on an equation which considers the CPU core frequency and application computing intensity. In this case, the number of packets in a traffic flow servers as base to estimate the number of CPU cores needed to process them. In addition, a hashing algorithm is proposed to distribute packets between the dedicated threads of same VNF.

The work presented in (GUO et al., 2019) proposes a cloud-edge SFC orchestration architecture based on blockchain and Deep Reinforcement Learning (DRL). The blockchain is used to establish trust between service providers and smart devices, selecting high-ability through the consensus. Then the blockchain is responsible for the resource and user registration, authentication and transaction registration, ensuring the guarantee of credibility and reliability for the SFC placement. The DRL module is used to support automatic SFC orchestration and dynamic adjustment. A DRL algorithm is used for the SFC placement in order to reduce the SFC cost (considering the SFC orchestration and migration), subject to resources and link constraints. An algorithm based on Asynchronous Advantage Actor Critic is proposed for the SFC placement.

In (NGUYEN et al., 2019), the authors propose a system that enables the deployment of common IoT applications as SFCs, where these functions are placed across multiple edges and clouds. The system is composed of three main components. The Edge Orchestrator is responsible to expose the resource information of the edge for the Global Optimiser, which is hosted at cloud and is responsible to make the SFC placement in a optimised way. There is also the Network Controller that manages the network resources and establishing connectivity among the VNFs that implement the SFCs. For the SFC placement, the authors formulate the problem as a non-convex Integer Programming problem. They consider latency for IoT devices while taking into account the specifications of connection between clouds (where are VNFs deployed) and IoT gateways, such as the distance to IoT devices and the connectivity to multiple edges and clouds. Furthermore two algorithms to solve the problem are proposed. The former is a Markov approximation that uses multistart and batching techniques, while the latter is a heuristic named node ranking-based placement algorithm that is based on the

number of VNFs that each node, creating a ranking that serves as base for the placement.

In (SANTOS et al., 2020b) the authors propose a controller to allocate container-based SFCs in Fog computing environments, specifically for smart cities use cases. The controller is implemented as an extension of the Kubernetes platform. The default container scheduler of Kubernetes calls the extension proposed when a scheduling decision is needed in order to optimise the SFC allocation. The SFC controller can select the most suitable nodes to allocate the SFC based on two approaches: location-aware and latency-aware. If the latency-aware schema is selected, the SFC controller best candidate node based on the calculation of Dijkstra's shortest path algorithm. If the location-aware schema is selected, the node selected is made based on the latency minimisation depending on the target location defined by the network manager.

The work presented in (SURIANO et al., 2020) proposes a methodology based on the ETSI-NFV framework to assess the trustworthiness and reliability of SFCs through remote attestation. Similar to the work presented in (SHEN; ZHANG, 2018), this methodology is based on the monitoring of VNFs deployed in the infrastructure. Some VNF properties and performance indicators are monitored by SDN controllers and reported to an orchestrator. The orchestrator is responsible to associates VNFs for an SFC based on the information gathered and integrity validation taking into account both trustworthiness and reliability requirements. The reliability requirements can be defined by the network manager and examples are: bandwidth conditions, CPU and memory usage, etc. These requirements are calculated through a "reliability function", which determines the reliability level of each VNF. Therefore, the VNFs with highest reliability levels are selected to compose the SFC.

Similar to the work presented in (NGUYEN et al., 2019), the work presented in (HU et al., 2020) also proposed a SFC management framework for IoT scenarios. The framework contemplates the first three steps of the SFC orchestration, as detailed in the Subsection 2.1: description, composition, and placement (they do not focus on scheduling). For the SFC placement, which is the more detailed part of the framework, the authors used a greedy approach to select the nodes to deploy the VNFs and the shortest path method to select the links.

The work presented in (LANGE et al., 2020) proposes a framework for the automated SFC placement. The authors propose an architecture which uses ML for the resource demand prediction for the SFC placement. In addition, they implement a testbed using the OpenStack. This testbed is used to conduct experiments and generate real data about the infrastructure.

Then,the data is used to evaluate deep learning models in the traffic prediction task. An additional use case about VNF anomaly detection is also presented, which can be used for the SFCs migration in case of problem detection.

The work presented in (SHAH; ZHAO, 2020) proposes a framework for NFV based resource allocation for IoT networks. The authors assume that the IoT devices are connected to a cellular network and cellular network is connected to the cloud network, where the framework executes. The SFCs are allocated as virtual nodes and links in the cellular network. The authors considered the placement problem in two stages. Firstly, the IoT devices are selected to be served, and after the framework selects the physical resources where the SFC will be placed. The objective function addressed in the paper is to optimise the utility of the system *"is the gain achieved from mapping the SFCs and the data rate achieved while subtracting the cost incurred over mapping to the substrate"* (SHAH; ZHAO, 2020). The problem formulation also considers several constraints about the nodes and links capacities, number of hops between the start and end node, and the power supply of IoT devices. To solve this problem, the authors considered a RL approach using multi-agents. Firstly, an agent (located in the network controller) decides the IoT devices will be served by the requested SFC. After, a set of agents decides in which physical nodes and links the SFC will be allocated. The action of these agents is then combined to compose the SFC placement. Then, the parameters of the network controller agent are updated based on the final action (SFC placement).

The authors in (TOUMI et al., 2020) proposed a physical programming based solution for multi-domain SFCs placement. A centralized framework was proposed for the SFC placement in distributed and multi domain networks. They formulated the SFC placement problem using three Physical Programming approaches: linear, non-linear, and global. In addition, they modelled the problem as a multi objective, where the end-to-end latency, the bandwidth per user, and the overall cost were considered. As constraints, the authors considered node affinity, end-to-end latency, and the SFC placement cost. Two different solutions are proposed to solve the optimization problem. The former is an exact solution, which is implemented as a Branch and Bound based algorithm which recursively creates solutions by eliminating branches that do not satisfy the constraints, as well as those with a fitness worse than the current best complete solution. The latter is a heuristic that combines genetic algorithm and local search to provide a scalable solution.

The work presented by Toumi et al. in (TOUMI; BAGAA; KSENTINI, 2021b) and (TOUMI; BAGAA; KSENTINI, 2021a) propose an RL based framework for the SFC placement with focus

on multi domain scenario. The framework architecture is composed of different orchestrators for a respective domain, the centralized Multi-Domain Orchestrator, and the Multi-Domain Interface. The RL agent is deployed in Multi-Domain Orchestrator, and is responsible to define placement of each VNF taking into account resource requirements (computing and network) and authorized domains. The Multi-Domain Interface is responsible to collect the information of each domain and build an abstracted view of the global network topology, which will be used to create the state fed to the agent. The local orchestrators perform a local placement and return to the Multi-Domain Interface the real cost and latency.

The previous two works are focused on the algorithms for the SFC placement and how to integrate them in the placement framework, but the work presented in (TOUMI et al., 2021) focuses on the solution itself. To implement the local domain orchestrator, the authors based on the ETSI's specification, where each domain is only aware of its local context, and ignores details about the other domains. When the SFC request arrives in the system, the Multi-Domain Orchestrator determines where the VNFs should be placed taking into account the information detailed in the Network Service Description (NSD). Afterwards, sub-NSDs are created based on the domains where the VNFs are placed. If two or more VNFs of the same SFC are placed in different domains, external links are created between the domains. Then, network service headers are created to determine the data flow of the SFC, inside and among domains. The authors proposed a proof-of-concept of the proposed framework. The VNFs are defined as LXC containers that are connected through OVS (Open vSwitch) switches.

Chen et al. (CHEN et al., 2021) proposed a lightweight SFC placement framework for wireless networks. The framework uses the Q-learning, an RL algorithm, to define the SFC placement with the purpose to minimize the wired and wireless delays delay of an SFC request. The authors argue that they used the Q-learning algorithm to ensure that the proposed framework would be lightweight. The framework is divided in three places. The first one is the data plane, which is composed of physical devices and links in the network. The second one is the control plane, which monitors the network status, such as topology information, link bandwidth, and resource usage. The control plane shares the information with the knowledge plane, which contains the RL, and is responsible to define the placement policy. In order to show the feasibility of the framework proposed, the authors presented a prototype based on Kubernetes and Open Network Operating System (ONOS). The Kubernetes is used to control the Docker container's life cycle, while the ONOS was used for the network controlling.

3.2 COMPARISON

In order to compare these approaches with the solution proposed in this thesis, we need to consider some relevant features about SFC placement frameworks. It is desired that the SFC placement are done in a dynamic environment and with the minimum human intervention (BHAMARE et al., 2016). These requirements are directly related to the need of handle context information about the infrastructure (to decide the best SFC placement according to the network situation) and the zero-touch paradigm (to place the SFC in an automatic way). Focused on the SFC placement, ML algorithms can be used to process this information and to choose the best SFC placement in an automatic way (CHEN et al., 2018b), instead to use deterministic solutions. In addition, it is desired that SFC management frameworks can be used in different scenarios, regardless of the software solutions used, thus regardless of management solutions (e.g., if the tool is based on VM or containers). Finally, the last feature considered for comparison is about the SFC availability, since customers' availability requirements need to be met (MOUALLA; TURLETTI; SAUCEZ, 2018).

Table 1 presents a comparison among the related work. One can note that to use the context information from environment is a very common feature in the works, since the information can guide the SFC placement and provide more satisfactory results. Many related works found also address the zero-touch paradigm, where the SFC placement is done without human intervention, i.e., fully automated. Only the work proposed by (GUPTA et al., 2017) does not specify mechanisms for the automated placement. The works proposed by (GUO et al., 2019) and (NGUYEN et al., 2019) specify approaches that can be used for automated SFC placement, but do not show how they implemented it. The works presented in (TOUMI; BAGAA; KSENTINI, 2021b) and (TOUMI; BAGAA; KSENTINI, 2021a) also did not present the implementation example of how to implement the automated solution, but the work presented in (TOUMI et al., 2021) illustrates it.

The use of ML algorithms for the implementation of frameworks for SFC placement has becoming more popular in recent works. One can note that seven works used ML in their solutions (GUPTA et al., 2017; GUO et al., 2019; LANGE et al., 2020; SHAH; ZHAO, 2020; CHEN et al., 2021; TOUMI; BAGAA; KSENTINI, 2021b; TOUMI; BAGAA; KSENTINI, 2021a), the majority of found works. The work presented in (TOUMI et al., 2021) mentions explicitly that the placement algorithm is out of the scope, but different solutions can be used.

No work presented solutions that are independent of MANO or VIM tools. For instance, the

works presented in (SANTOS et al., 2020b) and (CHEN et al., 2021) present specific solutions for Kubernetes, while the framework presented in (LANGE et al., 2020) is specific for the OpenStack. The work presented in (TOUMI et al., 2021) presents a proof-of-concept, but the authors do not use MANO tools, they only use LXC containers and OVS switches in the implementation. However, they mention that different software solutions could be used since the interfaces and protocols are implemented. The other works do not provide an implementation of their solutions, then is not clear if the solutions are specific for a VIM or not. Finally, one can see that no solution for SFC allocation takes into account the availability of SFC, even though it is an interesting point to take into account, especially in critical systems. For example, considering 5G scenarios, ultra reliable communication and keeping services running most of the time "*is envisaged as an important technology pillar for providing anywhere and anytime services to end users*" (MENDIS; LI, 2017).

It is important to mention that, although the works considered in the comparison did not propose frameworks for the allocation of SFCs with a focus on availability, other works contemplate the SFC availability.

Araújo et al. (ARAÚJO et al., 2020) propose an SFC placement solution, 'Optional Backup with Shared Path and Shared Function' (OBSPSF), that assigns backup resources only when strictly needed. When an SFC request arrives in the system, OBSPSF attempts to place the SFC that satisfies the connectivity, processing, and availability requirements. If there is more than one solution, the one with the lowest cost is selected. If no solution can satisfy the availability requirement, the backup resources for nodes and links are allocated to increase SFC availability. To find the "backup path", k-shortest paths are calculated offline and are used during the SFC placement. Our proposal differs from OBSPSF in that it is an RL based solution that can run online.

Wang et al. (WANG et al., 2021b) propose a strategy to guarantee the availability of a parallelized SFC based on placing multi-flow backups. Three different placement strategies and an algorithm to map the SFC onto the physical infrastructure are proposed. The placement strategies are based on affinity, which determines if the VNFs of the sub-flows will be placed in the same or different physical machines. Then, the placement algorithm uses different affinity strategies to map the VNFs and the flows into the infrastructure. Again, while there are similarities between Wang et al. (WANG et al., 2021b) and our proposal, Wang et al. only consider fixed redundancy strategies, while we use an RL agent to define to select the suitable candidate node and define the redundancy strategy to meet availability requirements during

the placement process.

Table 1 – Comparison among the related works (Y-yes, N-no, and *-not specified in the work)

Work	Context Information Sensitive	Zero-touch Paradigm	ML Based	Independent of Management Tool	Focus on the SFC availability
(GUPTA et al., 2017)	Y	N	Y	*	N
(SHEN; ZHANG, 2018)	Y	Y	N	*	N
(GUO et al., 2019)	Y	*	Y	*	N
(NGUYEN et al., 2019)	Y	*	N	*	N
(SANTOS et al., 2020b)	Y	Y	N	N	N
(SURIANO et al., 2020)	Y	Y	N	*	N
(HU et al., 2020)	Y	Y	N	*	N
(LANGE et al., 2020)	Y	Y	Y	N	N
(SHAH; ZHAO, 2020)	Y	Y	Y	*	N
(TOUMI et al., 2020)	Y	*	N	*	N
(TOUMI; BAGAA; KSENTINI, 2021b)	Y	*	Y	*	N
(TOUMI; BAGAA; KSENTINI, 2021a)	Y	*	Y	*	N
(TOUMI et al., 2021)	Y	Y	*	Y	N
(CHEN et al., 2021)	Y	Y	Y	N	N
SPIDER (our proposal)	Y	Y	Y	Y	Y

Source: the author (2023).

The framework proposed in this thesis, SPIDER, contemplates all aspects mentioned in Table 1. Our solution uses context information through ML approaches in order to make the SFC placement in an automated way (zero-touch paradigm). In addition, we propose a framework that is not focused in a specific VIM, allowing network managers to adapt our solution according to their infrastructure and the software used. Finally, our solution takes into account the SFC availability as requirement for its placement, while also consider computational and network requirements, placement cost, and energy consumption. In the next chapter, we describe the SPIDER with more details.

4 SPIDER

In this chapter, we describe SPIDER, its architecture and main modules. In Section 4.1, we detail the main requirements and features of SPIDER. Afterwards, in Section 4.2, we describe a simple SFC request scenario, which is useful to understand the SPIDER working. The SPIDER architecture and its components are described in Section 4.3, detailing their role for the SFC placement. We also detailed the Agent module in Section 4.3.4, which is the core of SPIDER and is responsible to make the SFC decisions using artificial intelligence algorithms. The results obtained from this chapter were published in (SANTOS et al., 2022).

4.1 SPIDER REQUIREMENTS

A requirement can be defined as "*a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents*"(PANDEY; SUMAN; RAMANI, 2010). The requirement engineering is one of the key processes in software development (OCHODEK; KOPCZYŃSKA, 2018). It is very important to specify what are the requirements about the software in order to define its main functionalities and to guide the software development process. In this way, we need to specify the main requirements of SPIDER and, consequently, define its main features.

Table 2 presents the SPIDER requirements. A very important requirement of SFC placement systems is that the allocation of resources and the placement of network functions need to be performed dynamically (based on the actual infrastructure condition) and automatically (with the minimum human intervention) (BHAMARE et al., 2016). The automation of SFC placement decisions is crucial to reduce the deployment time and to enable the zero-touch orchestration and management (RECSE; SZABO; NEMETH, 2020). In order to meet these requirements, an SFC management system need to be able to deal with its context information. Context is any information that can be used to characterise the situation of an entity (HONG; SUH; KIM, 2009). Considering the SFC placement, for example, information about the infrastructure can comprise the context. In this way, context-sensitive systems are able to use context information in order to provide more relevant services, adapting the system output according to which is important for the actual context (VIEIRA; TEDESCO; SALGADO, 2011). SPIDER should be **context-sensitive**, it can perceive and react to the dynamic changes in the

environment automatically (REDDY; MURALI; RAJESHWAR, 2019). Then, the framework should be able to collect static and real time information about the network in order to decide the best SFC placement. Context-aware applications can support three different features about the context information: presentation, execution, and tagging (ABOWD et al., 1999). Therefore, our framework should be able to represent internally the environmental information (presentation), to make decisions based on it (execution), and to classify all components of the infrastructure based on the information (tagging).

Table 2 – SPIDER Requirements.

Requirement	**Description**
Be sensitive to the context information	SPIDER must be able to process context information monitored from the physical infrastructure in order to define the placement for an SFC request which meets the availability requirements.
Present learning aspects to define the SFC placement	In order to define the placement for an SFC request, the SPIDER must be able to learn how to define the best placement for an SFC automatically. This dispenses with manually configuring algorithms and allows the framework to adapt to different network conditions.
Be able to consider different metrics during the SFC placement	In addition to the SFC availability, SPIDER have to consider other metrics during the SFC placement.
Be able to deal with different VIM/Management tools	The SPIDER must be able to deal with different network infrastructures and, consequently, to interact with different network management tools.

Source: the author (2023).

Considering that the SPIDER handles context information, other requirement rises. As mentioned by Chen et al. (CHEN et al., 2018b), a smart SFC should be "*adaptive to the changing environment so that it can learn to approximate the optimal SFC orchestration policy with minimal human interference for automation purpose, i.e., the learning aspect*". To process the static and dynamic context information and be able to learn how to process SFC requests, the SPIDER is **based on ML algorithms**. ML algorithms have been used in different fields of society: "*from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones*" (LECUN; BENGIO; HINTON, 2015). Recent researches suggest the use of ML for handling many network problems such as resource usage optimisation, automatic configuration, and automated decision-making (BOUTABA et al., 2018). As we will explain later in this chapter, we can implement different algorithms in the SPIDER to define the SFC placement, but in this work we used the state-of-the-art algorithms, i.e. DL and RL, in order to make the SFC placement automatically in an efficient manner.

Another important requirement of SPIDER is the focus on the SFC availability. Once the

user specifies the level of requirements desired for the SFC, the SPIDER needs to guarantee the level of availability, in order to avoid financial losses due to breach of SLA. However, other conflicting metrics should be considered during the placement decision. For example, to place redundant VNFs in distributed servers may increase the SFC, but also increases the overall energy consumption (LIU; CHENG; WANG, 2020). Therefore, SPIDER **is able to consider different metrics during placement**, provided these have been previously determined and implemented in the placement algorithm. Different metrics can be considered (or optimised) during the SFC placement process: operational cost (CHEN et al., 2018; PEI et al., 2018; YAO et al., 2020), network delay (MARTÍN-PÉREZ; BERNARDOS, 2018; SUBRAMANYA; HARUTYUNYAN; RIGGIO, 2020), resource utilisation (LI et al., 2019; SHANG; LI; YANG, 2018), revenue (XIE; WANG; DAI, 2020; LI et al., 2019), among others. However, different metrics have different weights for different customers. For instance, besides the high service availability, a telecommunications operator may require as little delay as possible, while a small company that is outsourcing its IT services may prioritise a lower cost, even if other metrics may be impacted. The framework proposed in this thesis should be able to consider different metrics for different customers during the SFC placement. As we add redundant VNFs to increase availability, the allocation cost increases, as more computational resources will be required to run the redundant VNFs, which also increases power consumption. Thus, in addition to SFC availability, we consider placement cost and energy consumption as conflicting metrics that need to be considered when defining placement.

Besides the SFC availability is a common concern of network managers, an SFC can be allocated in different distributed scenarios, each one with its requirements and features. For instance, an IoT scenario can be composed of edge devices with limited computing resources (XU et al., 2020), while an Internet Service Provider (ISP) can have distributed data centers with hundred of servers. In addition, different infrastructures can use different SFC management software solutions, in other words, different VIMs. It happens because different VIMs have different hardware requirements (VENTRE et al., 2016), then another desired requirement of SFC solutions is to handle different hardware and software platforms. Against this backdrop, the SPIDER **can be used with different VIMs and/or management tools**. Some previous works proposed SFC placement and orchestration solutions based on specific VIMs, such as the work presented in (LANGE et al., 2020) is based on Openstack[1], while the work presented in (SANTOS et al., 2020b) is based on Kubernetes. However, the SPIDER is not planned to deal

[1] <https://www.openstack.org/>

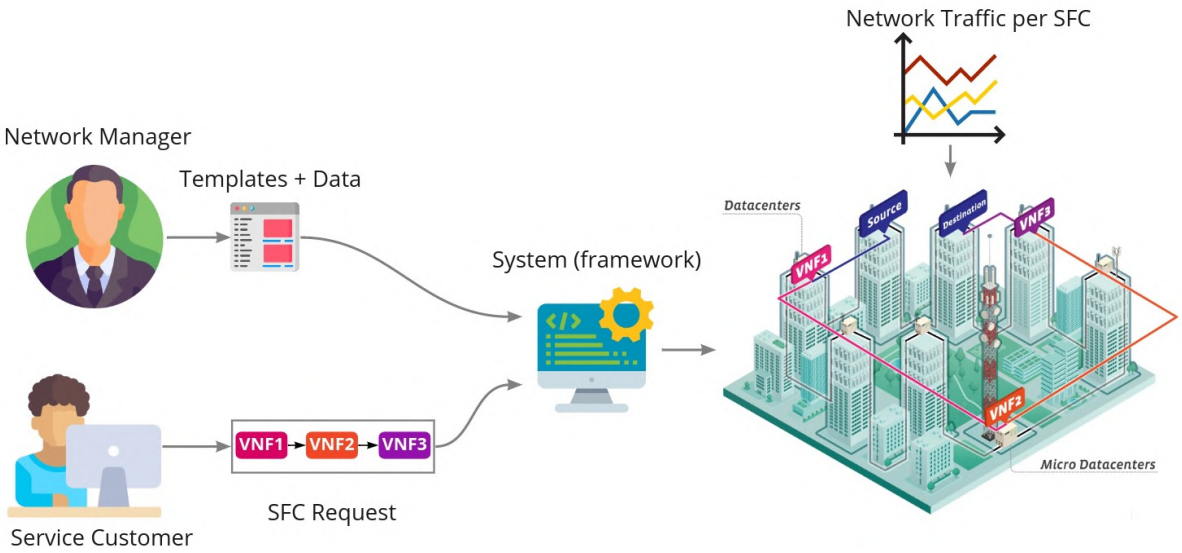
with a specific VIM, and new modules could be created in order to address API based VIMs. Therefore, network managers could use SPIDER with VIM that manages their infrastructure, avoiding significant changes in the software and architecture to which they are accustomed.

Before describing how SPIDER works, in the next section, we describe an example of an SFC request and what kind of information is needed for the placement.

4.2 SFC REQUEST EXAMPLE

Figure 9 shows a simple example of SFC placement and management scenario. Firstly, the network manager provides predefined SFC templates that the customers could require. The SFC template has a list of VNFs that composes the SFC, their order, and SLA information of the SFC. An example of predefined template can be an SFC for security, composed of firewalls, Deep Packet Inspection (DPI), and IDS (XU et al., 2018).

Figure 9 – Example of SFC system operation



Source: the author (2023).

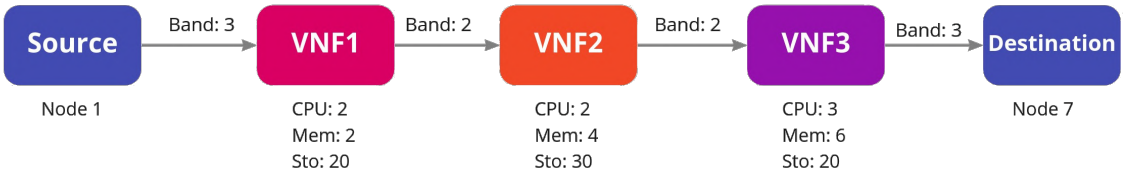
The network manager also provides static information about the network infrastructure that can be useful for the SFC placement and management such as the computing cost of allocate a VNF, the mean energy consumption of a server, the total bandwidth capacity of a link, the geographical location of a data center, the SFC templates, among others. This information is needed for the SPIDER operation in order to define in which servers the SFC will be allocated.

Since the SPIDER is operating, the customers can send SFC requests, either describing

the VNFs desired and their order or using a predefined SFC template (provided by the network manager). The SFC request also includes SLA requirements of user, such as minimum delay, maximum cost, and availability levels desired. Afterwards, the system, which implements the SPIDER, makes the SFC placement in the infrastructure. Then, each SFC allocated handle different network traffic, according to the customer service.

In SPIDER, the customer sends the SFC request through an API (or indirectly through a graphic user interface which access the API). We consider the SFC request as illustrated in Figure 10. The SFC request defines two fixed nodes in the infrastructure, representing the source and the destination of the SFC, similar to the representation presented in (WANG et al., 2021a). The source node refers to the node through which the traffic enters the SFC (for example the gateway of a data center), while the destination node indicates the last one that receives traffic from the SFC (e.g. the database server that stores the data referring to the SFC).

Figure 10 – Example of generic SFC request.



Source: the author (2023).

The SFC request also specifies a list of VNFs, each one having specific computing requirements according to the function it performs. For example, simple functions like a rule-based firewall used for filtering traffic usually demands less processing resources than a IDS that analyses traffic using advanced regular expressions and algorithms (LANGE et al., 2020).

We also establish a list of virtual links that connect the source, the list of VNFs, and destination nodes. Each virtual link defines bandwidth requirements for each VNF. This representation allows to model the data transformation, due to events such as traffic compression, where the required bandwidth changes at the egress after VNF processing.

It is important to highlight that other information could be considered in the SFC representation. In addition to the computing and storage requirements, it is possible to define requirements based on VNFs affinity, geolocation, node type, among others. With regard to the virtual links requirements, one may also specify requirements related to delay, type of

link (wired and wireless), among others. Such information can be easily described as a JSON document and expanded to define new requirements for both VNFs and virtual links.

The SFC placement task that SPIDER performs can be defined as selecting physical nodes to place the VNFs and physical links to place the virtual links. The selection of physical nodes takes into account the resources requirements of the VNF in terms of CPU, memory, and storage; while the selection of physical links considers the bandwidth requirements of the virtual links. During the SFC placement, the SPIDER must address the availability requirement specified by the user. To do this, the SPIDER defines the redundancy strategy for each VNF specified in the request. The most simple way to increase the SFC availability would be to add replica for all VNFs, however this strategy increases the placement cost and the energy consumption, since more resources from the server would be required and an extra processing is required to place the SFC. Therefore, in this work, we consider minimising the placement cost and energy consumption while the SFC availability is guaranteed.

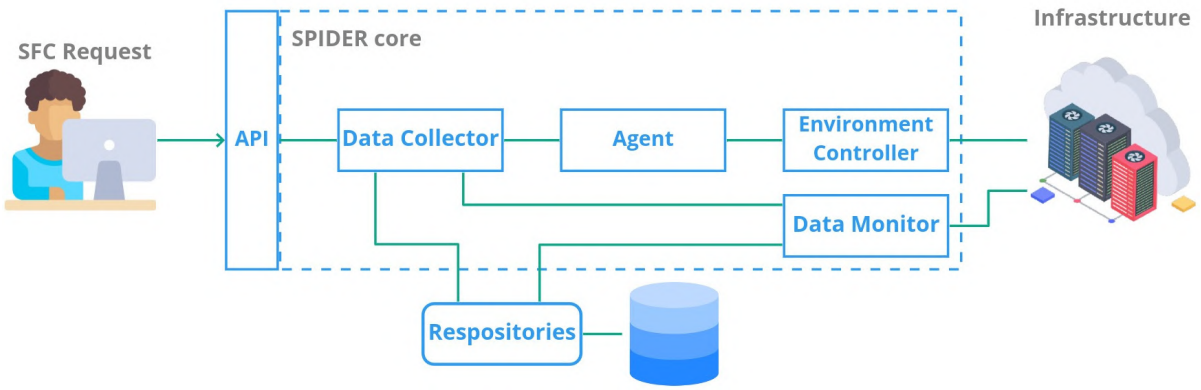
In the next section we describe the architecture of SPIDER, its modules, and how it processes the SFC requests from users.

4.3 SPIDER OVERVIEW

To address SFC requests, SPIDER adopts the high level view illustrated in Figure 11. It offers an API that exposes all SPIDER endpoints to external applications and interfaces. Therefore, this makes it easy to create different user interfaces (e.g. for web, mobile, and desktop applications) as they share the same API. The architecture also integrates a database that stores SPIDER infrastructure information (nodes and links status), the VNF catalog, and data describing placed SFC. This information can be accessed through data repositories, that provide a set of functions to create, retrieve, update, and delete entries within.

The heart of the proposed SPIDER architecture is the core module. It consists of a set of functions that, in fact, provides the main SPIDER services. This core module embeds several others, such as one that collects and monitors information about the infrastructure; a second module that specifies and executes the SFC placement based on user requirements; and a third module that performs the placement across the underlying infrastructure. We explain the core modules with more details in Section 4.3.3 The SPIDER core leverages functions from the repository to continuously gather information representing the state of the infrastructure. This is important and useful for the definition and execution of the SFC placement. Note that

Figure 11 – SPIDER Framework overview.



Source: the author (2023).

all the supported functions of repositories and SPIDER core modules can be accessed through the API. The next subsections detail each of the above modules.

4.3.1 Repositories and Data Models

SPIDER repositories consist of a set of Python scripts that interacts with the database to create, retrieve, update, or delete data of relevant SPIDER components. Table 3 shows the main Python scripts that compose the repositories.

Table 3 – Repository scripts.

Python Script	Functionality
Node Repository	Manages the information about the physical nodes present in the infrastructure. This script is also used by the Infrastructure repository.
Link Repository	Interacts with the database to manage information about the physical links from the infrastructure. It is used by the infrastructure repository.
Infrastructure Repository	Interacts with the database to manipulate information about the infrastructure where the SFCs are placed.
SFC Request Repository	Manipulates the information about the SFC already placed in the infrastructure. Since the VNFs and virtual links placed are related with the physical infrastructure, this script uses the Infrastructure Repository.
VNF Template Repository	This script interacts with the database to manage the information about the VNF templates that are available in the SPIDER

Source: the author (2023).

We create a set of scripts to manipulate physical infrastructure information where the SFCs are placed. Each repository provides functions to create, retrieve, update, and delete data of specific system components. As database technology, SPIDER uses MongoDB which is a

NoSQL and document-based database. MongoDB, differently from traditional relational SQL databases, stores data as a document (usually in JSON format) which increases its flexibility, enabling it to store new data with prior knowledge of its format. In addition, MongoDB has a superior performance than a relational MySQL database, considering the basic operations of creating, retrieving, updating, and deleting (GYŐRÖDI et al., 2015).

The Node Repository manages all information about the physical nodes present in the infrastructure. Therefore, when the information about one or more node needs to be updated (for instance due to changes in the amount of available resources) or retrieved from the database, the Node Repository is invoked. Table 4 summarizes the information about the physical node that we store in the database.

Table 4 – Physical node data model.

Database Field	Data Type	Description
ID	Int	Node ID
Name	String	Node name
Status	String	Node status (operational or failed)
Resources	JSON	Total of resources of the node
Available_resources	JSON	Amount of available resources of the node
Cost	Float	Cost of the node in $
MTTF	Float	MTTF value of node (in hours)
MTTR	Float	MTTR value of node (in hours)
Availability	Float	Availability value of node (in %)
Energy	Float	Energy consumption of node (in kWh)
Metadata	JSON	Additional information about the node
Capabilities	JSON	The VNF types that can be placed in the node

Source: the author (2023).

The basic information of a physical node are its ID, name, and status (where we consider as operational or in fail state). We store two different attributes regarding node resources: the overall resources of a node and its current amount of available resources. Both fields are in JSON format and contain node information describing its CPU, memory, and storage. The difference being that the former indicates the total (used and unused) available resources of a node, whereas the latter lists only the amount of currently available ones.

When new SFCs are placed, they consume node resources and consequently there is a need to update information related to available resources. This information is needed to guide

the SFC placement. We store in the database values that describe availability parameters for different SPIDER components. These entries include the MTTF, the MTTR, and the availability metric itself. The values about the physical devices are usually provided by the manufacturers, while the values about the software can be measures through fault injection. However, these values need to be provided by the customer. The availability value can be estimated based on the following Equation 2.2.

In addition to availability constraints, the SPIDER framework considers energy consumption as an important aspect for its resource allocation decision-making. As a result, it stores information on the energy consumption of nodes. This is used in calculating the operational costs of placing a VNF at a physical node. As mentioned in Section 4.2, the MTTF, MTTR, and energy consumption values must be provided by the network operator when the network is built into the SPIDER. Additionally, we have a field called metadata to take into account any additional information about the nodes, such as manufacturer name, year of acquisition, among others.

Since we are using MongoDB, and the data is saved as a JSON document, should a network operator not using any additional metadata, an empty document can be inserted in the database. The last field of Node Repository is the node affinity, which defines a list of VNFs that this node is able to serve. This can be used to state that not all nodes may host all network functions due to performance, compatibility or simply policy constraints, for example.

The Link Repository manages data that describe physical links. Table 5 shows all fields related to links that one may save in the database. We store the node ID and the IDs of two physical nodes that are connected by this link. We save the propagation delay needed to transmit data between the two nodes connected by a given link (in the milliseconds unit), as well as the cost of placing a virtual link in the link. These values must be provided by the user. Similarly to the node data model, we also save the total and current available amount of resources for each link.

In order to handle information representing the overall physical infrastructure, we propose the Infrastructure Repository. It uses both previously described Node and Link Repositories. Prior to a placement operation, the current status of the nodes and links must be taken into account. Next, the Infrastructure Repository can be used to select relevant information and generate a graph reflecting the current status of the network. Table 6 illustrates the data model of the Infrastructure. We save the ID of the infrastructure and two different lists: the list of nodes and links IDs. Then, using the Node and Link Repositories, one can select all

Table 5 – Physical link data model.

Database Field	Data Type	Description
ID	Int	Link ID
Source_node	Int	Source node of link
Destination_node	Int	Destination node of link
Delay	Float	Link delay (in milliseconds)
Cost	Float	Link cost (in $)
Resources	JSON	Overall resources of link
Available_resources	JSON	Current resources of link

Source: the author (2023).

information about the nodes and links of a specific infrastructure.

Table 6 – Infrastructure data model.

Database Field	Data Type	Description
ID	Integer	Infrastructure ID
Nodes	JSON	List of nodes
Links	JSON	List of links

Source: the author (2023).

The previous repositories dedicate themselves to the describing the physical infrastructure, in other words, the physical level managed by SPIDER. In addition, there two repositories that deal with the logical level type of resources. First, there is the SFC Request Repository which manages data for already placed SFCs. It also stores information about the physical nodes and links in which the VNFs and virtual links are placed. Then, the SFC Request Repository uses the Node and Link repositories to load their information. Table 7 shows the fields of SFC request data model. We save the SFC request ID and name (for example, it could be the name of the overall service). We also store the ID of source and destination nodes specified in the SFC request; and the list of VNFs and virtual links. Each VNF of the list has information about the computation requirements (CPU, memory, and storage), number of replicas placed (to ensure the overall SFC availability), and the nodes ID list where the replicas were placed. For each virtual link, we save the list of links where the virtual link is placed. We adopt a list because more than one link could be needed to connect nodes where two sequential VNFs are placed.

Finally, there is the VNF Template Repository. This repository maintains the VNF template

Table 7 – SFC request data model.

Database Field	Data Type	Description
ID	Int	SFC request ID
Name	String	SFC request name
Source	Int	ID of the source node of SFC
Destination	Int	ID of the destination node of SFC
VNFs	JSON	List of VNFs placed
Virtual links	JSON	Virtual links placed

Source: the author (2023).

data that is used to define characteristics of a VNF. Therefore, as shown in Table 8, we save the ID and name of the VNF template (for instance, firewall, an IDS, load balancer, etc). We also store the computing resources required by the VNF. It is interesting to differentiate functions that require more computational resources, such as applications that run machine learning models, from applications that demand less resources, such as, a simple load balancer or a firewall. Similar to the Node data model, we also store availability related parameters, including MTTF and MTTR of a VNF. These values are required to estimate VNF availability, which is then copied to the database. The last field of VNF template data model is the path pointing to where all files needed to create the Docker image for a given VNF type are located. In other words, this template includes pointers to files that contain the source code, libraries, dependencies, tools, and others needed for an application to run. For more details, see Subsection 7.1.

Table 8 – VNF template data model.

Database Field	Data Type	Description
ID	Integer	VNF template ID
Name	String	VNF template name
Resources	JSON	List of computing resources required by the VNF
MTTF	Float	MTTF value of VNF
MTTR	Float	MTTR value of VNF
Availability	Float	Availability value of VNF
Files_path	String	Path where the files about the VNF application are saved

Source: the author (2023).

We assume that the traffic about the SFCs placed in the infrastructure is stored in the MongoDB and are managed by the SFC Traffic Repository. Table 9 illustrates the data model

regarding the SFC traffic. We store an ID of the traffic and the ID of the SFC that this traffic is associated with, both are of type Integer. We also store the physical links of the infrastructure that are impacted by the traffic. The list of links can be easily retrieved from the database using the SFC Request Repository, as illustrated in Table 7, and since it is a list, we store it in the JSON format. The last field is the traffic data itself. MongoDB provides time series collection[2] that store sequences of measurements over a period of time efficiently. Time series collections store measurements, that are composed of at least a timestamp and the metric value. Then, we save the timestamp and the respective traffic that arrives to be processed in by the SFC.

Table 9 – SFC traffic data model.

Data field	Data type	Description
ID	Integer	Traffic ID
SFC ID	Integer	SFC ID the traffic is associated with
Physical Links	JSON	List of physical links where this traffic will pass through
Traffic	Time series	Traffic data

Source: the author (2023).

Both the API and SPIDER core modules use repositories to manage data of the SPIDER framework. The next subsections describe both modules.

4.3.2 SPIDER API

The SPIDER API controls access to all functionalities of the framework by user applications. The API exposes the resources implemented in SPIDER Core and different user interfaces can be developed to access them. Table 10 summarizes all endpoints of the API.

The endpoints of the SPIDER API fall into three groups. The first group of endpoints is related to the VNF templates. As shown in Table 10, Endpoint 1 (POST method) is used to create a new VNF template, where the user must specify the data for the new template, illustrated according to Table 8. Endpoints 2 and 3, which are GET methods, are used to retrieve information about the VNF templates already saved in the system: the former retrieves data for all VNFs, while the latter obtains information about a specific VNF, specified by its ID. Endpoint 4, which is a PUT method, is used to update the information about a specific

[2] <https://www.mongodb.com/docs/manual/core/timeseries-collections/>

Table 10 – API endpoints.

#	Endpoint	Method	Description
1	/vnf	POST	Create new VNF template in the system. The user must pass the VNF data as a JSON
2	/vnf	GET	Get information about all VNFs templates saved in the SPIDER
3	/vnf/<id>	GET	Get information about a specific VNF template
4	/vnf	PUT	Update an existing VNF template in the system. The user must pass the new VNF template data as a JSON
5	/vnf/<id>	DELETE	Delete an existing VNF template in the system. The use must specify the ID of VNF to be deleted
6	/infra/	GET	Get information about all infrastructures managed by the SPIDER
7	/infra/<id>	GET	Get information about a specific infrastructure according to its ID
8	/infra/	PUT	Update the information of an infrastructure saved in the system
9	/infra/<id>	DELETE	Delete the information of an infrastructure saved in the system according to its ID
10	/sfc_request	POST	Create new SFC request in the system. The user must pass the SFC request data as a JSON
11	/sfc_request	GET	Get the information of all SFC requests already created in the system
12	/sfc_request/<id>	GET	Get the information about a specific SFC request created in the system
13	/sfc_request/	PUT	Update the information about a specific SFC request created in the system. The user must pass the new information as a JSON
14	/sfc_request/<id>	DELETE	Delete an SFC request already created in the system according to its ID

Source: the author (2023).

VNF template. A user should provide a JSON with the information to be updated for the VNF template. The last endpoint related to the VNF template, Endpoint 5, is a DELETE method and is used to delete an existing VNF template. A user needs to specify the ID of the template to be removed.

The second group of endpoints is concerned with the infrastructure managed by the SPIDER framework. However, unlike the VNF template methods, the current API implementation does not provide a method for creating an infrastructure in the system. A user needs to provide information about the physical nodes of the infrastructure, as well as initiate the monitoring software to collect information about the infrastructure (see Section 4.3.3). Then, the infrastructure information (specified in Tables 4, 5, and 6) is generated and saved in the database by the SPIDER framework, as opposed to being created by the user through an API call. There are two endpoints (6 and 7), which are GET methods, used to gather information about the infrastructure. A user may also call the Endpoint 7 to obtain information about an infrastructure, specifying an infrastructure by its ID. Endpoint 8 is used to update the information about the infrastructure. However, it is important to point out that information on physical nodes and links cannot be updated through the API, as this information is saved and updated in the database through the Data Monitor module and monitoring tools (please check the Section 4.3.3). Finally, Endpoint 9 is used to delete an infrastructure. This method removes all information about the infrastructure from the database. However, the user must prior stop the monitoring tool from running in the physical nodes of the infrastructure.

The third group of endpoints is related to the SFC request. Similar to the previous groups, there are two endpoints (11 and 12), which are GET methods. These are used to retrieve information about the SFC request already placed in the infrastructure. Endpoint 11 gathers

information about the complete list of SFCs placed previously in the system, while the Endpoint 12 retrieves information about a specific SFC by its ID. Endpoint 13, which is a PUT method, is used to update the information of an SFC. However, only the information about the ID and name of an SFC can be updated, because the information about the source, destination, VNFs list, and virtual link list (as detailed in Table 7) are defined during the placement by the SPIDER framework. Endpoint 14 (DELETE method) is used to delete an SFC request given its ID as input. When a user calls this endpoint, the VNFs placed in the system will be removed from the physical infrastructure, as well as the used virtual links. Endpoint 10, listed in Table 10, is used to create a new SFC request, It implements a POST method. In Appendix B we detail the information about a new SFC request in JSON format.

As mentioned previously, the API provides the main functionalities of SPIDER framework to the interfaces and different applications. In the next section, we describe the SPIDER Core, which is the part of the framework that implements these functions.

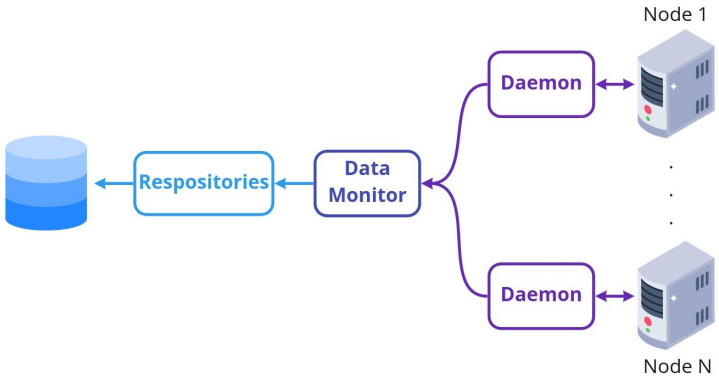
4.3.3 SPIDER Core

The SPIDER core is based on the MAPE-K loop described in Section 2.3 and its components assumes roles of different steps of the loop. Foremost, the user must set up the physical infrastructure to be monitored by the SPIDER framework. Under the current implementation, as shown in Figure 12, we implement a daemon which is executed on all nodes of the physical infrastructure. In addition, we implement a Data Monitor module that obtains information about all nodes and stores them in the database through the repositories (see Subsection 4.3.1). Considering the MAPE-K loop (Figure 5) the Data Monitor plays the role of a monitor step, while the Daemon is the sensor.

The daemon module is presented through an API capable of collecting the updated information describing the computing and network status of a node. The daemon is able to collect updated information about computing and network resources used by a node. When a node is included to be monitored by SPIDER, the user must specify a set of information about it, such as ID, name, location, MTTF, MTTR, availability, cost, and energy consumption. This information is used by SPIDER during the SFC placement definition. For more information about the node configuration, please see Appendix C.

A user also provides a list of links connected to the node. In the JSON example, we consider only a single link. As part of link information, we consider the link ID, the interface of the

Figure 12 – Data Monitor module and daemon.



Source: the author (2023).

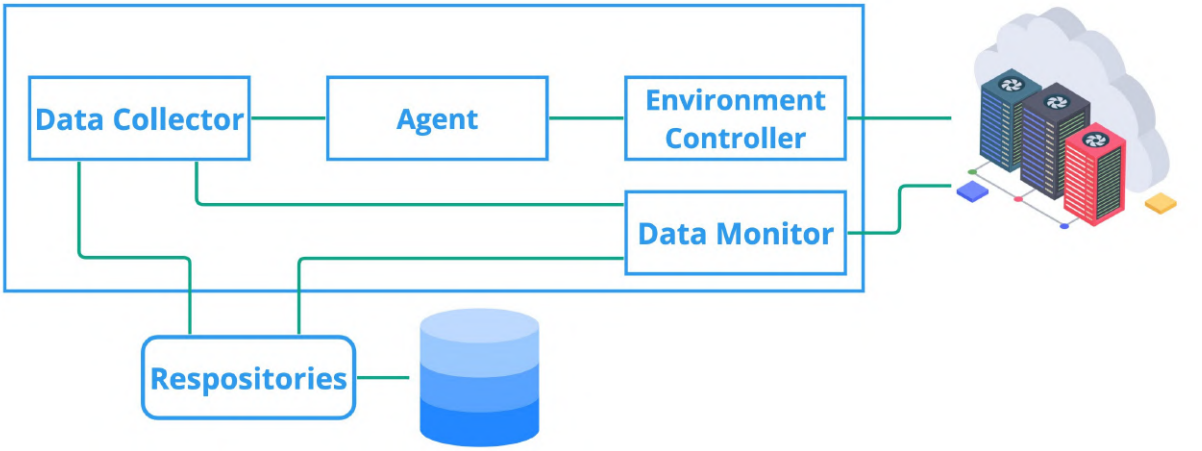
node connected to this link, the destination, and the network resources of the link. Please note that a link is only a means of connecting a physical node to another one (it could be a wired or wireless link). In the destination field, the user defines the ID, name, latitude, longitude, port, and IP of the other connected node. In the resources field, the user defines the overall resources for each link, including delay (in milliseconds), bandwidth (in Mbps), and the cost of this link (in $). Such link information is interesting to differentiate nodes that are located geographically close, with a smaller delay; from nodes located geographically distant, where the delay and allocation cost may be higher.

Considering all information collected and defined for each node of the physical infrastructure, the Data Monitor module communicates with all daemons running in the infrastructure to collect and update information about nodes and links (as illustrated in Figure 12). For this, the user needs to define the list of IPs of all nodes in the physical infrastructure. Then the Data Monitor can invoke the API of the daemon running in each node, aggregate all information and use the repositories to save this information in the database.

Assuming that the updated information about the infrastructure is saved in the database, SPIDER can perform the SFC placement, taking into account the submitted requests. The main functions of SPIDER framework are implemented in its core, and they are illustrated in Figure 13. When a customer sends an SFC request through the API, it is forwarded to the SPIDER core, which processes the request, and then places the SFC.

The first module of SPIDER core is the Data Collector, which is developed as an API using the Flask framework. The Data Collector module is responsible for gathering updated information about the infrastructure to be considered during the placement. The Data Collector can gather the infrastructure information in two ways, both configurable by the SPIDER user:

Figure 13 – SPIDER core overview.



Source: the author (2023).

1) through the repositories and 2) by calling the Data Monitor module.

On the one hand, when the Data Collector uses the repositories, it calls the Infrastructure Repository to obtain updated information of the infrastructure. It is saved by the Data Monitor module, as shown in Figure 12. The Data collector calls the API to have access to updated infrastructure information using the Infrastructure Repository (which uses Node and Link Repositories). On the other hand, when the Data Collector is configured to gather information directly through the Data Monitor, the information is collected directly from Daemons running on all nodes, as illustrated in Figure 12. However, instead of saving the information in the database, Data Monitor sends the information about nodes and links to the Data Collector. In this case, the information about the infrastructure may or may not be saved in the database. If the SPIDER user wishes to keep historical information about the infrastructure, it can use the repository to save it. However, this functionality is not implemented yet in the current version of SPIDER, and we plan to support it in future versions.

As the information is returned by the API, an undirected graph is created to model the physical infrastructure, which is a Python package for creation, manipulation, and study of graphs. The graph is used to define the placement for the SFC request. Then, the infrastructure graph and the SFC request information are merged into a JSON representation, and forwarded to the Agent module.

The Agent module defines the placement for the SFC request. To do this, the Agent considers the updated infrastructure graph and the availability requirement defined by the customer. One can use different algorithms and solutions to define the SFC placement. The

Agent module has functions associated with the analysis step from MAPE-K loop (Figure 5). This is due to the fact that the Agent analyses the information forwarded by the Data Collector and defines the placement decision.

SPIDER uses an RL agent that was trained to obtain the best placement solution, when seeking to add redundant VNFs to achieve a required availability level (defined by the customer) and minimize the operational costs (represented by the energy consumption of the nodes where the VNFs instances are placed). Therefore, the Agent defines in which nodes each VNF and the replicas have to be placed in the infrastructure. We describe the RL agent implementation with more details in Chapter 6. In addition, DL models are used for traffic prediction in order to estimate the future bandwidth available on the physical links. This information is useful to select the paths to place the virtual links that connect the VNFs from the SFC request. We describe the DL models with more details in Chapter 5. However, it is important to mention that as we implemented the Agent module as an API, we can easily use another algorithm as the underlying SFC placement.

Appendix D illustrates how the Agent module structures the placement decision information into a JSON representation, which is forwarded to the Environment Controller, which interacts with the tools that, in fact, manage the VNFs and perform the enforcement of the actual placement in the physical infrastructure. In this way, the Environment Controller plays the role of execute step from the MAPE-K loop, responsible to make changes in the system status. Once the Agent defines the placement decision for an SFC request and forward it to the Environment controller, there is just to possible situations: 1) place the SFC into the infrastructure or 2) discard the SFC request, in case that the SFC requirements not being met. In this way, the planning phase from MAPE-K is not necessary, because SPIDER does not perform any complex planning action in those situations. However, for future work, we can consider a planning step, where when an SFC request is rejected, we can plan migrations and/or reallocations of the previously allocated SFCs, in order to allocate the new SFC request.

The Environment Controller module submits HTTP requests to the API of these tools, indicating the placement configuration defined by the Agent module for the SFC request. The implementation of the Environment Controller module depends on the management tool used in the infrastructure. For instance, the Tacker[3] tool considers that the information about the SFC needs be defined following the TOSCA standard (BINZ et al., 2014). In this case, the Environment controller needs to translate the JSON data sent by the Agent into a TOSCA

[3] <https://wiki.openstack.org/wiki/Tacker>

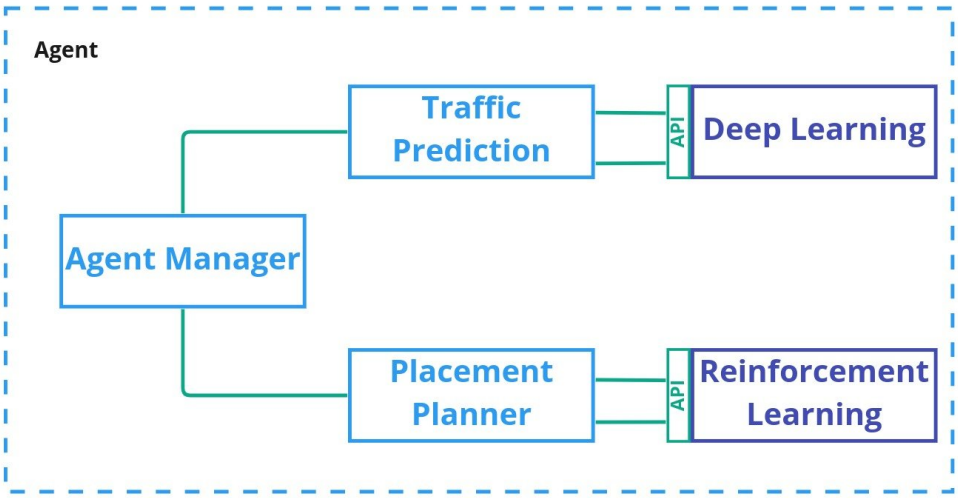
document.

The Agent model is the "smart" part of SPIDER. It is responsible to define the placement for each SFC request from user, taking into account the context information from the infrastructure and the user requirements. In the following section, we describe in more details the Agent module of SPIDER.

4.3.4 Agent Module of SPIDER

Figure 14 shows the SPIDER Agent in more details. The agent is responsible for two tasks: predict traffic based on the SFCs already placed in the infrastructure, and define the placement for a specific SFC request. We define a module named Agent Manager, which is responsible for receiving the requests from other modules. Then, the Agent Manager calls two other modules: Traffic Prediction and Placement Planner.

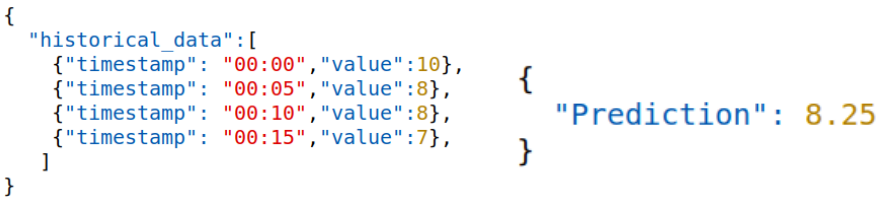
Figure 14 – Agent module description



Source: the author (2023).

Before define the best SFC placement, SPIDER predicts the traffic taking into account the SFCs already placed in the infrastructure. We use this information to estimate future information about the links bandwidth with the purpose to place new virtual links. In this way, we defined this problem as a time series prediction, where the Agent module uses the SFC Traffic Repository to collect the historial traffic for all SFCs placed and the Traffic Prediction module is used to predict the traffic of each SFC. Since the traffic of an SFC has information on the links that have been allocated, it is possible to estimate how much bandwidth will be available on each link based on the traffic predictions. Thus, we consider the predicted

Figure 15 – JSON examples for the Traffic Prediction module



(a) (b)

Source: the author (2023).

bandwidth and define it as weight in the infrastructure graph. Therefore, we consider the prediction data when choosing the physical links to allocate the SFC virtual links.

Several techniques can be used for traffic prediction such as autoregression Moving Average (IQBAL et al., 2019), Fourier-Series based forecasting method (YAO et al., 2020), RBM (SUN et al., 2019), among others. In this work, we use DL models for the traffic prediction since, more recently, a combination of DL techniques and new data sources have emerged that show promising results in mobile traffic prediction (NAREJO; PASERO, 2018; HUA et al., 2018; QIU et al., 2018). Then, considering network traffic divided into time slots, recurrent neural networks, such as LSTM and GRU, could be used to predict the traffic for the next time slot taking into account the historical data from the previous time slots. In Chapter 5, we describe the DL models used for traffic prediction in this thesis. We present the models architecture and compared with other strategies from the literature.

Since different techniques could be used in the Traffic Prediction module, it should call the implemented technique through a rest API, in order to define a common interface for all techniques. Figure 15 shows a JSON example of data for the Traffic prediction module and the response expected. The historical data (Figure 15a)) is defined as a list of data, each one with the timestamp and the respective traffic, while the response JSON (Figure 15b)) contains the prediction about the historical data.

The Placement Planner is another module of Agent that is responsible for selecting a set of nodes to place the VNFs for the SFC requirement. To do this task, the Placement Planner module can use many techniques: Integer Linear Programming (ILP) (MOUALLA; TURLETTI; SAUCEZ, 2018)(SUN et al., 2019), genetic algorithm (TAVAKOLI-SOMEH; REZVANI, 2019), Markov models (XU et al., 2018)(ZHANG et al., 2019), among others as mentioned by (BHAMARE et al., 2016) and (MIRJALILY; ZHIQUAN, 2018).

In order to implement the Placement Planner module, we use RL which was highlighted by

Souza et al. (SOUZA; DIAS; FERNANDES, 2020) as a good alternative to determine the optimal SFC placement for a given workload in real time. Indeed, RL has been used in several works in the literature for the SFC placement (KHEZRI et al., 2019; GUO et al., 2019; LUO et al., 2019; CHAI et al., 2019). As mentioned in the Section 2.5, a RL agent learns how to perform a task interacting with an environment, receiving positive and/or negative rewards. Therefore, in order to train the RL for the SPIDER, the environment representation is defined based on the information gathered by the Data Collector module (Figure 13), and the reward function could be defined based on the SFC placement success, for example. However, this representation could be adjusted according to the needs of the network manager.

Depending on the amount of available resources from the physical nodes, the RL agent defines in which each VNF will be placed and the redundancy strategy in order to meet the availability requirements. After defining the best nodes to place the VNFs, the links are selected in order to define the virtual links between the VNFs. The Placement Planner uses the information about the traffic prediction and information about the links, such as the location of data centers in order to estimate the delay. Then, a weight is calculated for each link and an algorithm to find the shortest path (e.g. Dijkstra) is used to define the shortest path which connects the servers selected.

It is important highlight that we consider RL and DL modules in SPIDER as presenting in the Figure 14, but this does not require that these modules be implemented together with the framework. For example, these modules can be located on the cloud and accessed through Rest APIs, or provided by other public services such as Amazon SageMaker[4], Google cloud[5], and Azure Machine Learning[6]. This allows network managers to use their own models in the framework, as well as outsource this service and pay according to usage.

Since the Agent module is the most important and complex of SPIDER, we will describe its implementation with more details in the next chapters. Firstly, we will describe the use of recurrent neural networks for the traffic prediction in Chapter 5. These networks are important for the development of the Traffic Prediction module of Agent, as illustrated in Figure 14. Afterwards, we describe in Chapter 6 the use of RL for the SFC placement. We conduct a study and propose algorithms to use RL to define the SFC placement in order to meet the availability requirements. The RL agent is used in the Placement Planner module of Agent (Figure 14).

[4] <https://aws.amazon.com/pt/sagemaker/?c=ml&sec=srv#sm_studio>
[5] <https://cloud.google.com/products/ai?hl=pt-br>
[6] <https://azure.microsoft.com/pt-br/services/machine-learning/>

Finally, in Chapter 7, we present a proof of concept of SPIDER, by implementing it using the Kubernetes. The VNFs are deployed as containers, and we use Kubernetes mechanisms to allow the communication among them. We also present experiments in order to show the performance of SPIDER in a real scenario.

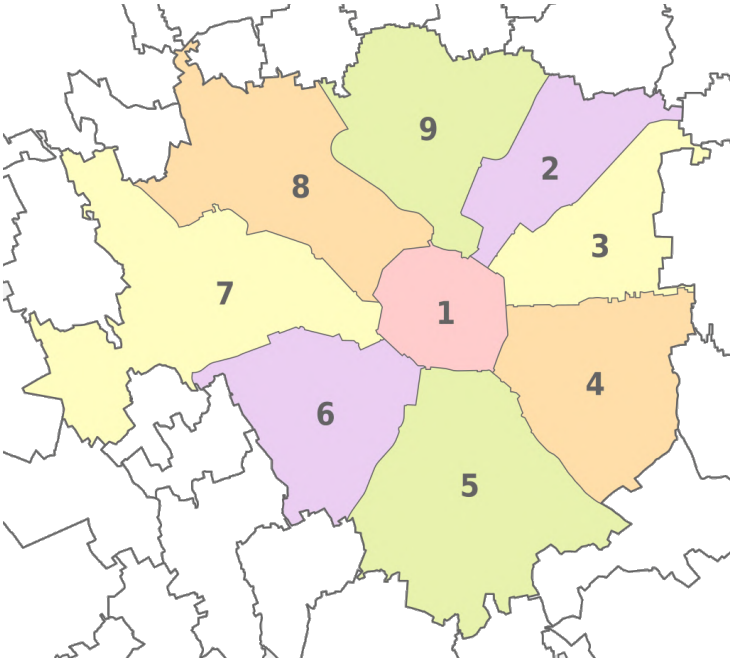
5 TRAFFIC PREDICTION FOR SFC PLACEMENT

In this chapter, we present DL models that are used to implement the Traffic Prediction module of the SPIDER, as shown in Figure 14. We compare the performance of two recurrent units types - LSTM and GRU - for predicting mobile Internet traffic using two months of Telecom Italia data for the metropolitan area of Milan. K-Means clustering was used *a priori* to group cells based on Internet activity and the Grid Search method was used to identify the best configurations for each model. The predictive quality of the models was evaluated using Root Mean Square Error (RMSE). We find variations in performance across clusters within the city. Overall, the LSTM outperformed the GRU in our experiments. These models can be used in the Traffic Prediction module (Figure 14) of SPIDER framework. The results obtained from this chapter were published in (SANTOS et al., 2020) and (SANTOS et al., 2020a).

5.1 DATASET

The metropolitan area of Milan is located in northern Italy and consists of nine different municipalities (see Figure 16). Milan is the largest metropolitan area in Italy and one of the ten most populous in the European Union (EUROSTAT, 2020).

Figure 16 – The Milan Metropolitan Area.



Source: adapted from (GUARINI; MAGLI; NOBOLO, 2018).

In this study, we use the Telecom Italia dataset for Milan from the *Big Data Challenge* (BARLACCHI et al., 2015). The dataset is organized into 10,000 cells (100 x 100) comprising over 10 million user activity logs, each related to a particular cell. The dataset has log data for two months (62 days) from 1 November 2013 to 1 January 2014 (HUSSAIN et al., 2019). Although this dataset was collected between 2013 and 2014, it still proves to be quite valuable for researchers exploring mobile traffic prediction and it has been used in a number of recently published articles (see, for example, (MEDEDOVIC; DOUROS; MÄHÖNEN, 2019; ZHANG et al., 2020; AMIN; CHOI, 2020)). The Telecom Italia dataset adopted in this study in fact is one of the few telecommunication datasets that are publicly available in contrast to the large number of datasets that are typically accessible to a restricted number of researchers under Non-disclosure Agreements (NDAs), or by third parties that have a contractual relationship with telecommunication providers.

The log activity is structured as Call Detail Records (CDRs) on the following activities: (i) incoming and outgoing voice calls, (ii) Short Message Service (SMS) messages, and (iii) Internet activity. A CDR is generated every time a user starts or finishes a voice call, sends or receives an SMS, and starts or terminates an Internet session (the data is recorded if the connection takes more than 15 minutes, or more than 5 MB is transferred during the session). In this thesis, we specifically focus on predicting Internet traffic.

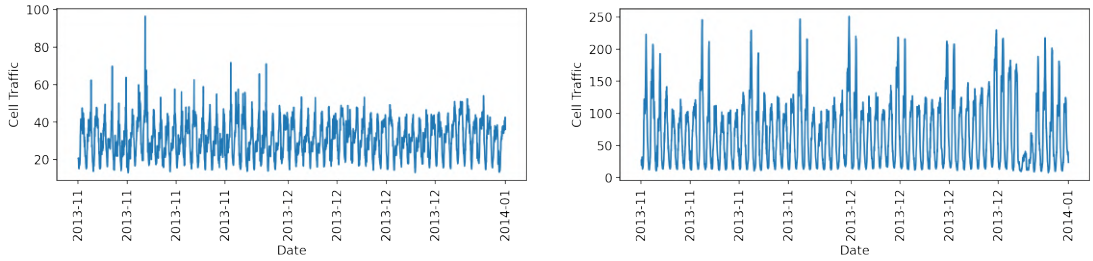
As the dataset has periods with no Internet traffic (e.g., a few minutes at night where there are no records), we aggregate all CDRs for Internet traffic into 30-minute periods. Consequently, we have 48 records per day related to Internet traffic. We use a sliding window strategy, which is a common strategy for time series prediction using neural networks (FRANK; DAVEY; HUNT, 2001), with a window size of four time periods thus we use the previous two hours to predict the Internet activity of the subsequent 30 minutes.

To create the training and testing datasets, the original dataset is divided in two parts: the first 80% of the time series for training and the last 20% for testing (PRAJAM; WECHTAISONG; KHAN, 2022). We also normalized these datasets to the [0,1] range to facilitate the training of DL models, since their parameters are very small, close to zero.

5.2 TRAFFIC PREDICTION PIPELINE

Analysing the traffic of the cells present in the Telecom Italia dataset, we noted that there are different traffic patterns. For example, Figures 18(a) and 18(b) show the traffic of the cells

Figure 17 – Number of Internet activities of cells 1 and 1000 in the Telecom Italia dataset.



(a) Cell 1 Traffic (b) Cell 1000 Traffic

Source: the author (2023).

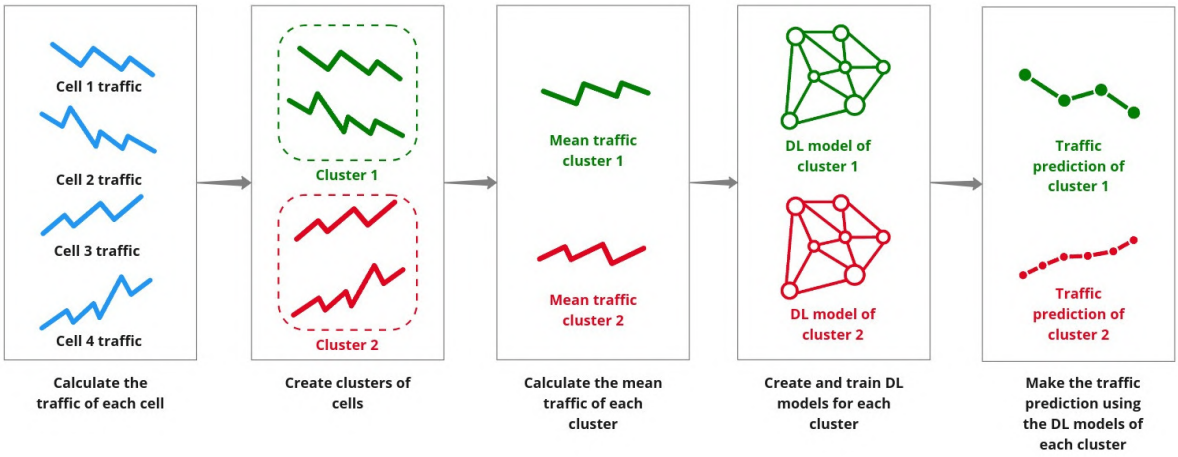
1 and 1000, respectively. The traffic of these cells is quite different. Cell 1 has some peaks at the beginning of the time series, with the biggest peak at middle of November 2013; in December the traffic is more stable. On the other hand, Cell 1000 traffic has regular peaks across November and December. At the end of December, Cell 1000 traffic drops, this does not happen in any other period of the time series, nor in Cell 1 traffic.

The traffic of these cells reinforces the complexity and dynamism of the scenario under study; different parts of Milan present different traffic across the periods. This poses a challenge when using just one model to predict the traffic for all cells, since DL models learn the pattern from the input data. Therefore, if a DL model trained with the Telecom Italia dataset learns the pattern of Cell 1 traffic, it may present a high error to predict the traffic of the Cell 1000, due to the different patterns. On the other hand, to create a prediction model for each cell present in the dataset will result in 10,000 models, which can be complex to manage, since these models must be trained and re-trained if the traffic pattern of the cells changes.

In order to deal with these issues, we create clusters of cells based on their traffic prediction patterns. Figure 18 illustrates the pipeline used to create the cells' clusters to make the traffic prediction. Initially, we calculate the traffic of each cell present in the Telecom Italia dataset. Afterwards, we create clusters of cells based on the traffic statistics, as we will explain in Section 5.3. We group all cells with similar traffic patterns into a cluster. Once the clusters of cells are created, we calculate the mean traffic of all cells for each cluster, composing N time series, where N is the number of clusters. These time series (mean traffic of each cluster) are used to train DL models for traffic prediction. Instead of training one model for each cell, which results in a high number of models to train (e.g. for the Milan region, approximately 10,000 DL models would need to be trained and managed), we create N DL models, one for each cluster. Since the traffic for cells in a given cluster have a similar pattern, the model of

the respective cluster can be used to predict traffic of the cells that compose that cluster. Cell clusters are discussed in more detail in the next subsection.

Figure 18 – Cell internet traffic prediction pipeline.



Source: the author (2023).

Since the DL models of each cluster are trained with the mean traffic of the respective cluster, we can use these models to predict the mean traffic for each cluster, the last step in our pipeline. To evaluate the performance of the DL models, we use RMSE and Mean Absolute Error (MAE) as metrics as detailed later in Subsection 5.5.

5.3 CLUSTERING THE CELLS

The dataset is composed of traffic data for different cells for Milan. As discussed previously, we propose created clusters using Internet activity as a statistical metric to propose DL models to predict Internet traffic. The aggregated Internet traffic can represent the traffic for the different SFCs allocated in the distributed scenarios (as shown in the Figure 9) We calculate the total number of Internet activities considering six periods in each day, as described in Table 11[1]

Each cell can be represented by a vector containing six values based on the average Internet activity for each period of the day. Based on these values, we create clusters of cells using the K-Means algorithm (ARORA; VARSHNEY et al., 2016) which has been widely used in a number of different research domains such as document classification, recommendation systems based on

[1] In Northern Italy, unlike other countries where lunch is 1230-1400, the working day often includes a break from 1200-1330 or 1430-1600. For the purposes of this study, we have aggregated this as a four hour block. Researchers may need to modify this for other countries.

Table 11 – Periods of the day.

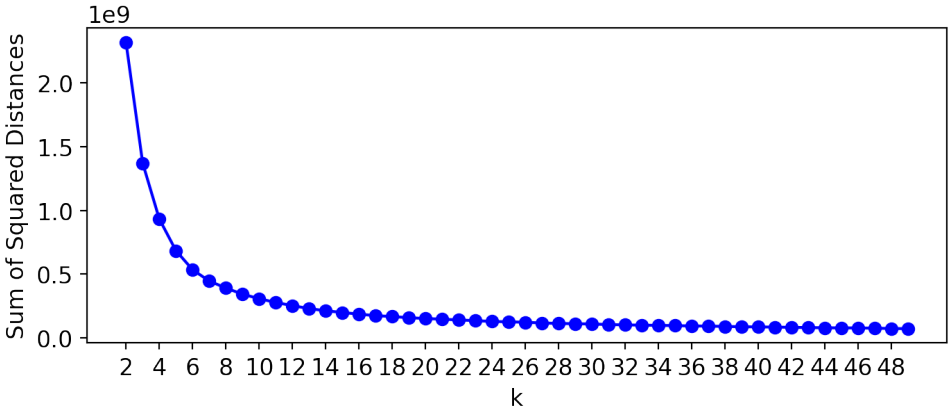
Period	Time (in hour)
Late Night	00:00 - 04:00
Early Morning	04:00 - 08:00
Morning	08:00 - 12:00
Afternoon	12:00 - 16:00
Evening	16:00 - 20:00
Night	20:00 - 00:00

Source: the author (2023).

user interests, classification based on user purchase behavior etc. The K-Means algorithm has many advantages compared to other clustering techniques including ease of implementation and fast convergence, even for Big Data (YUAN; YANG, 2019).

To automatically estimate the optimal number of clusters (k) of cells, we applied the Elbow method. This method varies the number of clusters within a range to find the optimal k based on the sum of square error (SYAKUR et al., 2018). We varied k from one to 50 as shown in Figure 19. As the number of cluster increases, the sum of squared distance tends towards zero with the elbow of curve being the optimal value. In our case, we select $k = 12$ since it is at the end of the elbow and the beginning of the stabilization of the sum of the squared distances. Based on the results presented in Figure 19, using more than 12 clusters would only increase the complexity of the algorithm with no significant gains in terms of performance.

Figure 19 – Elbow method results.



Source: the author (2023).

The 12 clusters have a similar Internet activity pattern across different time periods within each day, regardless of the cell location. Figure 20 shows the 12 clusters overlaid on a map of the Milan Metropolitan Area. To some extent, Cluster 1 represents the external areas of

Milan; Cluster 10 represents the border of the municipalities (excluding the municipality 1 in the center), while Cluster 6 is at the center of such municipalities. Other Clusters (2, 3, 4, 5, 7, 8, 9, 11 and 12) are regions within the city.

We calculate the mean Internet traffic for each of the 12 clusters using the 30-minute aggregated traffic of all cells included in each cluster divided by the number of cells of the cluster. Thus, for each cluster, we have a time series related to their mean cell's traffic. This time series is used to train and test the proposed DL models.

5.4 DL MODEL CONFIGURATION

In this thesis, we propose two different RNNs that are widely used in the DL literature for regression problems, LSTM and GRU (CAUX; BERNARDINI; VITERBO, 2020; ZHANG et al., 2017; CHNITI; BAKIR; ZAHER, 2017; CAO; LI; LI, 2019). To find the best configuration of the models, we apply a technique called Grid Search. This technique performs an exhaustive search in a subset of the previously defined parameters and provides the near optimal parameter combination within the given range (SYARIF; PRUGEL-BENNETT; WILLS, 2016). To apply the Grid Search, we vary the number of hidden layers and their units for both LSTM and GRU (see the parameters and levels in Table 12), which was adapted from the methodology used in (PONTES et al., 2016).

Table 12 – Grid Search parameters and levels.

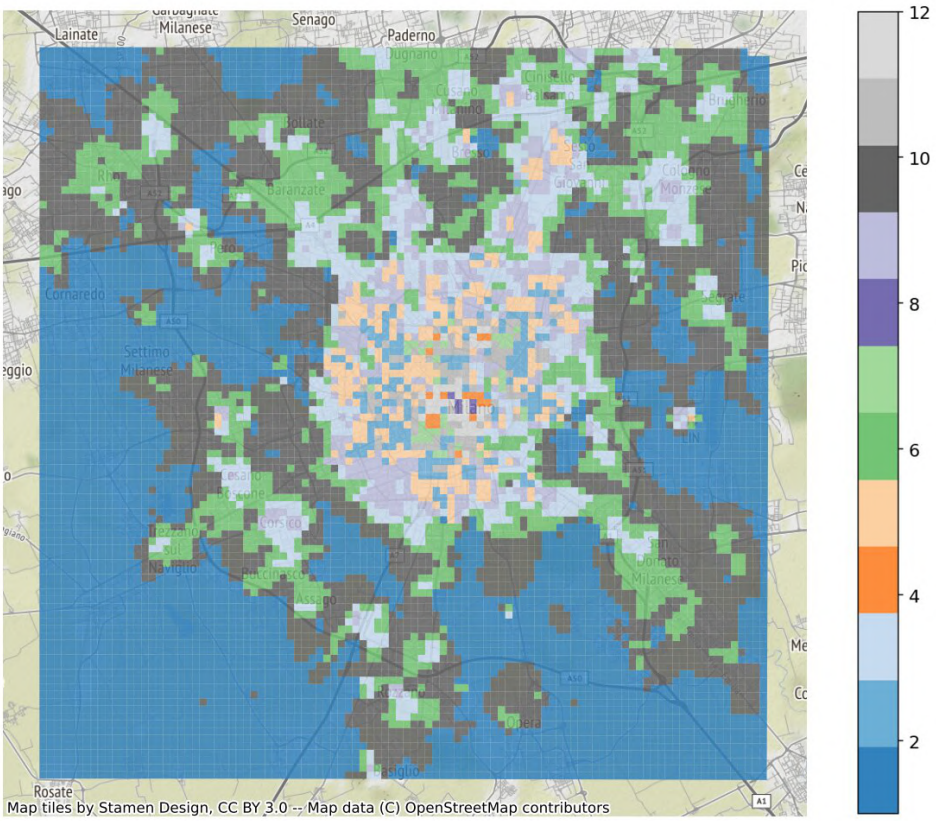
Parameters	**Levels**
Number of layers	1 to 4, step 1
Number of units	50 to 150, step 50

Source: the author (2023).

The first layer of the model is a fixed recurrent layer (the same as the hidden layers) where the number of units equals the input data length. The last layer is a fully connected layer with one neuron that gives the prediction value. Table 13 shows the fixed parameters (empirically chosen) to train the DL models.

Due to the random characteristics that exist in training (e.g. initialization of weights, selection of batches etc.), we perform the experiments 30 times and calculate the average RMSE and the average MAE.

Figure 20 – Overlay of the 12 clusters on a map of the metropolitan area of Milan.



Source: the author (2023).

Table 13 – Parameters used to train the DL models.

Parameter	**Value**
Activation function of recurrent layers	hard sigmoid
Activation function of last layer	sigmoid
Number of epochs	50
Optimizer	ADAM (KINGMA; BA, 2014)
Learning rate	0.001
Batch size	32
Loss function	mean squared error
Number of runs	30

Source: the author (2023).

5.5 METRICS FOR EVALUATING DL MODELS

To assess the performance of the models, we use two metrics: RMSE and MAE. The RMSE metric is calculated as per Equation 5.1:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2} \tag{5.1}$$

where N is the number of points from the traffic series, f_i is the model prediction at timestamp i, and y_i is the the real value at timestamp i (WANG; LU, 2018). We use RMSE because it measures the deviation between the true value and the value predicted by the model, and is widely used in extant literature for evaluating traffic prediction models (ZHANG et al., 2018; ZENG et al., 2020; KUBER; SESKAR; MANDAYAM, 2021; SHEN et al., 2021).

We also used MAE to evaluate the DL models. In contrast to RMSE, MAE assigns the same weight to all errors (CHAI; DRAXLER, 2014). MAE can be calculated as per Equation 5.2:

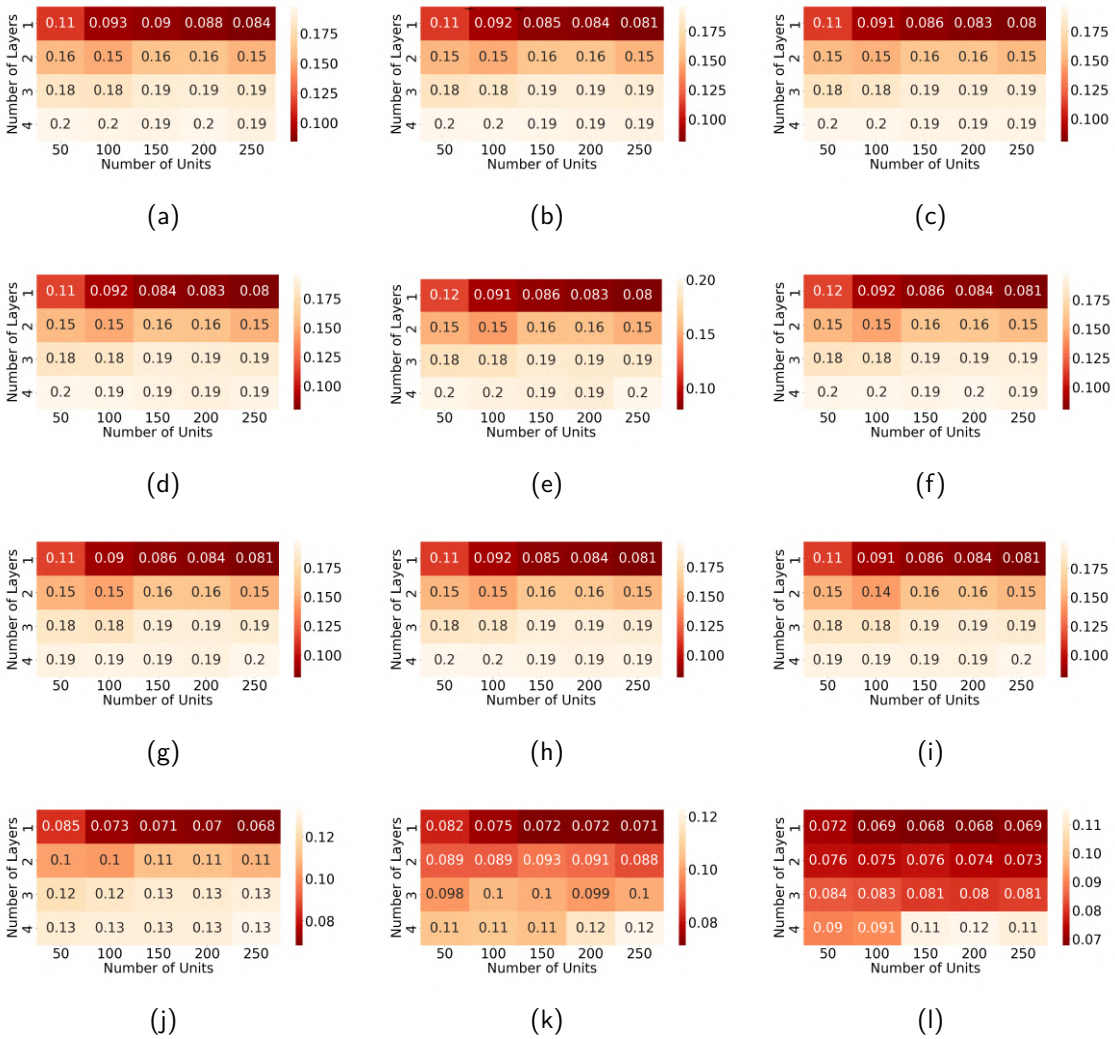
$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i| \tag{5.2}$$

where N is the length of time series, f_i is the prediction, and y_i is the actual value of timestamp i. Similar to the RMSE, MAE is widely used in the literature to evaluate traffic prediction models (ZANG et al., 2015; TROIA et al., 2018; CAO; LIU, 2019).

5.6 RESULTS

We use only the RMSE metric to assess the different DL architectures performance and find the best configuration for each one. In the Section 5.6.2 we use both RMSE and MAE to compare the best configurations of DL models between them and against ML models. Figures 21 and 22 present the Grid Search RMSE results for each cluster for each of the LSTM and GRU models, respectively. For the LSTM models (Figure 21), for the majority of the clusters, the best configuration has one layer and 250 units. The only exception is Cluster 12, where the best overall average RMSE is achieved using a configuration with one layer and 150 units (LSTM-12-1L-150U) and one layer with 200 units (LSTM-12-1L-200U). Cluster 1 achieves the worth average RMSE result (0.084) while Cluster 12 achieves the best average RMSE result (0.068).

Figure 21 – Mean RMSE of LSTM model for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12.



Source: the author (2023).

For GRU (Figure 22), the configurations with one layer present the best average RMSE, however the performance of different clusters varies based on the number of units. From Cluster 2 to Cluster 9 (GRU-2-1L-200U, GRU-3-1L-200U, GRU-4-1L-200U, GRU-5-1L-200U, GRU-6-1L-200U, GRU-7-1L-200U, GRU-8-1L-200U, and GRU-9-1L-200U), the configuration with the lowest average RMSE is one layer with 200 units. For Clusters 1 and 10, the configuration that provides the best average RMSE is one layer with 250 units (GRU-1-1L-250U and GRU-10-1L-250U). In Clusters 11 and 12, two configurations have the best average RMSE, all with only one layer. For Cluster 11, the best configurations have 150 units (GRU-11-1L-150U) and 250 (GRU-11-1L-250U) units while for Cluster 12 the best performing configuration has 150 units (GRU-12-1L-150U) and 200 units (GRU-12-1L-250U). Similar to LSTM, the clusters with the

worst and best average RMSE are Cluster 1 (0.0091) and Cluster 12 (0.0045), respectively.

The complexity of the model is directly related to the number of layers and units i.e. the more layers and units, the more complex the model becomes. The number of layers and units of the model has to be adjusted according to the input data. Figures 21 and 22 illustrate that very complex models (those with many layers and units) resulted in poorer performance, due to model overfitting. Simpler models with less complexity also performed poorly, most likely due to model underfitting.

In general, the DL models with one hidden layer obtain better average RMSE results than models with more layers; while those with 150 units or more result in better average RMSE results. Fine-tuning the models by increasing the number of units rather than layers result in better performance. Adding hidden layers results in performance degradation.

For the LSTM models (Figure 21), for the majority of the clusters, the best configuration had one layer and 250 units. The only exception is Cluster 12, where the best overall average RMSE was achieved using a configuration with one layer and 150 units (LSTM-12-1L-150U) and one layer with 200 units (LSTM-12-1L-200U). Cluster 1 achieved the worth average RMSE result (0.084) while Cluster 12 achieved the best average RMSE result (0.068).

5.6.1 Statistical analysis

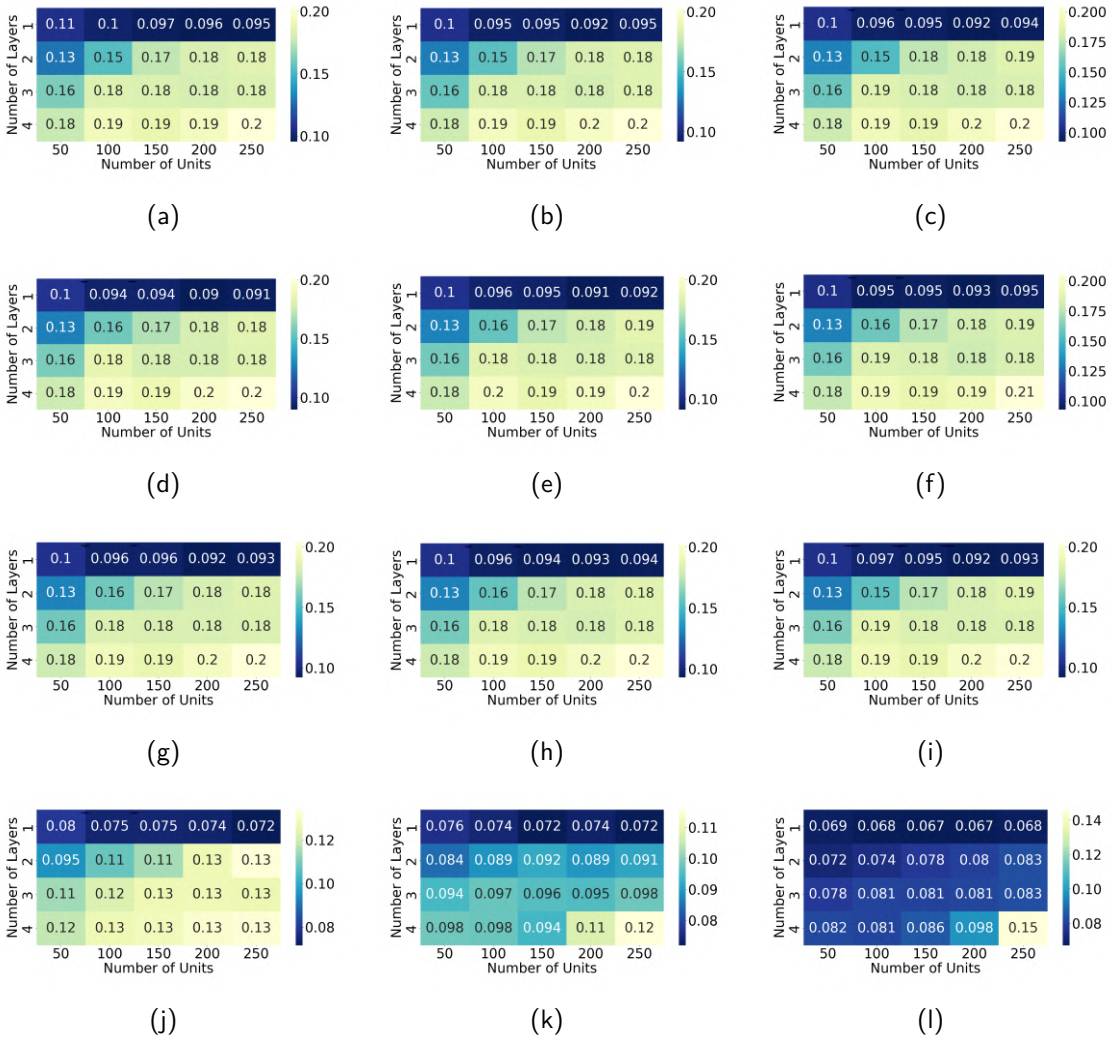
Some configurations achieved by the Grid Search obtained very similar average RMSE. To explore this further, we use Kruskal-Wallis non-parametric analysis to compare independent samples to check whether they are similar or not, based on the mean ranks of these samples (ELLIOTT; HYNAN, 2011).

In our LSTM results, Cluster 12 (LSTM-12-1L-150U and LSTM-12-1L-200U) has two configurations with the same average RMSE. Figure 23 presents the box plot of the RMSE of these Cluster 12 configurations.

While the best Cluster 12 configurations have the same average RMSE (0.068), they have different RMSE distributions. We can note that the LSTM-12-1L-200U has a lower dispersion and a lower median than LSTM-12-1L-150U. LSTM-12-1L-200U has an outlier below the minimum RMSE value, and LSTM-12-1L-150U has an outlier above of the maximum RMSE value. This analysis suggests that LSTM-12-1L-200U is the best configuration since it has the lowest dispersion and the lowest median.

For the GRU models, Clusters 1-3 and 5-9 each had at least one statistically similar best

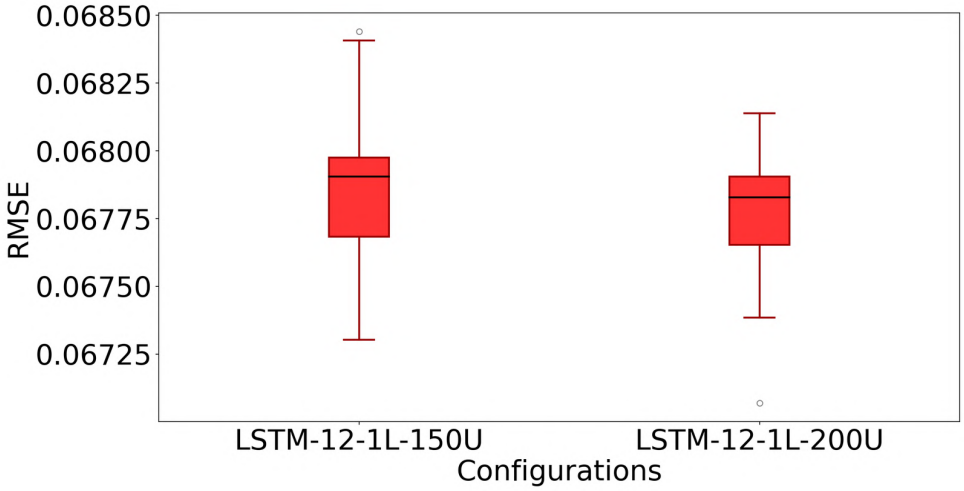
Figure 22 – Average RMSE of GRU model for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12.



Source: the author (2023).

configuration (Figure 24). For Clusters 1-3 and 5-7, the configurations with 250 units (GRU-1-1L-250, GRU-2-1L-250, GRU-3-1L-250, GRU-5-1L-250, GRU-6-1L-250 and GRU-7-1L-250) presented a higher dispersion than the configurations with 200 units (GRU-1-1L-200, GRU-2-1L-200, GRU-3-1L-200, GRU-5-1L-200, GRU-6-1L-200 and GRU-7-1L-200); GRU-1-1L-200 presented a higher median than GRU-1-1L-250. For Cluster 8, three models are statistically similar, those with one layer and 150, 200, and 250 units. Again, the configuration with 200 units presented lower dispersion and a lower median than the other configurations. Finally, both configurations of Cluster 9 (GRU-9-1L-200 and GRU-1-1L-250) had very similar distributions, with similar dispersion and a similar median. In general, for these clusters with statistically similar configurations, the configuration with one layer and 200 units presented a lower dis-

Figure 23 – Box plot of the RMSE of the best LSTM configurations for Cluster 12.



Source: the author (2023).

persion and lower median; this is considered the best performing configuration for the GRU models.

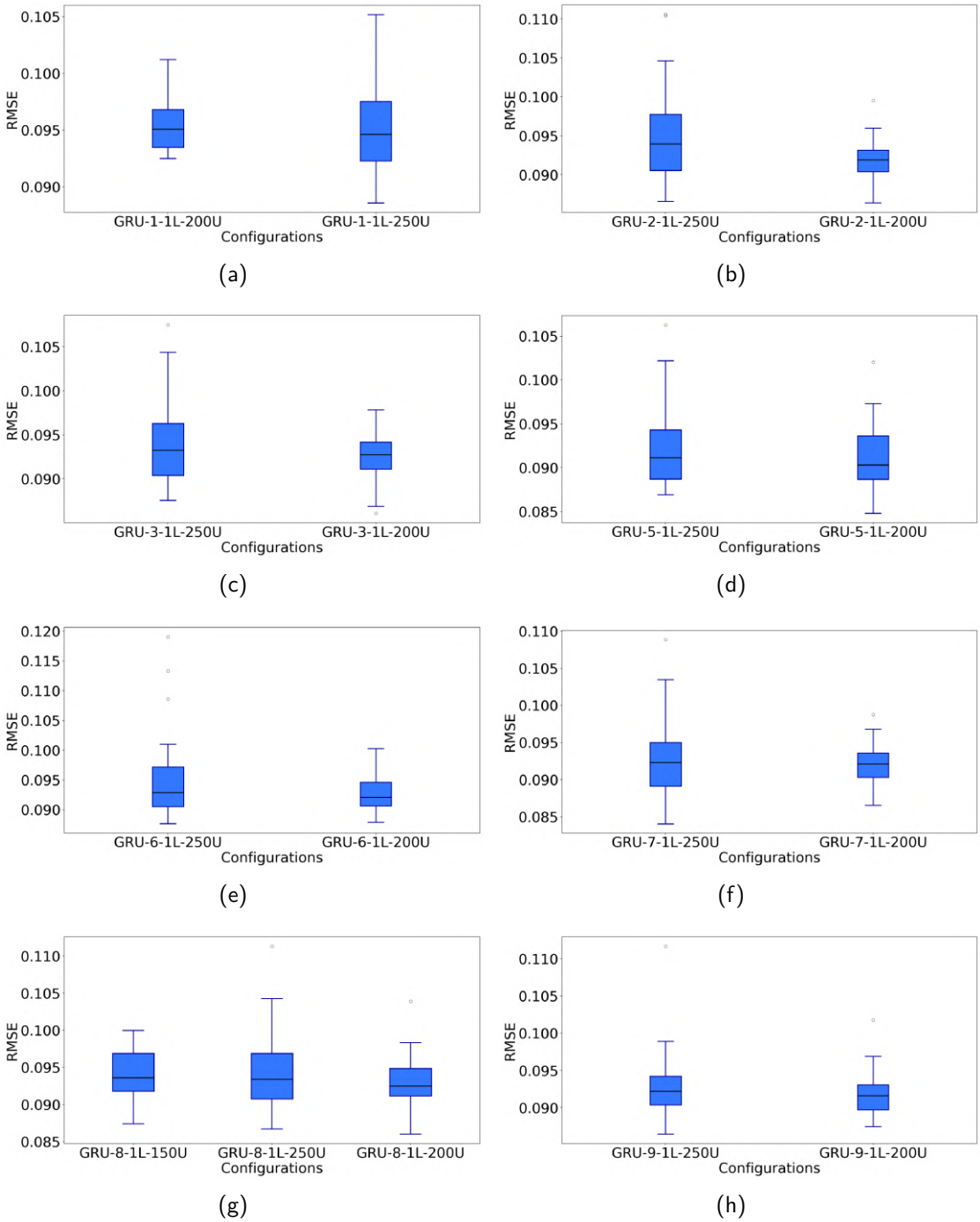
5.6.2 Comparison of LSTM and GRU models.

Table 14 presents the results for the RMSE and MAE metrics, comparing the LSTM and GRU models against two traditional ML models as baseline: Random Forest and Decision Tree. We chose these ML models as baseline for the comparison, but they were widely used for time series prediction (KANE et al., 2014; TYRALIS; PAPACHARALAMPOUS, 2017; KARASU; ALTAN, 2019). For each cluster, we created the DL and ML models, then trained and evaluated with the same training and testing datasets 30 times and calculated the average RMSE and MAE.

The DL models outperform the ML models for all clusters based on the RMSE and the MAE results. For the Cluster 1, the LSTM has a reduction of 50.32% in average RMSE and a reduction of 51.99% in average MAE when compared with Random Forest. Considering the Decision Tree, the LSTM presents a reduction of 62.67% in the average RMSE and a reduction of 62.96% in the average MAE. In the Cluster 1, comparing the GRU and Random Forest, the GRU has a reduction of 43.78% in the average RMSE and 45.66% in the average MAE. In the same cluster, the GRU presents a reduction of 57.64% and 58.08%, for the average RMSE and MAE respectively, when compared with Decision Tree.

The DL models are clearly superior to the conventional ML models for traffic prediction tasks; the ML models cannot achieve the same level of prediction error of DL models based

Figure 24 – Boxplot of the RMSE of the best configurations of (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 5, (e) cluster 6, (f) cluster 7, (g) cluster 8, and (h) cluster 9.



Source: the author (2023).

on the RMSE and the MAE.

ML model results suggest that the Random Forest model outperforms the Decision Tree model for all clusters. For Cluster 1, the Random Forest model presents an average RMSE of 0.1695, while the Decision Tree model presented an average RMSE of 0.2251, a difference of 24.70%. For Cluster 1, the Random Forest and Decision Tree present an average MAE of

Table 14 – Comparison of the LSTM, GRU, Random Forest, and Decision Tree models.

Cluster	LSTM		GRU		Random Forest		Decision Tree	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	0.0842	0.0653	0.0953	0.0739	0.1695	0.1360	0.2251	0.1763
2	0.0809	0.0639	0.0917	0.0716	0.1664	0.1330	0.2200	0.1677
3	0.0803	0.0634	0.0922	0.0720	0.1645	0.1316	0.2202	0.1670
4	0.0796	0.0629	0.0896	0.0701	0.1699	0.1348	0.2052	0.1630
5	0.0800	0.0632	0.0909	0.0709	0.1602	0.1276	0.2007	0.1567
6	0.0806	0.0636	0.0928	0.0724	0.1665	0.1331	0.2198	0.1671
7	0.0807	0.0638	0.0920	0.0718	0.1675	0.1339	0.2185	0.1654
8	0.0808	0.0639	0.0929	0.0725	0.1684	0.1347	0.2208	0.1680
9	0.0805	0.0636	0.0917	0.0716	0.1680	0.1345	0.2219	0.1699
10	0.0683	0.0544	0.0715	0.0564	0.0957	0.0740	0.1084	0.0824
11	0.0712	0.0564	0.0719	0.0565	0.0842	0.0627	0.1072	0.0788
12	0.0677	0.0546	0.0671	0.0539	0.0784	0.0580	0.1035	0.0772

Source: the author (2023).

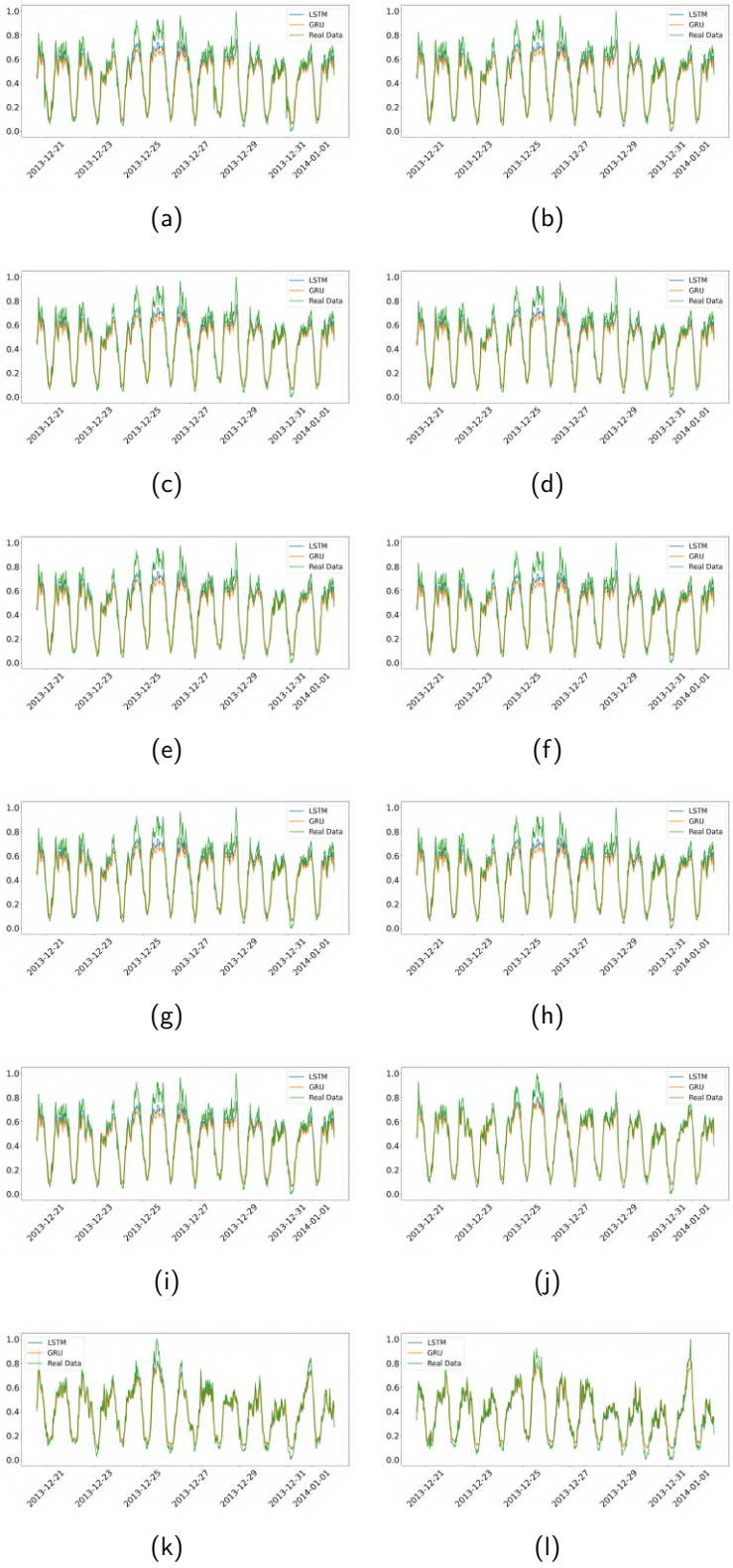
0.1360 and 0.1763, respectively, a difference of 29.63%.

We compared the best configurations for the LSTM and GRU models for each cluster using the Kruskal-Wallis test. The best configurations of LSTM and GRU based on the number of layers and RNN units were selected using the lowest average RMSE values (see Figures 21 and 22). We consider 30 RMSE values obtained from experiments. For all clusters, the RMSE distributions of the LSTM and GRU models are statistically different. From Table 14, one can note that the LSTM models obtain better average RMSE and MAE results except for Cluster 12, where the GRU slightly outperforms the LSTM.

Cluster 1 presents the worst results for RMSE and MAE. LSTM presents an average RMSE of 0.0842, while the GRU presents 0.0953, a difference of 13.18%. In the MAE results for Cluster 1, the LSTM presents 0.0653, while the GRU presents 0.0739, a difference of 13.17%. In contrast, Cluster 12 has the best average RMSE, 0.0677 and 0.0671 for LSTM and GRU, respectively, with a difference of 0.89%. For the same cluster, the LSTM and GRU present an average MAE of 0.0546 and 0.0539, respectively, with a difference of 1.28%.

To help understand the difference between the performance of the models across the clusters, we visualise the actual Internet activity (green line), the LSTM model internet traffic predictions (blue line), and the GRU model internet predictions (orange line) (see Figure 25). For visualisation and sensemaking purposes, we omit the ML model predictions. Both models

Figure 25 – Comparison against ground truth Internet activity and the predictions of LSTM and GRU models for (a) cluster 1, (b) cluster 2, (c) cluster 3, (d) cluster 4, (e) cluster 5, (f) cluster 6, (g) cluster 7, (h) cluster 8, (i) cluster 9, (j) cluster 10, (k) cluster 11, and (l) cluster 12.



Source: the author (2023).

learned the pattern of Internet activity data, capturing its seasonality. For Clusters 1-9, the models' predictions are slightly lower than the ground truth data in some periods, with the GRU prediction values lower than the LSTM predictions, consistent with the GRU models' higher average RMSE and MAE results. For Clusters 10-12, the predictions of both models are closer to the ground truth data in comparison to the other clusters, with the LSTM model closer to the ground truth data in more periods than the GRU models. It is important to notice that Cluster 10 is situated at the outskirt of the metropolitan area of Milan while Clusters 11 and 12 are closer to the center of the city.

The periods where the predictions of the DL models (for all clusters) are more distant from the ground truth data are in the Christmas period (24-26 December). During this period, the predictions of both models are much lower than the ground truth data. This can be explained easily by the seasonal, although predictable, traffic at that time. This could be addressed by augmenting the overall DL scheme with historical statistics that summarize prior knowledge of predictable long-term trends as per (ZHANG; PATRAS, 2018).

Based on our statistical tests, the LSTM models outperform the GRU models in all but one cluster. In that case, Cluster 12, the superior performance of the GRU model is relatively small. As shown in Figure 25, both RNNs capture the data pattern, however, the values predicted by the LSTM models are closer to the ground truth Internet activity data. This is not entirely surprising. As discussed in Section 2.4.1.2, typically LSTM is expected to outperform its less complex variant, GRU. We attribute the lower performance of the GRU to the lower complexity of the GRU units in comparison to the LSTM. Figure 25 shows that the predictions of the GRU model follow a similar pattern as the LSTM, but with lower values. These lower values increase prediction error, resulting in a higher RMSE and MAE. Notwithstanding this, the GRU model took less time to train, and thus, where a model needs to be retrained for multi-step traffic prediction, the benefits of efficiency gains (e.g. faster decision making, and computational and economic cost savings) may outweigh the differences in accuracy.

The work presented by Chen et al. (CHEN et al., 2018a) also used the Telecom Italia dataset for Milan and an LSTM to predict base station traffic. While they used a clustering strategy at a group base stations level, it was *post hoc* i.e. they forecast the traffic patterns using the LSTM and *then* cluster complementary base stations to BBUs based on the traffic patterns. As such, the results presented by Chen et al. (CHEN et al., 2018a) are not entirely comparable with this study since they considered a different clustering strategy and a different time interval in their experiments. In fact, the different clustering approaches may result in different traffic

patterns resulting in a direct impact on the model performance. Finding related works for direct comparison in the context of traffic prediction is a complex task as different studies may vary dataset, evaluation metrics, transformations, and data preprocessing. Notwithstanding the differences, Chen et al. (CHEN et al., 2018a) is the closest to study identified to that presented in this article, and consequently we use it as a basis for comparison. All average MAE values obtained for LSTM models were lower than the best result presented in (CHEN et al., 2018a), which was 0.074. For the GRU models, Cluster 1 obtained the same MAE as Chen et al. (CHEN et al., 2018a) (0.0739) while the models for the other clusters performed better. Cluster 12 achieved the best MAE. The LSTM presented an improvement of 26.22% and the GRU model presented an improvement of 27.16% when compared to the reported results in Chen et al. (CHEN et al., 2018a).

5.7 CONCLUDING REMARKS

In this chapter, we proposed and compared the performance of two RNNs, LSTM and GRU, to predict mobile Internet traffic in a large metropolitan area, Milan. We proposed a novel *a priori* clustering to group cells using K-Means clustering and used the Grid Search method to identify the best configurations for each RNN. We compared RNN performance using RMSE and MAE, and testing against ground truth data for Milan. Both RNNs were effective in modelling Internet activity and seasonality, both within days and across two months however were sub-optimal in predicting anomalies e.g. Christmas. In this case, this could have been addressed by augmenting the training with historic trend data as per (ZHANG; PATRAS, 2018). We also find variations by in clusters across the city. While the LSTM outperformed the GRU, the GRU had faster training times which may be relevant for multi-step prediction scenarios. We compared our results with Random Forest and Decision Tree, common ML model techniques used for time series prediction. Both LSTM and GRU models outperformed the ML models for all clusters of cells. We also compared our proposed RNN models against the results in Chen (CHEN et al., 2018a) using MAE. Notwithstanding the validity issues in such a comparison, results suggest our models present significantly better performance. In the next Chapter, we present the RL-based algorithm proposed to define the SFC placement, which is used to implement the Placement Planner module of SPIDER, as shown in Figure 14.

6 REINFORCEMENT LEARNING FOR SFC PLACEMENT

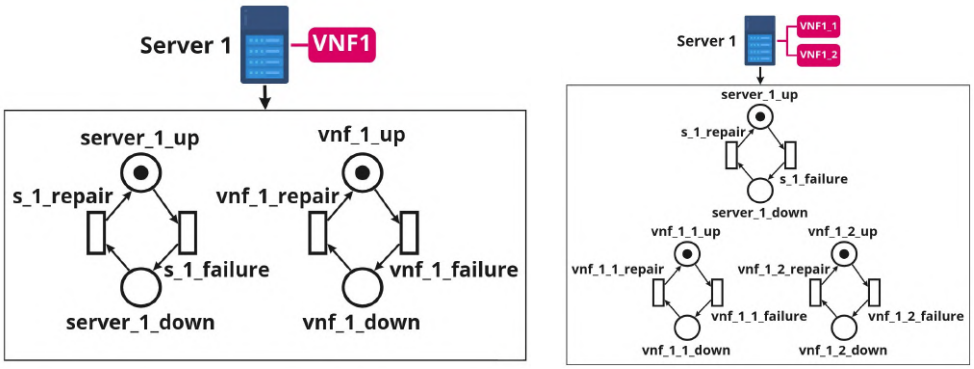
In this chapter, we describe how we use RL for the SFC placement task, which is used for the Placement Planner module of SPIDER agent, as shown in Figure 14. Firstly, we describe an algorithm to calculate the SFC availability by using SPN models in Section 6.1. The algorithm generates SPN models based on the SFC placement, i.e., in which each VNF is placed and the redundancy strategy. In Section 6.2, we present the model system considered for the SFC placement. We also formulate the SFC placement as a multi objective problem. Then, we propose the Cand algorithm, which is used to select candidate nodes to place the VNFs. The main advantage of use the Cand algorithm is to reduce the search space to define the node where each VNF is placed. After we formulate the SFC placement problem to be solved using RL, and the Cand algorithm is combined with a RL agent to create the Cand-RL algorithm, which is used to define the SFC placement. Finally, in Section 6.6, we present the evaluation of Cand-RL against greedy algorithms. The results obtained from this chapter were published in (SANTOS et al., 2021), (SANTOS et al., 2021), and (SANTOS et al., 2022).

6.1 MODELLING SFC PLACEMENT

Firstly, we generate the computational models to estimate the availability of the SFC placement. The model needs to represent the state of the hardware and software components as well as the interdependency between the VNFs and the physical nodes. We use SPN to model the SFC placement since it is a powerful technique to represent complex systems capable of accommodating a variety of aspects including concurrency, synchronization, communication mechanisms, deterministic, mutual exclusion, and conflict (ANDRADE et al., 2017). In addition, SPN, different of other modelling approaches, has a great representation power with a simple visual notation. Using SPN models, it is possible to model the operational states of physical nodes and VNFs, and calculate the overall SFC placement availability. In addition, an SPN model can be converted to a CTMC and solved using numerical or analytic methods; simulation methods could be used to derive the same results. In this work, we solved our SPN models through analytical methods in order to obtain the exact solution. For more information about SPN, please see (BAUSE; KRITZINGER, 2002).

Figure 26 illustrates two SFC placements in physical servers. These illustrations are com-

Figure 26 – Placement examples of VNFs of the same type



(a) Most simple placement example

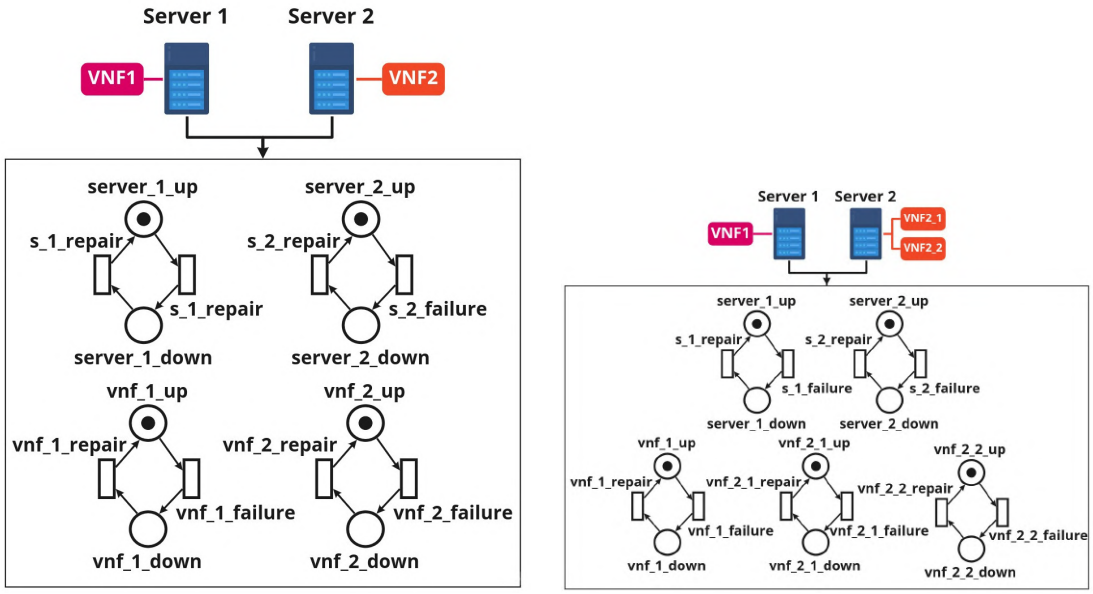
(b) Placement example with redundant VNFs

Source: the author (2023).

posed of just one VNF for simplicity, but in practice the SFC can be composed of many VNFs. Figure 27(a) assumes only one VNF is placed in a server. Server 1 and VNF1 are modeled as a building block composed of two places and two stochastic transitions. When there is a token in the place *server_1_up*, Server 1 is operational and working properly. In this state, Server 1 may fail; this event is represented by the stochastic transition *s_1_failure*. When this transition fires, one token is consumed from the place *server_1_up* and one token is produced in the place *server_1_down*. When there is a token in the place *server_1_down*, Server 1 is down, compromising all VNFs placed in this server. When Server 1 is down, it can be repaired thus making it available again. The transition *s_1_repair* represents the Server 1 repair event. When fired, one token is consumed from the place *server_1_down* and another is produced in the place *server_1_up*. VNF1 is modeled with a similar building block. The placement illustrated in Figure 27(a) is available if Server 1 and VNF1 are operational. Therefore, the availability can be calculated as the probability of one token being in the place *server_1_up* and in the place *vnf_1_up*.

Figure 27(b) shows an SFC placement example where the main VNF (VNF1_1) has a replica (VNF1_2). VNF1_1 and VNF_2 provide the same function, and both can keep the SFC available i.e. should any VNF fail, the other one can keep the function available. This redundancy approach can be used to increase SFC availability. The availability of overall SFC depends on whether Server 1 and VNF1_1 or VNF1_2 are operational. Using the SPN model illustrated in Figure 27(b), the overall SFC availability can be calculated as the probability of one token being in the place *server_1_up* and either one token being in the place *vnf_1_1_up* or one token being in the place *vnf_1_2_up*.

Figure 27 – Placement examples of SFCs with two VNF types



(a) SFC placement with two VNF types without redundancy

(b) SFC placement with two VNF types with redundancy in one VNF type

Source: the author (2023).

Figure 27 illustrates the SPN models for SFC placements with two VNF types. Figure 28(a) illustrates a placement without redundancy where each VNF is placed in different servers. Since the overall SFC depends on VNF1 and VNF2, these functions must be operational as well as the servers in which they are placed. Therefore, the availability of SFC represented in Figure 28(a) is calculated as the probability of one token being in the places *server_1_up*, *vnf_1_up*, *server_2_up*, and *vnf_2_up*.

In Figure 28(b), there is a replica of VNF2 (VNF2_2) to increase the overall SFC availability. Similar to Figure 27(b), if any replica of VNF2 is working, this function is operational for the SFC. Thus, the availability of the SFC is calculated as the probability of a token being in places *server_1_up*, *vnf_1_up*, *server_2_up*, and either *vnf_2_up* or *vnf_2_up*.

6.1.1 Generating SFC Availability models Automatically

As can bee seen in the SPN models presented in Figures 26 and 27, as more components are considered (additional servers and VNFs) the models become larger. This increase in the SPN model results in a larger CTMC. This can be seen in the SPN models presented in Figures 27(a) and 27(b). When just one building block is added in the SPN, it results in CTMC models with four and eight states, respectively. Similarly, in the SPN model presented in Figure 28(a),

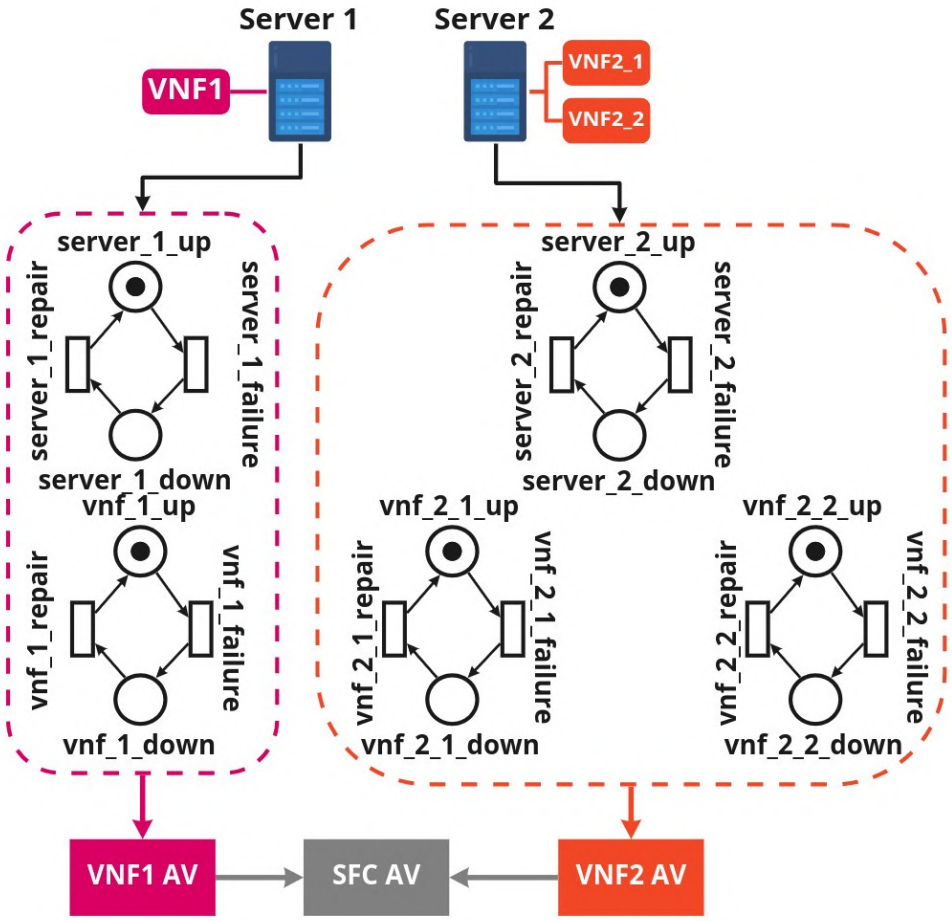
the respective CTMC has 16 states, while the CTMC of the SPN presented in Figure 28(b) has 32 states. As the SFC placement components increases, the size of the respective CTMC also increases. The generation process, and the CTMC resolution to obtain the probabilities of its states, can be computationally costly. To avoid large SPN models, we proposed an algorithm to generate relatively smaller SPN models and combine the results of these models in order to calculate the overall SFC availability.

Figure 28 presents the basic functioning of our algorithm. We generate an SPN model for the placement of each VNF type. The models are composed of building blocks relating to the VNF instances of a VNF type and the servers in which those instances are placed. Then, the respective CTMC models are generated and solved in order to calculate the availability for each VNF type. For instance, considering the SFC placement example showed in Figure 28, two SPN models are generated, one each for VNF1 and VNF2. Then, the availability of VNF1 and VNF2 are calculated. Since we assume that the SFC availability depends on the availability of VNF1 and VNF2, the overall SFC availability is calculated as the multiplication of availability values in regard to VNF1 and VNF2.

Figure 28 illustrates a placement where VNF instances of different types are placed in different servers. However, it is possible to place VNFs of different types in shared servers. Figure 29 shows an example where VNFs of different types are placed in the same server. There is one instance of VNF1 and one instance of VNF2 in Server 1. Since these two VNFs instances, of different types, are placed in the same server, we generate an SPN model to calculate the availability of these two VNF types. On the other hand, as VNF3 is placed in another server (Server 2), an SPN model is generated for this VNF type. As a result, these two SPN models are solved and two availability values are calculated: the availability of VNF1 and VNF2, and the availability of VNF3. These availability values are multiplied in order to calculate the overall SFC availability.

Algorithm 1 illustrates how the SPN models are generated and solved to calculate the overall SFC availability. The input, *sfc_placement*, is defined using the following JSON format: *{vnf_type:server_id:[vnf_instance_id]}*. Thus, for each *vnf_type*, we define on which servers (*server_id*) the instances (*vnf_instance_id*) are allocated, where it is possible to have several instances of the same *vnf_type* on the same server. For example, the input data for the SFC presented in Figure 28 is {1:{1:[1]}, 2:{2:[1,2]}}, while the input data for the SFC placement presented in Figure 29 is {1:{1:[1]}, 2:{1:[1]}, 3:{2:[1]}}. The output of the algorithm is the availability of SFC, which is a value between 0 and 1, depending on the server and the VNF

Figure 28 – Generation of SPN models based on the SFC placement



Source: the author (2023).

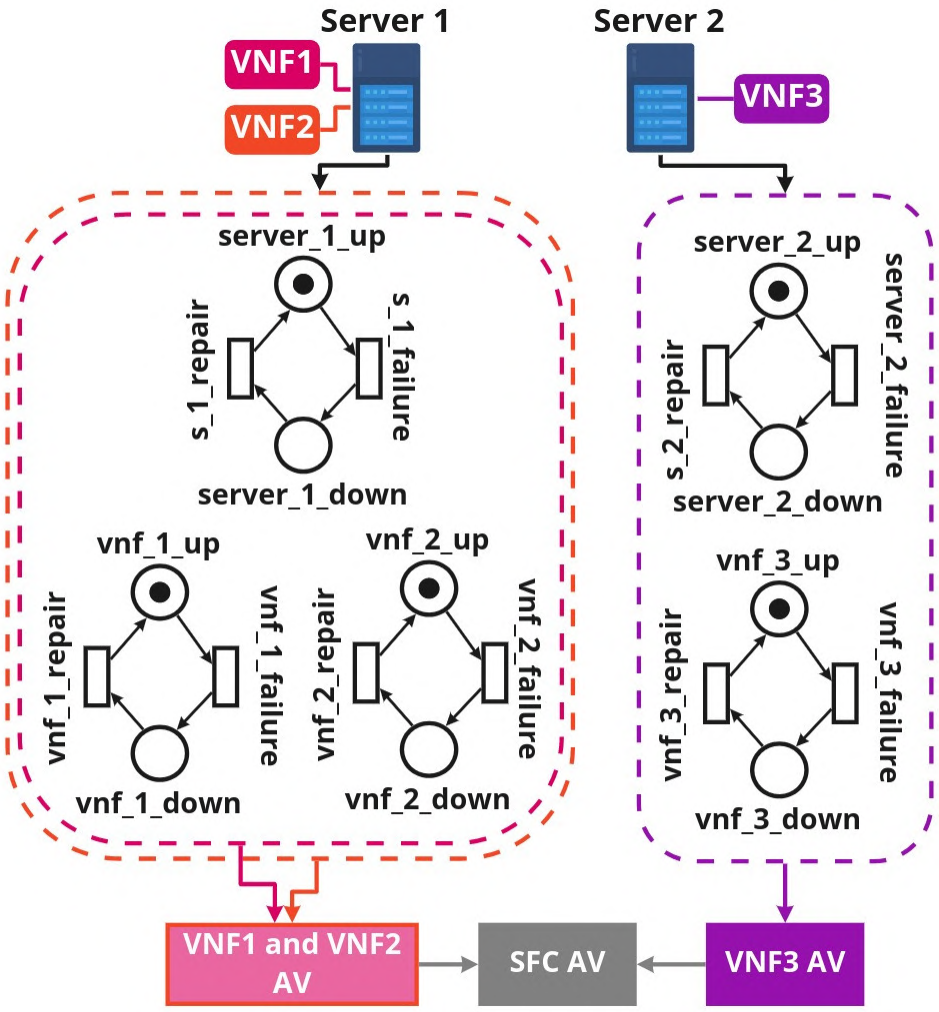
instances.

The first step in our algorithm is to find where VNF instances of different types are placed on shared servers. In Line 2, we create a variable that stores the mapping between the server IDs and the VNF types that are stored in this map. In the for loop of Line 3, we iterate over all VNF types present in the input data, while the for loop of Line 4 iterates over the servers. The mapping between the servers and VNF instances that are placed in these servers is carried out in Line 5.

After discovering the VNF instances of different types that are placed on shared servers, we create a list with just the IDs of these VNF types in Line 8 by the function *get_vnfs_shared_servers*. The variable *vnfs_list* holds the IDs of VNFs, and in cases of VNFs located in shared servers, their IDs are grouped in a list. In Line 9, we define the SFC availability variable and initialise it to 1.

The for loop in Line 10 iterates over the VNF types that are placed on shared servers. It

Figure 29 – Generation of SPN models with different VNFs types placed in shared servers



Source: the author (2023).

is important highlight that if the SFC placement does not present VNF types placed in shared servers (such as those in Figures 26, 27, and 28), our algorithm will generate an SPN for each VNF type. In Line 11, the function *get_sub_placement* obtains the placement information about a specific VNF type. For example, assuming the input data {1:{1:[1]}, 2:{2:[1,2]}}, the placement information about the VNF type 1 is {1:{1:[1]}}. However, information about more than one VNF can be generated depending on the number of VNF types that are placed on shared servers (as defined in variable *vnf_id* in Line 10).

The placement information about these VNFs that are placed on shared servers acts as the input of function *generate_spn*, that in turn generates the SPN model (line 12). The function *do_spn_analysis* converts the SPN model into a CTMC and solves it analytically in Line 13. In Line 14, the function *calc_placement_availability* calculates the availability based on the solution from the SPN and multiplies it by the *sfc_availability* variable. The availability is

Algorithm 1: Pseudocode for algorithm to generate SPN models of SFC placements

```
   Input  : sfc_placement
   Output: sfc_availability
 1 begin
 2     servers_vnfs ← {∅}
 3     for vnf_type ∈ sfc_placement do
 4         for server_id ∈ sfc_placement[vnf_type] do
 5             servers_vnfs[server_id].append(vnf_type)
 6         end
 7     end
 8     vnfs_list ← get_vnfs_shared_servers (servers_vnfs)
 9     sfc_availability ← 1
10     for vnf_id ∈ vnfs_list do
11         vnf_placement ← get_sub_placement (vnf_id,sfc_placement)
12         spn ← generate_spn (vnf_placement)
13         do_spn_analysis (spn)
14         sfc_availability ← sfc_availability * calc_placement_availability
            (vnf_placement)
15     end
16     return sfc_availability
17 end
```

calculated taking into account the probability of have tokens in specific places of SPN model, as explained previously. Therefore, the availability of all VNFs types are multiplied to calculate the overall availability of the SFC, which is returned in Line 16.

We carried out experiments about different SFC placements strategies and how they impact the overall SFC availability. We also compared the simplified SPN models generated by Algorithm 1 against traditional SPN models. These results are presented in Appendix A.

As we use RL agents to define the SFC placement, the SFC availability must be calculated during the agent training. We use the Algorithm 1 to generate SPN models and estimate the availability of the SFC placement configuration defined by the RL agent. In the next section, we present the system model considered and in the next sections we present the algorithm proposed for the SFC placement.

6.2 SYSTEM MODEL FOR SFC PLACEMENT PROBLEM

We assume an infrastructure composed of a set of physical nodes and physical links that connect them. Table 15 summarizes all the parameters used in the model.

The physical nodes can be different kind of devices in which VNFs can be placed, such

Table 15 – Summary of parameters used in the model.

Notation	Definition
G_{net}	Graph of network infrastructure.
N_{net}	Set of all physical nodes in G_{net}.
$Rcpu_n$	CPU Resources from node n.
$Rmem_n$	Memory Resources from node n.
$Rsto_n$	Storage Resources from node n.
A_n	Availability of node n.
L_{net}	Set of all links in G_{net}.
B_l	Bandwidth of link l that connects two different nodes.
D_l	Delay of link l that connects two different nodes.
G_{sfc}	graph of an SFC request.
N_{src}	Source node of an SFC request.
S_{dst}	Destination node of an SFC request.
$Vcpu_v$	CPU requirements for a VNF v.
$Vmem_v$	Memory requirements for a VNF v.
$Vsto_v$	Storage requirements for a VNF v.
Br_f	Bandiwdth requirement of virtual link f.

Source: the author (2023).

as servers, switches, routers, etc. Physical nodes have limited computational capabilities in terms of CPU, memory, and storage. These resources are consumed by the VNFs placed in the nodes. The resources are represented as a vector: $R_n = \{Rcpu_n, Rmem_n, Rsto_n, n \in \{1, 2, ..., N_{net}\}\}$, where, for the node n of a set of nodes N_{net}, $Rcpu_n, Rmem_n$, and $Rsto_n$, are the CPU, memory, and storage, respectively.

Each node in the physical infrastructure has associated MTTF and MTTR values. The MTTF refers to the meantime a node has failed since it started operating. In computational systems, failures can happen due to several reasons, such as hardware or software failures, planning mistakes, human error, or external attacks (ENDO et al., 2017). On the other hand, the MTTR is the mean time the node takes to be repaired after a failure. The repair time is composed of the failure discover time and the repair time itself, which can be the equipment replacement, the hardware repair, the software reboot, and so on. Although the failure and repair rates can vary during the infrastructure operation, a common assumption in the works that model computing systems is to assume that these rates are constant (ALMURSHED; RANA; CHARD, 2022) (PEREIRA et al., 2022), i.e., the MTTF and MTTR are fixed. These values are essential to estimate the availability, A_n, of a node n.

The physical nodes also have constraints relating to the VNF types that they can receive. For instance, in a cellular network scenario, some nodes close to the base stations, with low computational capability, may receive VNFs about signal processing, while nodes in the cloud may receive VNFs that demand computational resources. Naturally, some nodes may receive all VNFs types defined by the network manager while other nodes may not be capable of receiving any VNF.

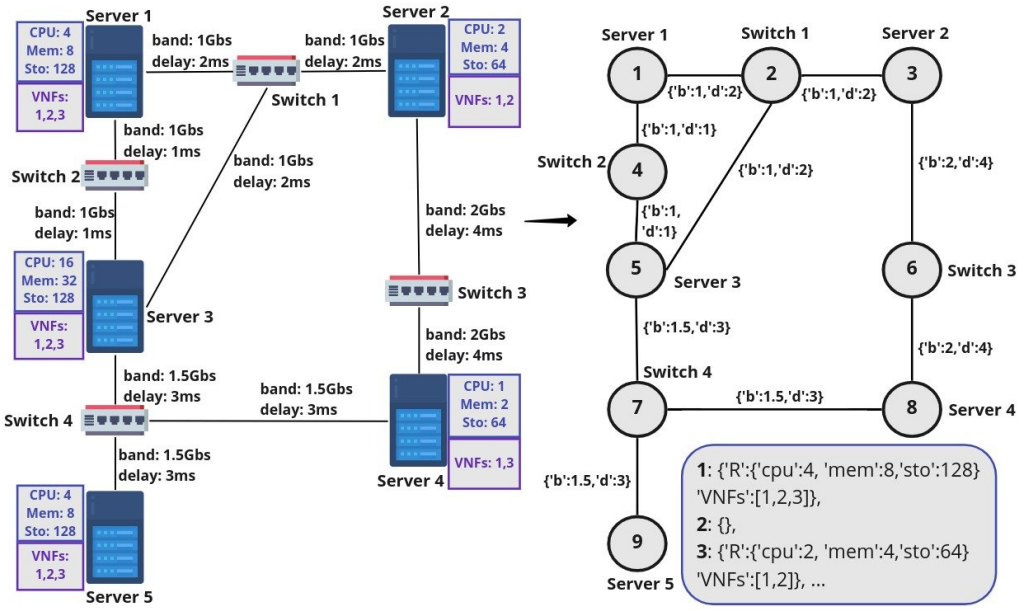
The physical links (L_{net}) provide connectivity between two physical nodes and are characterized by the bandwidth capacity and delay. The bandwidth, B_l, of a link l between the source and the destination nodes is consumed by the data flow between the VNFs placed in those nodes, since these VNFs belong to the same SFC. The link delay, D_l, defines the time to send data between the source and destination nodes connected by the link l.

We model the physical infrastructure as an undirected graph, G_{net}, as illustrated in Figure 30, which shows five servers connected by three switches totalling eight physical nodes. Each server lists the respective computational resources and the list of VNF types that it can receive. One can note that the switches don't have information about the computational resources; we omit this information in the figure because it is not capable of receiving any VNF type. The physical nodes present in the infrastructure are represented as nodes in the undirected graph, each one with a unique ID. Please note that while the switches are not able to receive the VNFs, they are modelled in the graph as they provide connectivity between other nodes. The edges in the graph are identified using the IDs of the nodes that are connected. For instance, the link that connects Server 1 to Switch 1 is represented by the ID *(1,2)*, the associated IDs of these nodes in the graph.

We store the information about the nodes and links in a JSON file, which allows storage of a substantial variety of characteristics about the physical infrastructure.

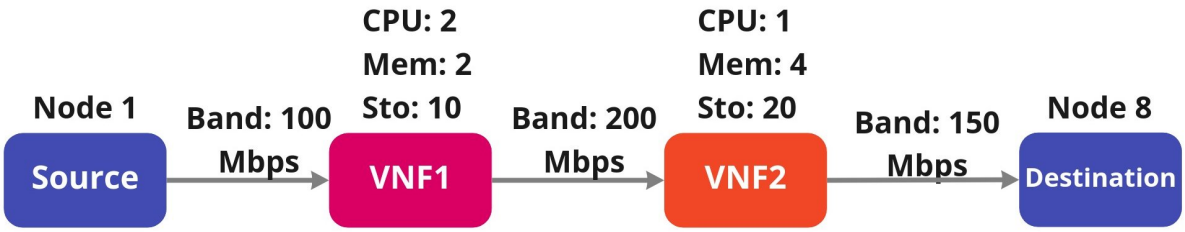
The SFC request specifies the requirements from a customer with respect to VNFs and generally. Figure 31 shows an example of an SFC request. The customer must specify the source and destination nodes, which define the nodes of the physical infrastructure where the traffic that will be processed by the SFC start and end. Then, the customer specifies the ID of the source and destination of the physical nodes, which, in this case, are 1 and 8, respectively (see Figure 31). We assume that these IDs are related to the IDs of the nodes in the physical infrastructure graph. For instance, in Figure 30, if the customer defines the source and destination as nodes 1 and 8, the source and destination of SFC are, respectively, Server 1 and Server 4.

Figure 30 – Sample physical infrastructure and associated graph.



Source: the author (2023).

Figure 31 – SFC request example.



Source: the author (2023).

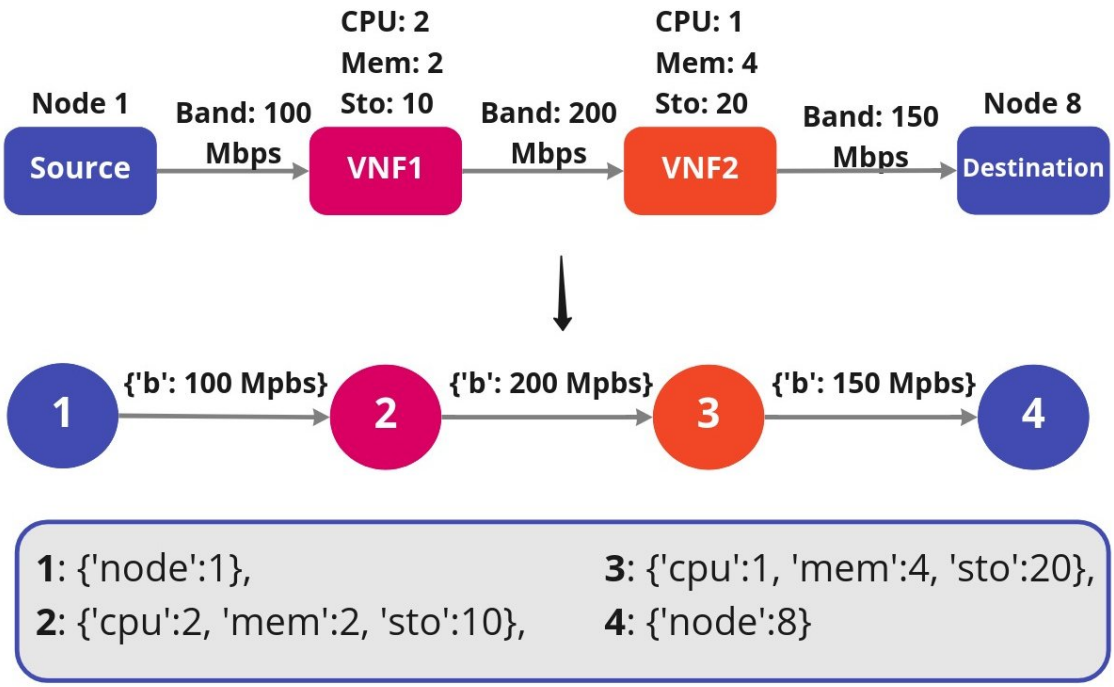
Next, the customer defines the functions that comprise its SFC by creating a list of VNFs and their order. For each VNF, the customer specifies a type and the computational requirements in terms of CPU, memory, and storage. The information about the computational requirements can also be defined by the network manager and made available to customers in VNF template formats. In Figure 31, the SFC is composed of just two VNFs, VNF1 and VNF2, each one with associated computational requirements.

The customer also defines requirements about the links between the source, the destination, and the VNFs, referred to as virtual links. In Figure 31, the virtual link specify requirements in terms of bandwidth, but other requirements could be defined. This representation is flexible to model different operations that VNFs can perform on the traffic and the requirements between the VNFs. For instance, by taking into account the bandwidth requirements, the customer could define the behaviour of the traffic data size after the processing of VNFs; the

size could change depending on the function applied, e.g., data compression or addition of headers.

We represent the SFC request as a tuple $sfc_{req} = \{N_{src}, N_{dst}, G_{sfc}\}$. N_{src} and N_{dst} are, respectively, the source and destination nodes from physical infrastructure specified by the customer. G_{sfc} is the directed graph that models the SFC request, as shown in Figure 32. We use a directed graph to model the order of the VNFs as defined by the customer. The VNFs are represented as nodes in the graph (N_{sfc}). Each node has the tuple $req_v = \{Vcpu_v, Vmem_v, Vsto_v, v \in 1, ..., N_{sfc}\}$, where $Vcpu_v$, $Vmem_v$, and $Vsto_v$ are the CPU, memory, and storage requirements of VNF v, respectively. The virtual links are the edges of the graph (L_{sfc}). For each edge, we define a bandwidth requirement of the virtual link, Br_f, where $f \in \{1, ..., L_{sfc}\}$.

Figure 32 – SFC request represented as a directed graph.

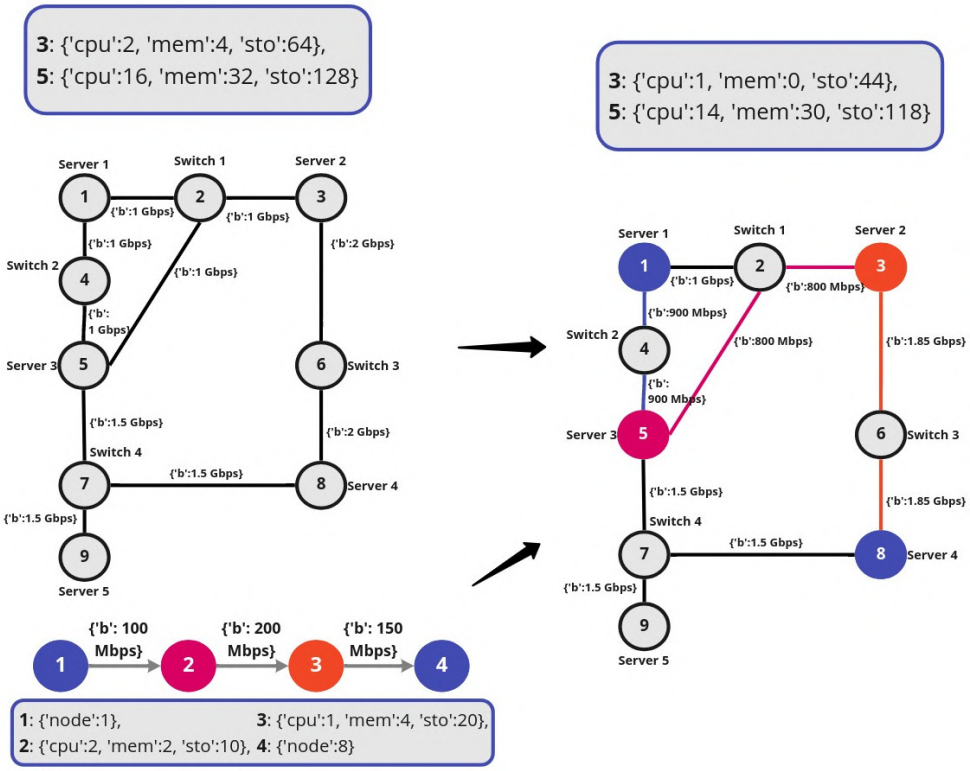


Source: the author (2023).

We store the requirements data about the VNFs and virtual links in JSON format. However, there is a difference between the format of the data about the source and destination nodes, and the VNFs. On one hand, the nodes in the graph that represent source and destination of the SFC request have only information about the physical node. On the other hand, the nodes in the graph that represent the VNFs have information regarding the computation requirements. Our SFC placement problem is basically a graph match where the SFC request graph needs

to be mapped into the infrastructure graph. Figure 33 illustrates a placement based on the infrastructure graph of Figure 30 and the SFC request graph of Figure 32.

Figure 33 – SFC placement represented as a graph matching.



Source: the author (2023).

The source node is Server 1 and the destination node is Server 4, since the customer specified these nodes as 1 and 8, respectively. The first step is to place the first VNF (node 2 of SFC graph), and then select the **candidate nodes**. The candidate nodes must have the minimum computational requirements and be able to receive the VNF type. As illustrated in Figure 30, all servers are able to place VNF1, and accordingly these are the candidate nodes for such a VNF.

To illustrate, let's assume that VNF1 is placed on Server 5. Now, the first virtual link must be placed on the physical links between Server 1 and Server 5. Many paths can be considered for the virtual link placement, but for simplicity let's assume the shortest path which passes through Switch 2. Please note that a virtual link can be placed on one or more physical links. Afterwards, VNF2 is placed on Server 2, and the virtual link between VNF1 and VNF2 is placed on the links through Switch 1. Since VNF2 is the last VNF, the last virtual link is placed between Server 2 and Server 4 (destination), through Switch 3.

After placing the VNFs on the server, the computational resources are consumed based

on the requirements of each VNF type. At the same time, the bandwidth of all physical links where a virtual link is placed is consumed. Updating physical infrastructure status is important because it impacts the selection of candidate nodes to place the VNFs of future SFC requests.

We consider the SFC placement as a graph matching problem, where the nodes of SFC requests graph need to be mapped into the infrastructure graph, the main challenge is to select the nodes of infrastructure graph for each VNF, i.e., the candidate nodes.

6.2.1 Problem Definition

For the SFC placement, we formulate a multi-objective problem, as showed by Equations 6.1-6.8:

$$min \left(\sum_{n=0}^{N_{net}} \sum_{v=0}^{N_{sfc}} X_{v,n}(C_n + C_v) \right) \tag{6.1}$$

$$min \left(\sum_{n=0}^{N_{net}} \sum_{v=0}^{N_{sfc}} X_{v,n} E_n \right) \tag{6.2}$$

$$s.t.$$

$$1 =< \sum_{n}^{N_{net}} X_{v,n} <= 2, v \in \{1, ..., N_{sfc}\} \tag{6.3}$$

$$Av(X_{v,n}) >= Av_{req} \forall n \in \{1, ..., N_{net}\}, \forall v \in \{1, ..., N_{sfc}\} \tag{6.4}$$

$$\sum_{v}^{N_{sfc}} X_{v,n} \times Rcpu_n \leq Vcpu_v, \, \forall \, n \in N_{net} \tag{6.5}$$

$$\sum_{v}^{N_{sfc}} X_{v,n} \times Rmem_n \leq Vmem_v, \, \forall \, n \in N_{net} \tag{6.6}$$

$$\sum_{v}^{N_{sfc}} X_{v,n} \times Rsto_n \leq Vsto_v, \, \forall \, n \in N_{net} \tag{6.7}$$

$$\sum_{l}^{L_{sfc}} Y_{lp} \times Br_f \leq bw_p, \, \forall \, p \in L_{net} \tag{6.8}$$

Equations 6.1 and 6.2 are the main goals of our problem. Equation 6.1 calculates the placement cost. $X_{v,n}$ is a binary variable that define if the VNF v is placed in the physical

node n, C_n is the cost of each node n, and C_v is the cost of each VNF v. Then, we sum all costs to calculate the overall placement cost.

The second objective is to minimize the energy consumption of the placement, which is described in Equation 6.2. E_n is the energy consumption of node n which is considered for the placement. If the node n was considered, its energy consumption, E_n, is calculated as shown in Equation 6.9. We assume that the CPU ($Ecpu_n$) and memory ($Ememory_n$) operation of the node impacts on the energy consumption.

$$E_n = Ecpu_n + Ememory_n \tag{6.9}$$

We also consider constraint functions in our optimisation problem. Equation 6.3 defines the maximum number of VNFs of the same type that we can place in different nodes. Since $X_{v,n}$ is a binary decision variable, only one instance of the same VNF can be placed in a node. We define that, for each VNF, the maximum of two replicas can be placed in different nodes and at least one replica must be placed. In Subsection 6.4.1 we explain the redundancy mechanism adopted in our placement solution in order to guarantee the required SFC availability.

Equation 6.4 defines that the availability of the SFC ($Av(X_n, v)$) placed must be higher than or equals to the requested availability (Av_{req}). Please note that the availability depends on the physical nodes chosen for the placement and the VNFs instances that will be placed. This equation is directly related to the two objective functions, because as more servers and VNF instances are considered to increase the availability, the greater the energy consumption and cost. The availability of systems can be estimated using different techniques, and in Subsection 6.4.4 we describe how we use SPN models to estimate it.

Equations 6.5, 6.6, and 6.7 define constraints about CPU, memory, and storage, respectively. The binary variable $X_{v,n}$ defines if the node n is considered to place the VNF v. Then, the sum of amount of resources for each component of that node n ($Rcpu_n$, $Rmem_n$, $Rsto_n$ for CPU, memory, and storage, respectively) must be equals or higher than the resources required by the VNF v ($Vcpu_v$, $Vcpu_v$, and $Vcpu_v$ for requirements about CPU, memory, and storage, respectively). Equation 6.8 defines that the bandwidth required by the virtual link Br_f must be lower than the bandwidth available (bw_p) in the physical link p. The binary variable Y_{lp} defines if the physical link p is considered to place the virtual link l.

We propose the Cand-RL algorithm to solve the SFC optimisation problem. The goal is to minimize placement cost and energy consumption while meeting availability requirements.

In the next section, we present the Cand algorithm to select the candidate nodes during SFC placement. In Subsection 6.4, we describe how the SFC placement problem is modelled to be solved with RL, defining the state representation, the action representation and reward function. In Subsection 6.5 we describe the Cand-RL algorithm, which combines the Cand algorithm and use an RL agent to define the SFC placement (nodes to place the VNFs) and the redundancy strategy to meet the availability requirements.

6.3 THE CAND ALGORITHM FOR SELECTING CANDIDATE NODES

The Cand algorithm is used to select sequentially the candidate nodes to place a VNF; its pseudocode is shown in Algorithm 2.

Algorithm 2: Pseudocode of Cand algorithm

```
Input  : vnf_requirements, flow_entry_requirements, infra_graph, source_node,
         destination_node, k
Output: candidate_nodes
1  begin
2      infra_graph ← get_subgraph_enough_band (infra_graph,
         flow_entry_requirements)
3      paths_lenght ← calc_shortest_path_length (infra_graph)
4      scores ← {∅}
5      for phy_node ∈ infra_graph.nodes do
6          meet_requeriments ← False
7          if vnf_requirements.type ∈ phy_node.supported_vnfs then
8              meet_requeriment ← True
9          end
10         for req_name, req_value ∈ vnf_requirements.computational_req do
11             if phy_node[req_name] < req_value then
12                 meet_requeriment ← False
13                 break
14             end
15         end
16         if meet_requirement == True then
17             scores[phy_node[id]] ← paths_lenght[phy_node[id]][source_node] +
                 paths_lenght[phy_node[id]][destination_node]
18         end
19     end
20     scores ← sort(scores)
21     candidate_nodes ← get_k_nodes (scores, k)
22     return candidate_nodes
23 end
```

The Cand algorithm receives several input data. The computational requirements of a VNF, in terms of CPU, memory, and storage are modelled as a vector of numbers, $vnf\_requeriments$. The requirements about the virtual link, $flow\_entry\_requirements$, is a number that defines the bandwidth required by the VNF. However, if necessary, more requirements about virtual links can be specified; in this case, the input variable $flow\_entry\_requirements$ would become a vector. The infrastructure graph is also an input of the Cand algorithm as a variable $infra\_graph$. The source and destination nodes of a SFC request are the input of the Cand algorithm ($source\_node$ and $destination\_node$, respectively). Finally, the last input is the parameter k, which defines how many candidate nodes will be selected for the VNF. The output of the Cand algorithm is a list of the IDs of candidate nodes.

The first step of the Cand algorithm is to get a subgraph from the infrastructure graph where all edges have enough bandwidth to place the virtual link. The function $get\_subgraph\_enough\_band()$ checks all edges of the input graph and returns a subgraph, where all edges have bandwidth higher or equals to $flow\_entry\_requirements$. The infrastructure graph then receives this subgraph as shown in Line 2.

We select the candidate nodes based on a **score**, which is the sum of the distance from the source to the candidate node and the distance from the candidate node to the destination node. The idea is to select candidate nodes near to the shortest path between the source and destination nodes of the SFC in order to reduce the consumption of the physical links in the network.

Then, we calculate the shortest path length of all-pairs in the infrastructure graph using the Dijkstra algorithm as shown in Line 3. In Line 4, we declare the $scores$ variable as an empty map. This variable will store the score of all nodes in the infrastructure graph.

The loop in Line 5 iterates over all physical nodes in the infrastructure, considering the links that meet the bandwidth requirements of virtual link. The goal is to evaluate what nodes meet the computational requirements of the VNF and check what nodes can receive the VNF.

In Line 7, we check if the physical node can receive the VNF based on its type. In a positive case, the variable *meet_requirements* is true. In the loop of Line 10, we iterate over all computational requirements of the VNF. In Line 11, we check if the node has less resources than required by the VNF, considering all resources types (req_name) and the value specified in the VNF requirement (req_value). If the node does not have enough computational resource of any type (CPU, memory, or storage), the variable $meet\_requirement$ is false, and this node will not be considered as a candidate.

Then, we check if the physical node meets the requirements of the VNF and can receive the VNF in Line 16. When this happens, the score of the node is calculated as the sum of the distance from the source node to the node and the distance from the node to the destination node. The score is stored in the variable $scores$, where it is associated with the node ID in Line 17. It is important to highlight that the *if* in Line 16 is there to ensure that only the score nodes that meet the requirements of the VNF are calculated.

The score of candidate nodes are sorted in Line 20 as the nodes score in descending fashion. Thus, the first K nodes are selected in Line 21, i.e., the nodes with the lowest scores. There is a possibility of the number of candidate nodes being lower than k. In this situation, we complete the $candidate\_nodes$ list with a fake ID (-1) in Function get_k_nodes until its length equals k. This is needed for the correct working of the RL agent; more details are provided later in Section 6.4.3. Finally, the Cand algorithm returns the IDs of candidate nodes in Line 22.

6.3.1 An Illustrative Example of How the Cand Algorithm Works

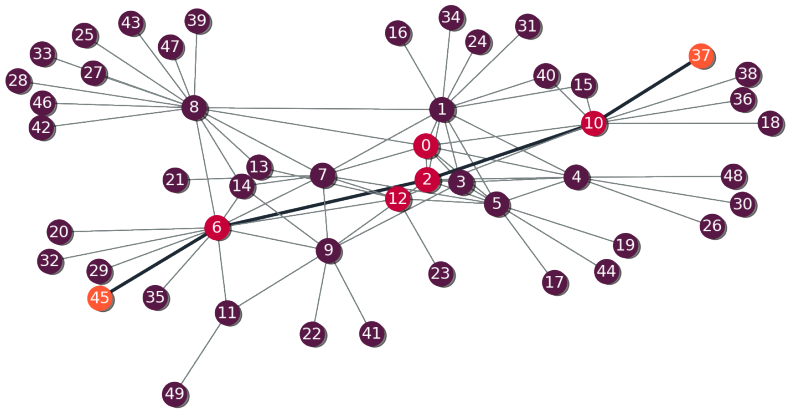
To illustrate how the concept of the score used in Algorithm 2 works, Figures 35(a) and 35(b) illustrate the candidate node selection for a VNF assuming a graph with 50 nodes. The graph models an Authonous System (AS) network (ELMOKASHFI; KVALBEIN; DOVROLIS, 2010) generated using the NetworkX tool[1]. The source and destination nodes are nodes 45 and 37, respectively, and they are highlighted in orange in the graph. For illustrative purposes, we assume that all physical nodes in the graph can receive the VNF and have enough computational resources.

Figure 35(a) shows the candidate nodes selected (in pink) with $k = 5$. The black edges in the graph are the shortest path between the source and destination nodes, and the selected nodes are located close to this path. The nodes that are in the shortest path (10, 2, and 6) were selected as candidate nodes. If the VNF was placed in one of these nodes, only links close to the shortest path between destination and source nodes would be occupied by the virtual link of this VNF. If the VNF were allocated to a node very far from the source and destination nodes (for example, Node 28), the virtual link of that VNF would use too many links to connect the source to the chosen node.

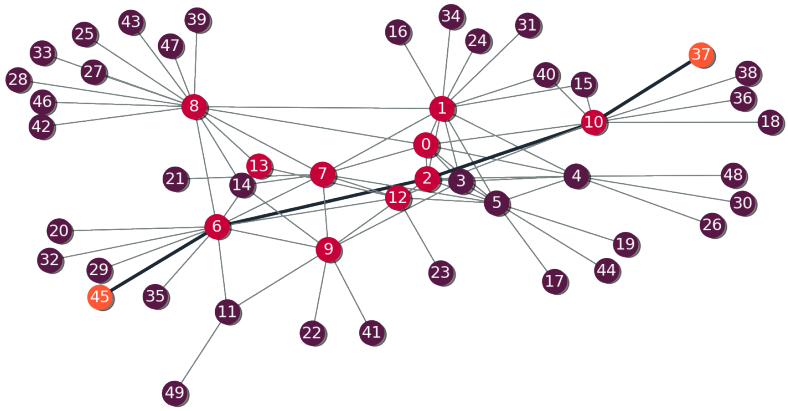
The k value can be increased if more candidate nodes can be considered to place the VNF.

[1] <https://networkx.org/documentation/stable/index.html>

Figure 34 – Selection of candidate nodes.



(a) $k = 5$



(b) $k = 10$

Source: the author (2023).

Figure 35(b) shows the result of candidate selection with $k = 10$. One can note that the same nodes selected with $k = 5$ were selected in addition to more five nodes (1, 7, 13, 9, and 8), but all nodes are close to the shortest path between the source and destination nodes. The difference is that these additional nodes have a higher score than the nodes shown in Figure 35(a).

It is important to highlight that in some cases, the candidate nodes for a VNF can be located far away from the shortest path between the source and destination nodes. Consider the graphs illustrated in Figures 35(a) and 35(b). If the only nodes that met the requirements of the VNF were, for example, Nodes 28, 33, 25, and 43 (in the upper left corner of the graph), these nodes would be selected in the loop of Line 5 of Algorithm 2, and their scores calculated. Therefore, even if the candidates nodes are located far away from the source and candidate nodes, they will be selected and ordered based on their score by the Cand algorithm.

Another possibility is a situation where any node can receive the VNF, e.g, if all nodes that can support such type of VNF do not have enough computational resources. In this case, the variable $meet\_requeriments$ will be false for all nodes of the physical infrastructure graph, and consequently the variable $scores$ will be empty at the end of the Cand algorithm. In this case, we assume that the VNF cannot be placed in the infrastructure, and the SFC request is rejected.

Using the Cand algorithm, we can restrict the number of nodes to place on a VNF. This is useful in large-scale infrastructure situations where we may have hundreds or thousands of possible nodes to allocate a VNF. As we will explain in the next subsection, we can use RL algorithms to define the best candidate node to place a VNF and define if a backup replica needs to be placed in order to meet the availability requirements. However, if we consider the multidimensional information about all nodes of the infrastructure (e.g. CPU, memory, storage, among others) and a large scale infrastructure, with dozens, hundred or even thousands of nodes, both input and output could have high dimensionality, and training neural networks on high-dimensional data poses a significant problem (WÓJCIK; KURDZIEL, 2019). Thus, we can use the Cand algorithm to select only nodes that have the computational capacity to allocate the VNF, that accept that type of VNF and that are close to the shortest path between the source and destination nodes of the SFC. It contributes to the reduction of input and output dimensionality data of RL agent, since the input of RL agent will be the information about the candidate node to place a VNF.

In the next subsection, we will describe how we can use an RL agent to select the candidate nodes to place the SFC and then we will describe how the Cand algorithm is used with the RL agent for the SFC placement.

6.4 RL FOR SFC PLACEMENT

We use RL to create an agent to select the best candidate node to place a VNF. The Cand algorithm explained in the previous section is used to select the candidate nodes to avoid having the agent evaluate all nodes in the physical infrastructure. In the next subsections, we present the characteristics of SFC requests and the RL formulation for the SFC placement problem for large-scale scenarios. We define the environment state representation, the action representation, and the reward function.

6.4.1 Characteristics of SFC requests

We represent the SFC request detailed in Section 6.2 as a MDP based on Xiao et al. (XIAO et al., 2019b), where the VNFs are processed sequentially. For the SFC request $\{N_{src}, N_{dst}, G_{sfc}\}$ that arrives at time T, we split it into subrequests for each VNF in N_{sfc}. Then, each subsresquest is processed in different sub slots $\{T_1, T_2, ...T_N\}$, where N is the number of VNFs.

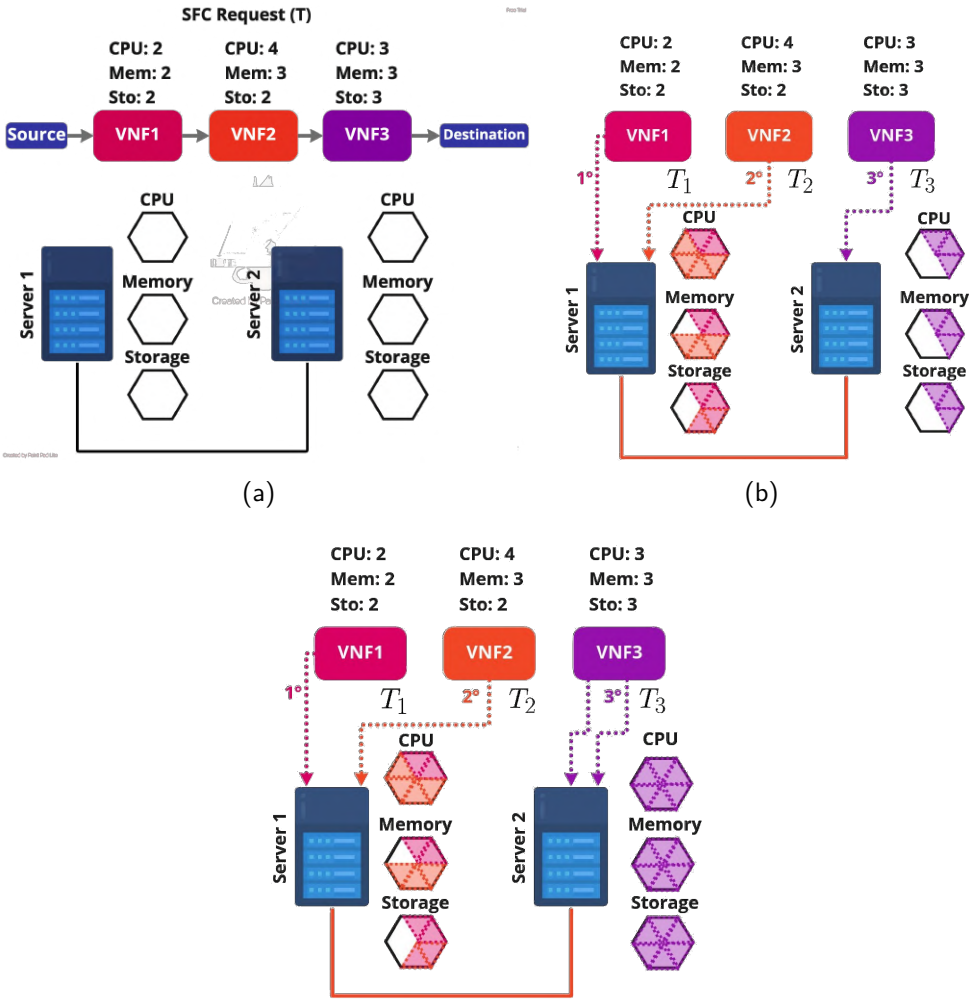
When a subrequest is processed, a physical node ($n_i \in N_{net}$) needs to be selected to place the respective VNF. The node needs to meet the requirements for CPU, memory, and storage. After placing the VNF in the node, its resources are consumed, changing the network status. The changes of network status are related to the different states of the MDP. For each MDP state, the candidate nodes are selected to place a new VNF, and when it is placed, the resources of nodes will be consumed changing the MDP states.

After place a VNF in the node, the respective virtual link must be placed. The virtual link is placed in the same subrequest of the VNF, consequently, in the same sub slot time. If the previous VNF was placed in node n_j, where $n_j \in N_{net}$, the links that connect the node n_j and n_i would be selected to place the virtual link. Then, the resources of physical links are consumed, i.e. the available bandwidth ($B_{i,j}$), according to the virtual link requirement (Br_i).

In order to increase the overall SFC availability, many replicas can be considered and placed on the physical infrastructure. We assume that VNF instances of the same type can assume two different roles. The main instance is connected to the VNF instances of different types, and is responsible for receiving/forwarding the flow from/to other VNFs. The backup instance of a VNF works only in cases where a failure occurs in the main instance. As the backup instance keeps the SFC working in the case of main instance failure, the SFC is impacted when the main instance and all backup instances fail. The decision to place a backup instance of a VNF is done in the same subrequest and sub slot time. We assume that both main and backup instance of the same VNF type consume the same computing resources from the respective nodes.

An example is illustrated in Figure 35. We assume an SFC request composed of three VNFs and an infrastructure composed of just two servers connected by a physical link as shown in Figure 36(a). As discussed previously, the servers have limited capacity in terms of CPU, memory, and storage. For the sake of simplicity, we assume that the two servers can receive all VNFs from the SFC request, i.e., they are candidate nodes for all VNFs types. To aid comprehension, we do not present the source and destination nodes of SFC in Figure 35.

Figure 35 – SFC Request represented as an MDP.



Source: the author (2023).

The SFC request arrives at time T, and is divided into three subrequests, one for each VNF as illustrated in Figure 36(b). These three subrequests are processed in the same instant time T, but we divide it in three sub slots, T_1, T_2, and T_3, in order to define the processing order. The first VNF, VNF1, is processed and placed in Server 1, consuming computational resources required by VNF1.

Next, the subrequest for VNF2 is also placed in Server 1, consuming the resources of this server. Since VNF1 and VNF2 are placed in the same server, no links will be affected by the virtual link between these two VNFs.

Finally, the last subrequest (VNF3) is processed and the complete SFC is placed in the physical infrastructure. The virtual link from VNF2 to VNF3 is placed in the physical link that connects Server 1 and Server 2.

While the placement illustrated in Figure 36(b) assumes just one replica of each VNF, Figure 36(c) shows an example of a placement where main and backup instances of VNF3 are placed in Server 2 in order to increase the SFC availability.

The approach of using redundant replicas can increase SFC availability however it also increases the resource consumption of servers, and may increase the overall energy consumption (SANTOS et al., 2021). Therefore, an intelligent approach to find a balance between availability and resource consumption should be applied in the context of SFC placement to meet the SLA requirements of customers and to mitigate the risk of over-consumption. Based on the candidate nodes selected by the Cand algorithm, we use RL to select the best candidate node to place each VNF and also define with a redundant replica needs to be allocated. It is expected that the agent RL, after training, will be able to select the candidate nodes as well as the redundancy strategy to meet the availability requirements and also minimize the costs of allocation and energy consumption. In the next sections, we will describe the details about the RL implementation.

6.4.2 Environment State Representation

The state representation is the information that the RL agent receives from the environment to take an action at a time t, that will return a given reward in the long run (SUTTON; BARTO, 2018). This information must be relevant for the agent to take the right action for a given state.

In our context, the RL agent will decide the candidate node where the main instance of VNF from the SFC request will be placed and if a backup instance of such VNF will be placed in another candidate node. Therefore, the state representation relates to the list of candidate nodes for the VNF being processed. Equation 6.10 shows the state representation at time t (S_t).

$$S_t = \{R, C_i, A_s\}, i \in \{1, 2, ..., VNFs_s\}, s \in \{1, 2, ..., SFCs\} \tag{6.10}$$

where R is a matrix composed of the percentage of available computational resources of each candidate node:

$$R = \begin{bmatrix} CPU_1 & Mem_1 & Sto_1 & Av_1 \\ \vdots & \vdots & \vdots & \vdots \\ CPU_k & Mem_k & Sto_k & Av_k \end{bmatrix}$$

where k is the number of candidate nodes. We assume three computational resources for each node (CPU, memory, and storage) and the availability of the node. To calculate the percentage of available computational resources of candidate nodes, we divide the total of resources of a type by the number of available resources of this type. The goal is to avoid using absolute values of resources (for example, memory and storage in terms of GB, or CPU in terms of number of cores), and use values between 0 and 1. As modern RL agents are typically implemented using neural networks, the input normalization must be used to disentangle the natural magnitude of each component from input data and improve the learning process (HASSELT et al., 2016). Accordingly, these input values were normalized. As mentioned previously, there is a possibility that the number of candidate nodes is lower than k. In this case, we fill the matrix R with zero-valued rows until we have k rows in the matrix. For example, if three candidate nodes are selected and $k = 5$, two zero-valued rows are added in matrix R.

C_i is a vector in Equation 6.10 with the requirements of the VNF i that is being processed at time t from the SFC request s. The vector is composed of the CPU, memory, and storage values required by the VNF. A_s is the availability requirement of the overall SFC s from the list of all SFC requests $SFCs$.

The matrix R gives a picture of the available resources of the candidate nodes for the RL agent. The agent can select a candidate node with more or less resources depending on the requirements of the VNF, which is specified in the vector C_i. Based on the availability value, A_s, the agent can decide if the VNF will be duplicated or not in order to achieve the availability required for the SFC.

6.4.3 Action representation

The action representation is a numerical representation of the action that the agent can take at a given time, t, and affect the future state of the environment, and will give a reward in the future (SUTTON; BARTO, 2018). In our context, the action relates to the placement decision for each VNF sequentially, as illustrated in Equation 6.11.

$$A_t = \{s, r\}, s \in \{0, 1, ..., k\}, r \in \{0, 1\} \tag{6.11}$$

The agent action, A_t, at time, t, is composed of just two values. s is the index of node from the $candidate\_nodes$ list where the VNF will be placed, assuming k candidate nodes. As mentioned in Section 6.3, in cases where there are less candidate nodes for a VNF than k, the $candidate\_nodes$ list is padded with a fake ID until its size equals to k. It is needed because to implement the RL agent, we need to define lower and upper bound values for s. Therefore, when the $candidate\_nodes$ list has a size equal to k, the agent will always select a valid candidate node to place the VNF. When the size of the $candidate\_nodes$ list is lower than k, the RL agent can select the "fake ID" from the list. In this case the VNF will not be placed in a node, and the agent will be penalized as we will explain in Section 6.4.4.

r is the decision if the VNF will be replicated ($r = 1$) or not ($r = 0$). The strategy to define where the VNF backup instance will be placed can vary depending on the network operator's goals. For instance, the backup instance can be placed in the same node of the main instance or in another candidate node.

6.4.4 Reward function

In the context of RL, reward is a signal that the environment returns to the agent after it takes an action. The reward can be negative or positive depending on the action taken at a time, t, given a state of the environment. The agent's goal is to maximize the total reward it receives over the long run (SUTTON; BARTO, 2018).

For the SFC placement, the reward is defined based on the success of the agent in placing an SFC that meets the availability requirements. However, adding replicas can increase OPEX due to the cost to place a VNFs instance in the node and associated greater energy consumption (ERAMO; AMMAR; LAVACCA, 2017). Therefore, our reward function takes into account the availability of the SFC, the availability required, and the energy consumption and cost resulting from the SFC placement defined by the agent.

Equation 6.12 shows our reward function. A reward signal is returned to the agent after the placement of each VNF. We have different reward values if the VNF is the last VNF of an SFC or not. This difference is because when the last VNF of an SFC request is placed, the overall SFC availability, placement cost, and energy consumption can be calculated.

$$\mathcal{R}(s_t, a) = \begin{cases} -10, & \text{invalid node selected} \\ 0, & \text{valid node selected} \\ -100, & \text{r is rejected} \\ ((AV_{diff})\varrho(-O_{sfc})) + c, & \text{r is accepted} \end{cases} \tag{6.12}$$

Firstly, we will explain the reward for processing a non-final VNF from the SFC request. When the RL agent selects an invalid node to place the VNF (in cases where there are less candidate nodes than k), the agent receives a penalty of -10. On the other hand, if a valid candidate node is selected to place the VNF, the agent receives a reward of 0, i.e., it is not penalized.

When the last VNF from the SFC request is processed, the placement defined by the reinforcement agent can be evaluated. If the SFC is rejected, i.e., if any VNF is placed in an invalid node, the agent receives the highest penalty, -100. When all VNFs from the SFC requests are placed in valid nodes, we can evaluate the placement quality. To do this, we calculate the availability, cost, and energy consumption of the SFC placement defined by the RL agent.

As illustrated in Section 6.1, the availability is estimated through SPN models, considering the failure and repair events of nodes and VNF instances. The reader can also refer (SANTOS et al., 2021) and the Appendix A to see the evaluation of SPN models. Since the SPN models are considered for the SFC availability estimation, the constraint shown in 6.4 is not linear, which makes our problem non-linear. In addition, as we generate a different SPN model for each different SFC placement, the constraint 6.4 will be not the same for different solutions evaluations.

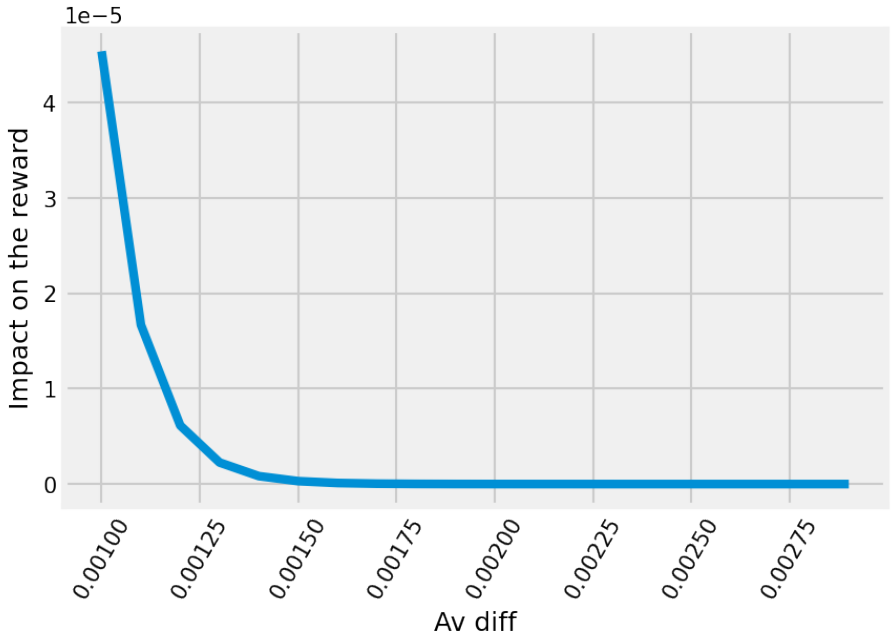
As mentioned in Section 6.2.1, we need to meet the availability constraint defined in the SFC request. Therefore, the availability of the allocated SFC must be greater than or equal to the requirement. However, if the availability is much higher than the requirement, the energy consumption and placement cost will increase. Based on the SFC availability, we need to define the impact on the reward based on the availability requirement for that SFC. This impact is defined in Equation 6.12 by AV_{diff}, which is detailed in Equation 6.13. When the SFC availability, av_{sfc}, is lower than the requirement, av_{req}, AV_{diff} will be $av_{sfc} - av_{req}$, i.e., AV_{diff} is negative, and it gets smaller as the availability of the SFC is less than the requirement. On the other hand, when av_{sfc}, is greater than or equal to av_{req}, the SFC availability meets

the availability requirement or surpasses it.

$$AV_{diff} = \begin{cases} av_{sfc} - av_{req}, & av_{sfc} < av_{req} \\ e^{-(av_{sfc} - av_{req})}, & av_{sfc} \geq av_{req} \end{cases} \tag{6.13}$$

As the availability difference increases, the SFC availability is higher than the requirement resulting in higher placement cost and energy consumption, since more VNFs will be placed. To avoid placements of SFCs with availability in excess of the requirement, the reward decreases following an exponential function as the availability difference increases, as shown in Figure 36. When the availability difference increases considerably, the impact on the reward tends to zero, and the RL agent will tend not to carry out this SFC placement, but rather those that approach the user's requirement.

Figure 36 – The availability difference impact on the reward.



Source: the author (2023).

We define the operational placement impact, O_{sfc}, as the sum of the energy consumption and the placement cost, which is illustrated in Equation 6.14.

$$O_{sfc} = C_{nodes}\sigma + C_{vnfs}\nu + E_{nodes}\varsigma \tag{6.14}$$

The placement cost for the nodes, C_{nodes}, is calculated using the first sum of Equation 6.1, while the placement cost for the VNFs, C_{vnfs}, is calculated using the second sum of Equation

6.1. The energy consumption for the nodes chosen for the placement, E_{nodes}, is calculated using Equation 6.2.

The node cost, VNFs cost, and energy consumption can have different scales, which can result in a different impact on the reward. In order to deal with this problem, we multiply the node cost, VNF cost, and energy consumption by constants, ς, σ, and ν, respectively, to bring these values to the same scale.

Since the goal of the RL agent is to maximize the reward, O_{sfc} is negative in Equation 6.12 because we try to minimize it. The greater the O_{sfc}, the greater the discount on the difference in availability as explained previously. Thus, the agent will try to minimize operating costs as it maximizes the value of SFC availability. We multiply the availability difference ($AV_{sfc} - \theta_s$) by the operational placement impact (O_{sfc}), however it is important to note that these values can be in different scales. Similar to the solution used in Equation 6.14, we multiply the availability difference by a constant ϱ, in order to use the same scale of the operational placement impact (O_{sfc}). The values of constants ϱ, ς, and σ, were defined empirically by evaluating the values calculated in each part of the equation with the aim of keeping all values on the same scale, and not favoring any factor.

6.5 THE CAND-RL ALGORITHM

We combine the Cand algorithm (Algorithm 2) and the RL agent described in Section 6.4 in order to create a unique algorithm for SFC placement - the Cand-RL algorithm, as described in Algorithm 3.

The input of the Cand-RL algorithm is the SFC request ($sfc\_request$) that the agent will process, the infrastructure graph where the SFC will be placed ($infra\_graph$), and the number of candidate nodes (k) that will be evaluated to place each VNF. The output of Cand-RL is a variable that indicates whether the SFC placement was successful or not.

The first step of the Cand-RL algorithm is to select the source and destination nodes that are specified in the SFC request as per Lines 2 and 3, respectively. These nodes will be used in the Cand algorithm to calculate the score of the candidate nodes to place each VNF, as explained in Section 6.3.

The loop in Line 4 iterates over all VNFs of the SFC request to process one by one sequentially. In Line 5, the candidate nodes to place the VNF are selected using the Cand algorithm. If there is no candidate nodes to place the processed VNF, the SFC request is

Algorithm 3: Pseudocode of the Cand-RL algorithm for SFC placement.

Input : sfc_request, infra_grapgh, k
Output: placement_success

```
1  begin
2      source_node ← sfc_request.source_node
3      destination_node ← sfc_request.destination_node
4      for vnf_request in sfc_request.vnf_requests do
5          candidates_nodes ← cand_algorithm (vnf_request.requirements,
            vnf_request.flow_entry_requirements, infra_graph, source_node,
            destination_node, k)
6          if length(candidate_nodes)==0 then
7              return False
8          end
9          candidates_resources ← get_nodes_resources (candidates_nodes,
            infra_graph)
10         action ← RL_agent ({candidates_resources, vnf_request.requirements,
            sfc_request.availability})
11         node_chosen ← action[0]
12         redundancy ← action[1]
13         infra_graph ← allocate_vnf (node_chosen, vnf_request, infra_graph)
14         links ← find_shortest_path (source, node_chosen,
            vnf_request.flow_entry_requirements, infra_graph)
15         infra_graph ← allocate_flow_entry (links, infra_graph)
16         if redundancy == 1 then
17             redundant_node ← get_redundant_node (candidates_nodes)
18             infra_graph ← allocate_vnf (redundant_node, vnf_request,
                infra_graph)
19         end
20         source_node ← node_chosen
21     end
22     links ← find_shortest_path (source, destination_node)
23     infra_graph ← allocate_flow_entry (links, infra_graph)
24     return True
25 end
```

discarded and the algorithm returns false. This condition is checked in Line 6.

As explained previously, the nodes selected by the Cand algorithm are potential candidates to place the VNF, but just one must be selected. The Cand-RL algorithm uses an RL agent to make such a decision. As detailed in Section 6.4.2, the RL agent considers the available resources of candidate nodes to take an action. Therefore, in Line 9, the function $get\_nodes\_resources$ returns the percentage of available computational resources of candidate nodes, considering the status of physical infrastructure. Since the Cand algorithm selects only k candidate nodes, we can control the input size of the RL agent irrespective of infrastructure size.

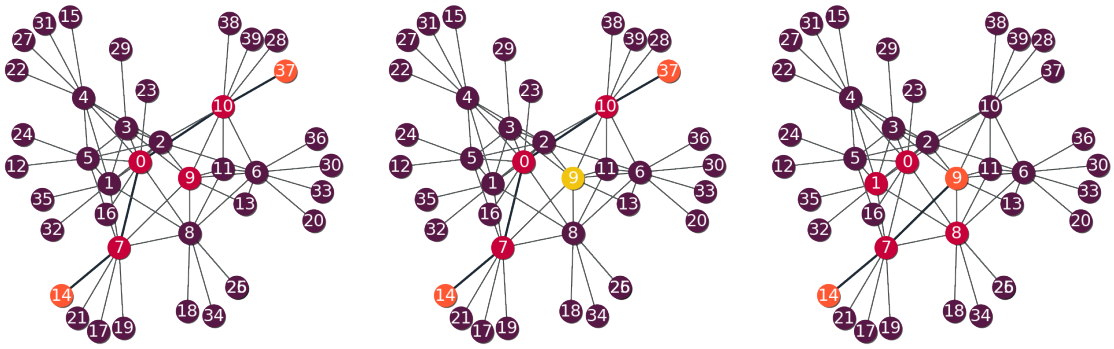
The RL agent takes an action in Line 10, taking into account the percentage of available resources, the VNF requirements, and the availability requirements of the SFC. The action defines the suitable candidate node to place the main VNF instance in and if a backup instance will be placed in another candidate node. To increase the overall SFC availability requires a decision to place a redundant VNF. The selected candidate node is extracted from the action in Line 11, while the decision to place a backup instance is obtained from the action in Line 12.

The VNF is placed in the chosen node in Line 13, consuming the resources from the physical infrastructure. The infrastructure graph status is updated after the VNF placement. Since the VNF is placed in the infrastructure, the virtual link from the source to the VNF needs to be placed in the physical links. The function, $find\_shortest\_path$, in Line 14, finds the shortest path from the source to the chosen node in the infrastructure graph, where the links in the shortest path must meet the requirements of virtual link in terms of bandwidth. To find the shortest path, we apply the Dijkstra algorithm. Afterwards, the virtual link is placed in Line 15, consuming the resources of physical links which are updated in the infrastructure graph.

In Line 16, whether the agent decided to place a backup instance of the VNF is checked. If the variable $redundancy$ is equal to 1, a backup instance will be placed, and we opt to place it in a candidate node different of the main instance. This is due to the fact that in situations where a failure happens in the node where the main instance is allocated, only that VNF instance will be impacted.

If the RL agent decided to place a backup replica, a node from the infrastructure graph must be selected in which to place it. In Line 17, the function $get\_redundant\_node$ selects the candidate node that has the highest quantity of available resource, and in Line 18 the

Figure 37 – Example of candidate nodes selection in Cand-RL algorithm.



(a) (b) (c)

Source: the author (2023).

backup replica is placed in the redundant node, which changes the status of the physical infrastructure.

After the placement of the VNF instance (main and backup, if defined by the RL agent), the new source will be the node chosen to place the main instance of VNF (Line 20). Thus, in the next iteration of the loop in Line 4, the Cand-RL will select candidate nodes (through the Cand algorithm) calculating their score from the new source (i.e., the chosen node to place the main VNF instance) to the destination.

Figures 38(a), 38(b), and 38(c) illustrate how the successive selection of candidate nodes works in the Cand-RL algorithm. We assume the infrastructure graph is composed of 40 nodes, and the source and destination nodes are, respectively, Nodes 37 and 14.

Assuming $k = 4$, Nodes 9, 7, 9, and 10 are candidate nodes to place the first VNF of an SFC request, as shown in Figure 38(a). If the RL agent chooses Node 9 to place the main instance (Figure 38(b)), Node 9 will act as the new source to select the candidate nodes for the next VNF of SFC.

In the next iteration of the loop in Line 4 of Algorithm 3, the source node will be Node 9, and the candidate nodes will be selected based on the shortest path between Node 9 and Node 14. As illustrated in Figure 38(c), Nodes 0, 1, 7, and 8 are the candidate nodes selected to place the second VNF.

After processing all VNFs, the virtual link between the last VNFs and the destination must be placed. Then, the shortest path is selected in Line 22, placed in the infrastructure in Line 23, and the algorithm returns *true* in Line 24, since the entire SFCs was allocated.

6.6 EVALUATION

The PPO algorithm is an evolution of the A2C algorithm; they are both actor-critic based RL algorithms. This class of RL algorithms combines a value-based approach and a policy-based approach (DU; DING, 2021). In actor-critic algorithms, two different functions (usually neural networks) are created with different roles - the actor and the critic. The actor network is responsible for optimising the policy model, $\pi(s, a)$, by defining an action at each time step according to the current state. The critic network evaluates the quality of action to optimise the value function $Q(s, a)$. In this way, since the critic learns which states are better, the actor uses this information to seek these states, avoiding bad states (HAN; LIANG, 2022). The PPO algorithm improves the A2C by implementing the idea of exploring the policy region in the optimisation problem during the agent training. It tries to make small updates using a clipped surrogate objective function thereby stabilizing the training process. For more information about the PPO algorithm, the reader can refer to (SCHULMAN et al., 2017).

The PPO algorithm has been proven to be very effective in dealing with different types of challenging problems, while being relatively simple to implement and tune (SCHULMAN et al., 2017; WANG; CAO; YANG, 2020), and also applied in recent works for the SFC placement problem (NING et al., 2022) (LI; KORDI, 2023). Notwithstanding this, Wang et al. (WANG; HE; TAN, 2020) note that it is less studied than other policy gradient methods. In this work, we use the PPO algorithm to create the RL agent in the RL-Cand algorithm.

6.6.1 Simulation setup

In this work, we use a simulation approach to evaluate the performance of the Cand-RL algorithm in an SFC placement task. We built a simulator using the Python language to represent the main aspects about the physical infrastructure and the SFC requests.

We assume that the arrival of SFC requests follows a Poisson process defined by a constant arrival rate, λ. Each SFC placed in the physical infrastructure has a lifetime defined by an exponential distribution with rate, $1/\mu$, where μ is the average SFC lifetime (LI et al., 2018). These parameters, λ and μ, are used to represent different types of customers, where each one has its own demand for SFC placement on the physical infrastructure.

Based on the λ and μ parameters, the simulator generates a list of SFC requests, each one with an associated arrival time and lifetime. Therefore, each SFC request has a time to be

processed by the Cand-RL algorithm and a time to be removed from the physical infrastructure (if it is placed). As explained in Section 6.4.1, each SFC request is divided into VNF requests that are processed sequentially by the Cand-RL algorithm. All VNF requests of an SFC request are processed in the same simulation time.

Regarding the physical infrastructure, the nodes may fail and be repaired during their operation. The failure and repair events are simulated through exponential distribution with constant failure and repair rates, respectively (LIMA; NETO; MACIEL, 2020; TORQUATO et al., 2019). For a given node in the physical infrastructure, the failure rate is defined by the inverse of the MTTF, while the repair rate is the inverse of the MTTR. These parameters are defined in the nodes of the physical infrastructure graph and can be modified in order to model different scenarios in our simulator.

The VNFs instances created in the nodes also have MTTF and MTTR values associated with them. Similar to the physical nodes, the failure and repair events of VNFs instances are defined by exponential distributions.

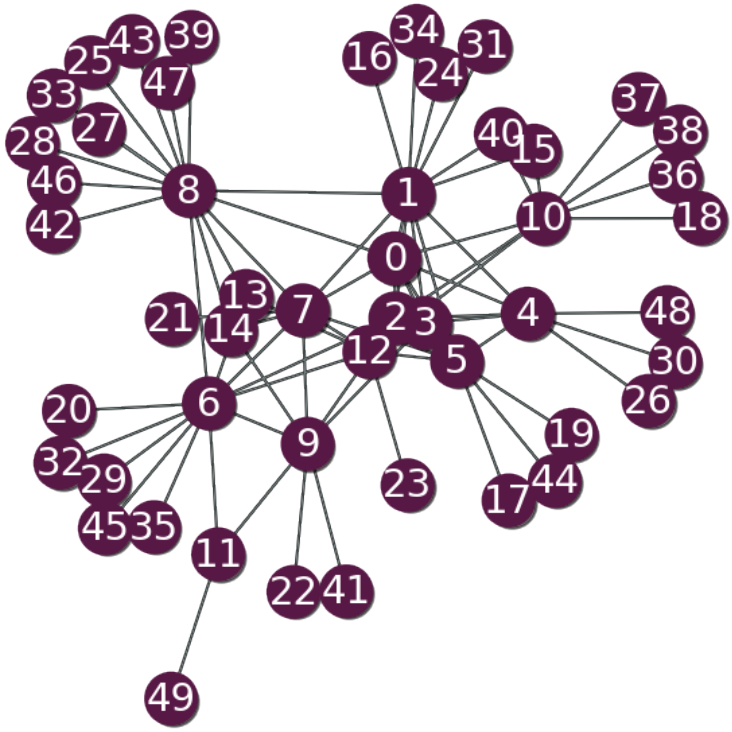
Table 16 presents the simulation parameters about the physical infrastructure of our basic scenario. It's important to highlight that we considered these parameters for the experiments in this thesis, but other parameters can be considered to simulate other scenarios, such as network topology, number of node, nodes and links configuration, etc. We assume a physical infrastructure graph generated using the NetworkX tool. The graph follows an Internet Autonomous System (AS) network (ELMOKASHFI; KVALBEIN; DOVROLIS, 2010) and has 50 nodes and 79 links, as illustrated in Figure 38.

Table 16 – Simulation parameters about the physical infrastructure.

Parameter	**Value**
Number of physical nodes	50
Network type	Internet AS network (ELMOKASHFI; KVALBEIN; DOVROLIS, 2010)
Number of CPUs of physical nodes	32 cores (GUO et al., 2019)
Memory size of physical node	64 GB (XIAO et al., 2019a)
Storage of physical node	300 GB (GUO et al., 2019)
Physical node MTTF	8760 h (ARAUJO et al., 2014)
Physical node MTTR	1.667h (ARAUJO et al., 2014)
Physical node energy consumption (CPU)	40 W (ALI et al., 2015)
Physical node energy consumption (memory)	30.17 W (ALI et al., 2015)
Physical node cost	1 (JIN; ZHU; ZHAO, 2019)
Link bandwidth	40 Gbps (ERAMO et al., 2017)

Source: the author (2023).

Figure 38 – AS graph with 50 nodes.



Source: the author (2023).

We assume the base number of CPU, memory, and storage as 32 cores (GUO et al., 2019), 64 GB (XIAO et al., 2019a), and 300 GB (GUO et al., 2019), respectively. Physical nodes have MTTF and MTTR values of 8760 hours and 1.667 hours, respectively (ARAUJO et al., 2014). These values are used as input for SPN models to calculate the overall SFC availability (see Section 6.4.4). As shown in Equation 6.9, we assume the total node energy consumption as the sum of CPU and memory energy consumption, and we assume these values as 40W and 30.17W, respectively (ALI et al., 2015). For simplicity, we also assume that when a physical node is chosen to place a VNF, its cost will be 1 (JIN; ZHU; ZHAO, 2019). The link capacity to place the virtual links from SFC requests is 40Gbps (ERAMO et al., 2017). It is important to highlight that these parameters could be easily updated to model different scenarios.

We consider a list of customers sending SFC requests to be placed. The customers have an arrival rate (λ) of 0.04 request/hours and the SFC average lifetime (μ) is 1000 hours (PALHARES et al., 2014). To compose the SFC, we assume four different VNFs types as presented by Tashtarian et al. (TASHTARIAN et al., 2019). The VNFs are related to Amazon EC2's flavors, and are illustrated in Table 17. Each VNF type has different requirements in terms of CPU,

memory, and storage. In addition, the different VNF types have relative prices according to the computational requirements.

Table 17 – VNF types.

VNF type	CPU (cores)	Memory (GB)	Price ($)
1	1	1	0.01
2	1	2	0.02
3	2	4	0.04
4	4	16	0.16

Source: the author (2023).

We also assume that different VNFs have different MTTF values. We assume that the VNF of type 1 has an MTTF value of 2880 hours (ARAUJO et al., 2014). For the other types of VNFs, we assume that as the price increases, the MTTF increases 5% based on the assumption that the more expensive a VNF is, the more reliable it is. Therefore, the MTTF value of VNF types 2, 3, and 4 are 3024 hours, 3175.2 hours, and 3333.96 hours, respectively. We also assume that the virtual links of the SFC request requires 1Gpbs from the physical links of infrastructure.

For each customer, we assume that the source and destination nodes of the SFC requested are generated randomly from all nodes of the physical infrastructure graph using a uniform distribution. The availability requirement of the customer is 0.9995%.

6.6.2 RL Agent Parametrization

We adjust the learning rate (α) and the discount factor (γ) of the PPO algorithm, since these parameters directly impact the learning when an agent is interacting with the environment. For the parametrization, we consider 50000 steps, which represents c. 650 SFC requests during the training. We assume SFC placement as an episodic task, where one episode is approximately 1000 hours. To evaluate the performance of an RL agent during the training, we consider the total reward at the end of an episode. We carried out 10 rounds of training to calculate the average reward at the end of each episode.

Figures 40(a)-40(i) show the parametrization results of the Cand-RL algorithm using the PPO algorithm. We also plot a regression line regarding the average reward at the end of each episode. The larger the slope of the line, the better the performance of the algorithm,

reflecting that it is learning to get good rewards faster. In general, the accumulated reward at the end of each episode increases during the training, which means that after some episodes the agent learned good strategies for SFC placement that met the customer requirements with low energy consumption and operational cost. It is important to note that depending on the parameter configuration, the reward growth may occur in different ways. The configuration $\alpha = 0.00005$ and $\gamma = 0.9$ presents a good performance (Figure 40(a)), since the reward grows fast and achieves the highest reward value in comparison with other parameter configurations (about 38). In addition, after Episode 60 the reward starts to grow faster, while in other configurations it takes more episodes to start the reward growth. As we increase the learning rate α (Figures 40(b) and 40(c)), the rewards in the later episodes are lower, never reaching more than 35.

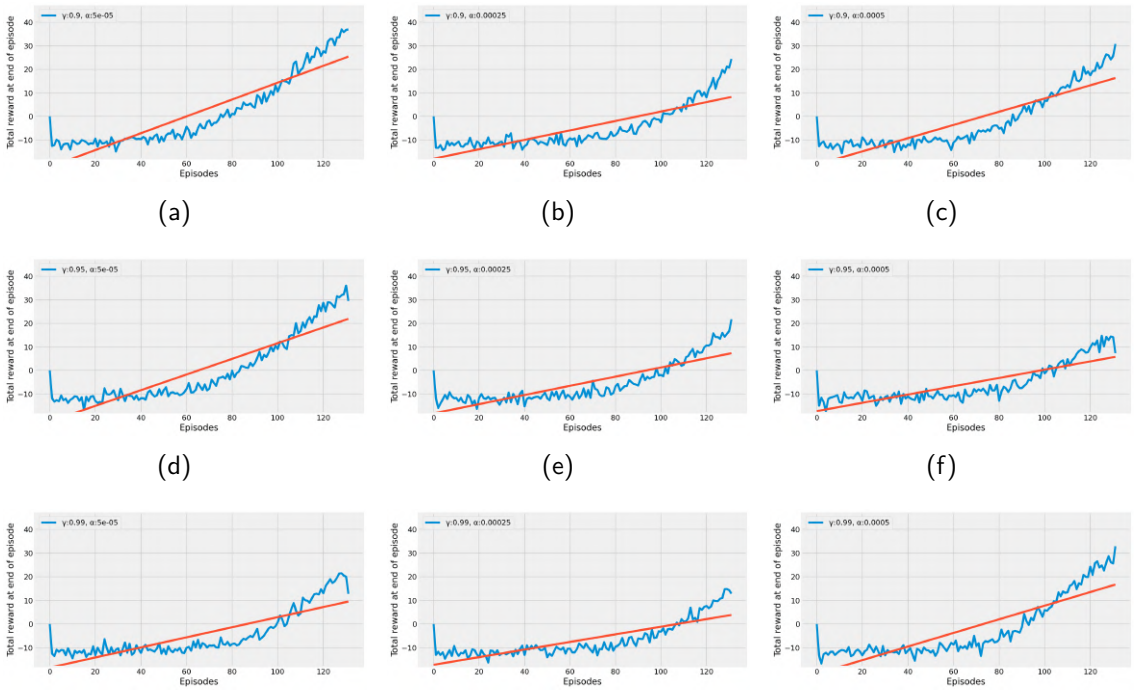
As we increase the discount factor (γ) to 0.95 (Figure 40(d)), the reward behavior is similar to $\gamma = 0.90$. However, by keeping $\gamma = 0.95$ and increasing the learning rate (Figures 40(e) and 40(f)), the reward grows slowly and reaches values of about 20, a clear degradation in the performance.

For $\gamma = 0.99$, the agent's learning deteriorates. For $\alpha = 0.00005$ and $\alpha = 0.00025$, as per Figures 40(g) and 40(h), respectively, the reward starts to grow only around Episode 100, and does not exceed 20. Therefore, in our SFC placement problem, a high discount factor value results in a degradation of agent's learning. The only exception is when $\gamma = 0.99$ and $\alpha = 0.0005$, as shown in Figure 40(i). In this configuration, the reward in the later episodes of training reaches around 30, but it starts to grow only after approximately 80 episodes, substantially different from the best configuration (Figure 40(a)).

When evaluating the learning rate impact, α, one can note that it does not have a constant impact on agent learning. For instance, evaluating when $\gamma = 0.9$, the increasing of α from 0.00005 to 0.0005 (Figures 40(a), 40(b), and 40(c)) results in minor degradation of agent learning, since the reward in the later episodes decreases. When $\gamma = 0.95$ and we increase the learning rate (Figures 40(d), 40(e), and 40(f)) the reward in the later episodes of training decreases, but not significantly. In contrast, for $\gamma = 0.99$ (Figures 40(g), 40(h), and 40(i)), increasing the learning rate improves agent learning, since the reward in the later episodes increases in comparison to $\alpha = 0.00005$ and $\alpha = 0.0005$. In sum, compared to the impact of the discount factor, learning rate impact is not as constant and significant.

We define two different strategies to define the best configuration for the agent using the PPO algorithm. The first strategy is based on the angular coefficient of linear regression,

Figure 39 – Parametrization results of Cand-RL algorithm using a PPO agent.



Source: the author (2023).

illustrated in Figure 40(a) - Figure 40(i). In this case, the best configuration presents the highest angular coefficient, i.e., the agent got the highest reward values from this configurations during training when compared to other configurations. The second strategy is based on the variance of the average reward at the end of an episode. The best configuration is the one that presents the lowest variance, i.e., there is a low oscillation in the reward during the training, a stabilization of the training. Table 18 shows the results of angular coefficient and variance of the PPO algorithm for the different parameter configurations.

The configuration $\gamma = 0.9$ and $\alpha = 0.00005$ presented the best results for the angular coefficient strategy. As shown in Figure 40(a), this configuration also obtains the highest reward during the training. For the variance, the configuration $\gamma = 0.99$ and $\alpha = 0.00025$ presented the best results. As shown in Figure 40(h), this configuration achieves a reward in the last episode of approximately 15. While other configurations achieve higher reward values, we use this configuration for benchmarking as it presents a stabilization of reward during the training.

Table 18 – Angular coefficient and variance results of the PPO algorithm for different parameter configurations.

Parameters configuration	Angular Coefficient	Variance
γ: 0.9, α: 0.00005	**0.3580**	223.2028
γ: 0.9, α: 0.00025	0.2002	83.0839
γ: 0.9, α: 0.0005	0.2808	146.5101
γ: 0.95, α: 0.00005	0.3332	199.7468
γ: 0.95, α: 0.00025	0.1950	76.9777
γ: 0.95, α: 0.0005	0.1761	64.2130
γ: 0.99, α: 0.00005	0.2132	96.4584
γ: 0.99, α: 0.00025	0.1609	**55.6205**
γ: 0.99, α: 0.0005	0.2878	161.4473

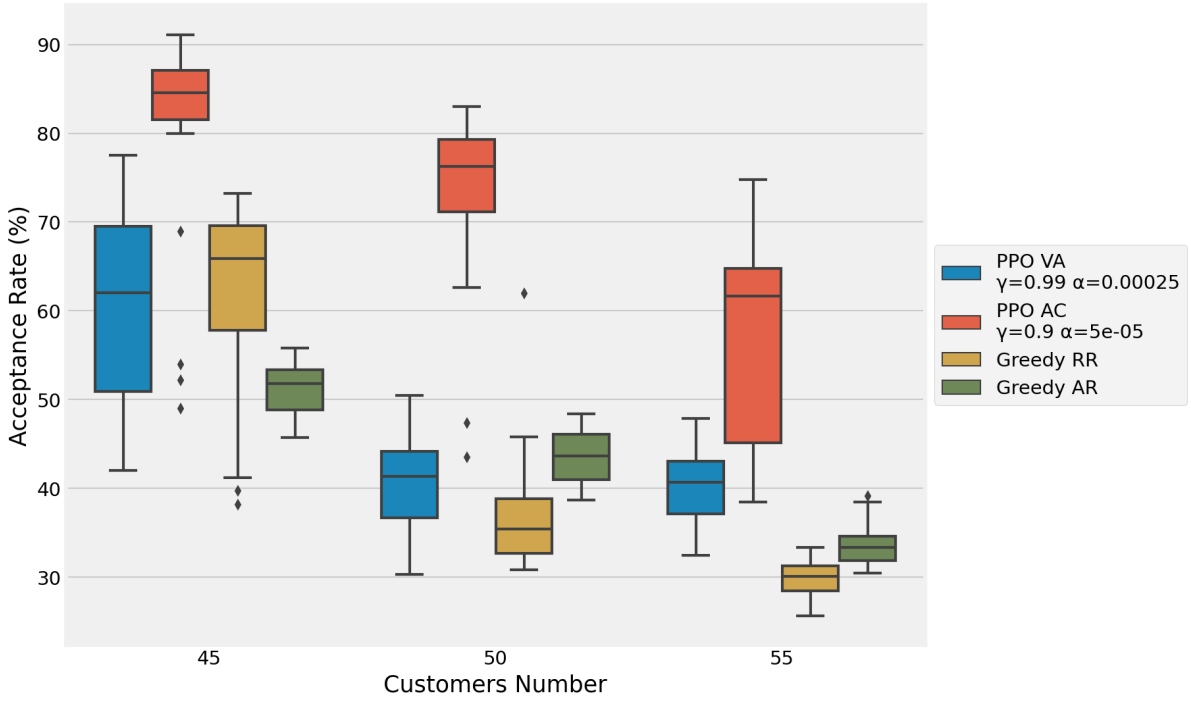
Source: the author (2023).

6.6.3 Scenario Variation

After defining the best parameters for the PPO agent in the Cand-RL algorithm, we varied the basic simulation scenario to assess performance. We also compare the Cand-RL algorithm with two variations of greedy algorithms. As mentioned by Gupta et al. (GUPTA et al., 2017) greedy algorithms are suitable for SFC placement, an NP-hard problem. The greedy algorithm selects the candidate node with the most resources. The first greedy algorithm variation adds a redundant VNF following a random distribution, which we label Greedy Random Redundancy (RR). The second greedy algorithm variation always adds a redundant VNF, which we label Greedy Always Redundancy (AR). We compare the acceptance rate of the two greedy algorithms and the Cand-RL algorithm using PPO with the best parameter configuration for the angular coefficient (AC) and the best parameter configuration for the variance (VA). We varied the number of customers sending SFC requests for the system as per Figure 40.

When there are 45 customers sending SFC requests to the system, the PPO AC presents the best acceptance rate with a median of 84.58%. The greedy RR presents the second-best acceptance rate with a median of 65.85%, followed by the PPO VA with a median of 62.06%. The greedy AR algorithm presents the worst performance with the lowest median acceptance rate with 45 customers, i.e., 51.79%. As illustrated in Figure 40(a), the Cand-RL with PPO AC achieved the highest reward. Therefore, the PPO AC obtains the best results when compared to the other algorithms. Since the greedy AR adds a redundant VNF, this algorithm consumes the nodes resources faster, making it difficult to place new SFCs, explaining the low acceptance rate. Evaluating the PPO VA and the greedy RR, these algorithms present

Figure 40 – Acceptance rate results for different numbers of customers.



Source: the author (2023).

a similar performance based on the median acceptance rate. As presented in Figure 40(h), the PPO configuration that presents the best results for the variance obtains low reward values at the end of the training, explaining the similar results to the greedy RR.

When we increase the number of customers to 50, the acceptance rate of all algorithms decreases. This is expected because more SFC requests will require more resources from the nodes in the infrastructure, which makes it difficult to find candidate nodes to place the VNF. The PPO AC still presents the best performance of the algorithms evaluated with a median acceptance rate of 76.26%; the median of other algorithms decreases considerably. The greedy AR presents the second best median acceptance rate (43.66%) followed by the PPO VA (41.35%) and greedy RR (35.40%). Even the greedy RR outperforms the PPO VA although the difference is relatively small at 2.31%. Based on the median acceptance rate, the Cand-RL algorithm using the PPO AC outperforms the two different greedy algorithms.

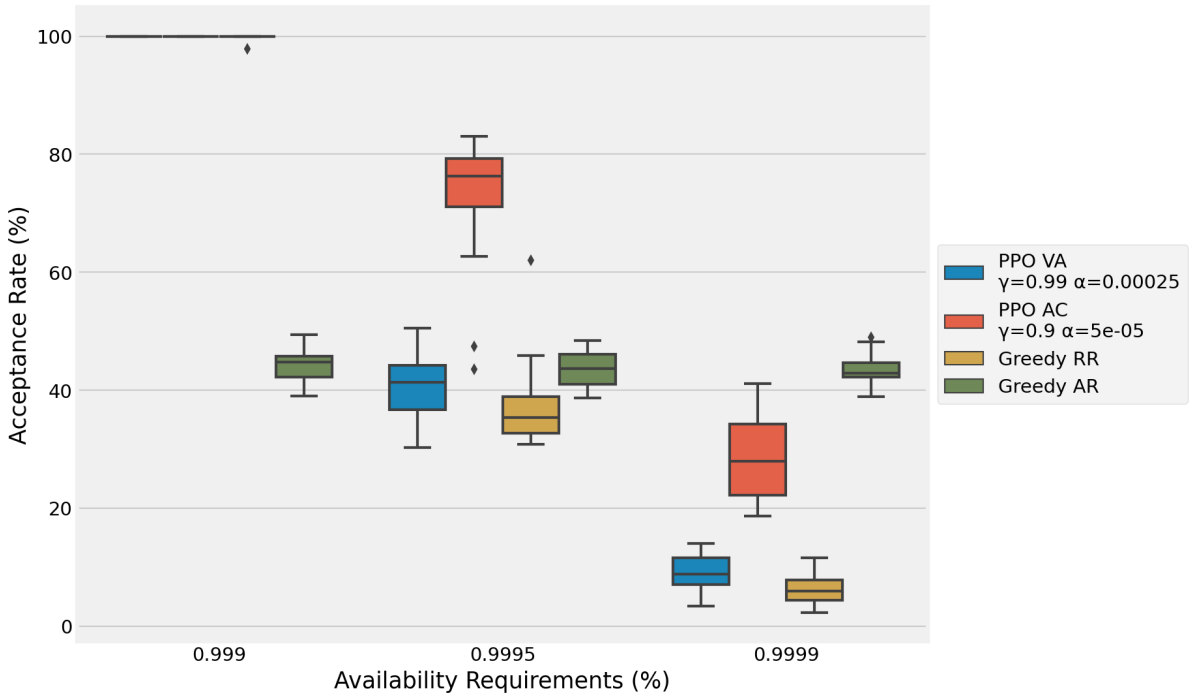
When we consider 55 customers sending SFC requests, the PPO AC performance decreases significantly, since the median acceptance rate only achieves 61.67%. However, PPO AC still has a better performance than other algorithms. PPO VA presents the second best performance, with a median acceptance rate of 40.72%, while the greedy algorithms present the worst results. The median acceptance rate of greedy AR and greedy RR are, respectively,

33.33% and 30.12%.

As the number of customers increases, the superiority of the Cand-RL algorithm using PPO increases in comparison to the two greedy algorithms. In addition, the PPO AC outperformed the other algorithms significantly in terms of acceptance rate. For instance, assuming 55 customers, the PPO AC outperformed the PPO VA, the greedy AR, and the greedy RR with a difference of 20.95%, 28.34%, and 31.55%, respectively. Therefore, the results suggest that PPO AC is the best solution to deal with an increasing number of customers in the system, with a superior performance than greedy solutions.

We also varied the availability requirement of the SFC requests using the basic simulation scenario. Figure 41 shows the results.

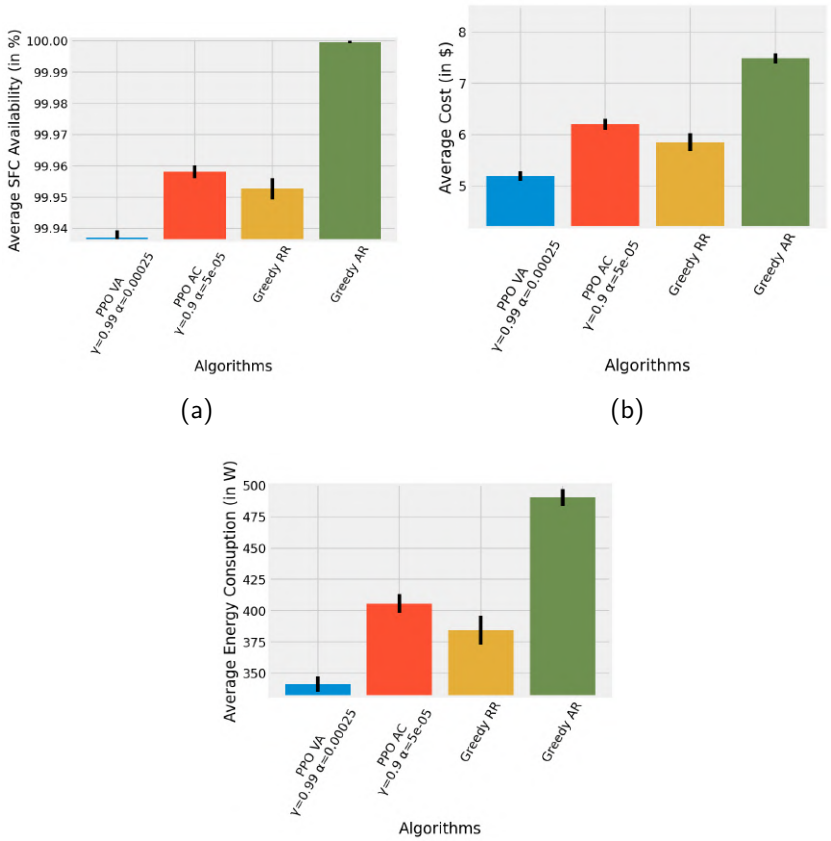
Figure 41 – Acceptance rate results for different availability requirements.



Source: the author (2023).

When we define an availability requirement of 99.9%, the PPO AC, the PPO VA, and the greedy RR obtains a 100% acceptance rate. This happens because it is a low availability requirement, which requires few redundant VNFs. On the other hand, the greedy AR presents a median acceptance rate of 44.74% since it adds a redundant VNF for all VNFs of the SFC request, consuming the nodes resources quickly. Furthermore, the performance of greedy algorithms are almost the same for the different availability requirements, since the redundancy strategy is always the same, regardless of the level of redundancy required.

Figure 42 – Comparison about SFC aspects.



Source: the author (2023).

When we increase the availability requirement to 99.95%, the algorithms (except the greedy AR) are impacted. This is because a high availability requirement requires more complex placement strategies and redundant VNFs that consume more resources. The PPO AC presents the best median acceptance rate, 76.26%, followed by the greedy AR, 43.66%. The PPO VA presents a median acceptance rate of 41.35%, and the greedy RR, which presents the worst result, presents a median of acceptance rate of 35.40%.

When we assume an availability requirement of 99.99%, all algorithms obtain a median acceptance rate below 40% with the exception of the greedy AR which achieves 42.86%. This was because the algorithms were unable to create placement strategies that met a higher availability requirement. Even adding redundancy in all VNFs (greedy AR), the acceptance rate is low. This underlines the difficulty of achieving a good acceptance rate in this scenario configuration. The PPO AC obtains a median acceptance rate of 27.93% followed by the PPO VA (8.84%) and greedy RR (5.96%).

We also compare different aspects of the SFC placed by the algorithms assuming the basic

simulation scenario. We first calculate the average of all SFCs allocated during simulation, then we calculate the average for 30 simulations. We also plot the standard deviation.

Figure 43(a) shows the average availability of the SFCs placed by the algorithms. It is important to highlight that the availability requirement of customers is 99.95% for the scenario considered. The PPO VA achieves the lowest average availability, 99.9371%, which explains the low acceptance rate illustrated in Figure 40 and 41. The greedy RR achieves an average availability of 99.9527%, which is the closest to the customer availability requirement. However, as shown in Figure 40, the greedy RR achieves a median acceptance rate below to 40%, which can be explained by the fact that many SFC requests still have an availability below the customer requirement, as the standard deviation shows. The PPO AC presents an average availability of 99.9581% with a small standard deviation. This result demonstrates the superiority of PPO over other approaches evaluated. It has a high acceptance rate (Figure 40) and competitive levels of availability. The greedy AR presents, as expected, the highest average availability, since it places the SFC with redundancy in all VNFs. Therefore, the greedy AR presents an average availability of 99.9996%. However, there is a trade-off between availability and SFC placement cost and energy consumption, as illustrated in Figures 43(b) and 43(c). The PPO VA, which presents the lowest average availability, also presents the lowest average cost ($5.19) and the lowest average energy consumption (341.43W). The greedy RR presents an average cost of $5.85 and an average energy consumption of 384.44W, while the PPO AC presents the average cost and average energy consumption of $6.20 and 405.78W, respectively. The greedy AR presents the highest cost and energy consumption: $7.49 and 490.48W, respectively, since it places a fully redundant SFC.

The trade-off must be considered by the network operator during SFC placement. While the PPO VA achieves low energy consumption and cost, the placed SFCs has a small average availability. On the other hand, the greedy AR is always able to place SFCs with high availability, but with high cost and energy consumption. The PPO AC and the greedy RR are able to allocate SFCs with a better balance between availability, cost, and energy consumption. Notwithstanding this, the acceptance rate must also be considered. For example, PPO AC presents a better acceptance rate than other algorithms evaluated in most of the scenarios considered, while at the same time it is able to achieve relatively good average availability and low cost and energy consumption. On the other hand, the greedy AR can achieve high levels of availability, but with low acceptance rates.

6.7 CONCLUDING REMARKS

In this chapter, we present a solution for SFC placement on large-scale networks with a focus on availability. We formulate the SFC placement problem as an MDP where the VNFs are processed sequentially and candidate nodes are selected to place them. We propose the Cand algorithm for selecting candidate nodes based on the computing and network requirements of the VNF, as well as the shortest path between the source and destination nodes of the SFC. The main goal of Cand algorithm is to reduce the amount of information from the physical infrastructure thereby reducing the input size of RL agent. We then combine the Cand algorithm with a RL agent to create the Cand-RL algorithm for defining the placement of SFC requests. The Cand-RL algorithm is based on an RL agent and, after training, is able to define the SFC placement that meets the customer availability requirements while minimizing cost and energy consumption. To create the RL agent, we used the PPO technique, which is the state of the art in actor-critic based techniques, surpassing other RL techniques in the literature. Simulation results show that the Cand-RL algorithm outperforms two different greedy approaches in terms of acceptance rate, and also presents the best balance between availability, energy consumption, and cost. In the next chapter, we present the prototype of SPIDER developed in this thesis. We show how we implemented the SPIDER using Kubernetes, as well as the example of an SFC was implemented using containers. We also present experiments to measure the placement time and the processing and communication delays of the SFCs placed.

7 SPIDER PROOF OF CONCEPT

In this chapter, we present a proof of concept of SPIDER framework. The main goal is to present how the SPIDER can be implemented in a real scenario and conduct experiments to assess the SPIDER performance. We detail how we use Kubernetes mechanisms to implement the VNFs. Afterwards, we present experiments in order to assess the placement time of SPIDER using a real SFC implemented using the Python language. We also carried out experiments to assess the communication and processing delay of VNFs under different network conditions (centralized and geographically distributed). The results obtained from this chapter were published in (SANTOS et al., 2022).

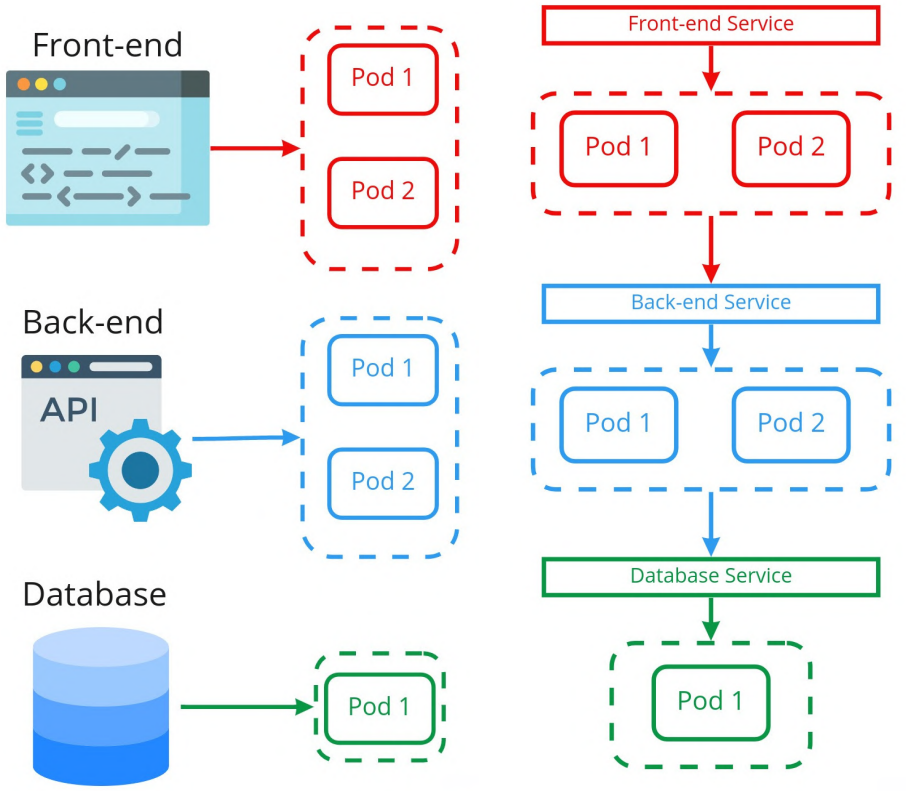
7.1 CONTAINER-BASED SFCS USING KUBERNETES

Under Kubernetes, applications (for example, web front-ends, back-end applications, APIs, databases, etc) are deployed as a group of containers, which is the smallest working unit named pod[1]. A pod is a group of one or more containers where they share the same storage and network resources, and runs over the same specification. For example, consider a web application composed of a front-end, a back-end, and a database, as shown in Figure 44(a). One can deploy each part of the application as one or more pods, where each pod consists of one or more Linux containers. In order to define the launching conditions of the pods, such as the number of replicas, we can define a deployment[2]. A deployment is responsible to define declarative updates for Pods. On can describe the desired state for a pod in a deployment, and the Deployment Controller changes the actual state to the desired state. For instance, we can define in the deployment of front-end that two pods need to be created, and these pods will consider the Docker image of the front-end. We can also define in which ports these containers listen for requests.

During the application runtime, Kubernetes could destroy and recreate the pods in order to meet the requirements of the deployment. However, the IP of these containers will be not the same after a recreation. In this context, considering that the front-end needs to access functions of the API implemented in the back-end, it is necessary to solve how the front-end finds out and keeps track of which IP address of the back-end. In order to mitigate this

[1] <https://kubernetes.io/docs/concepts/workloads/pods/>
[2] <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

Figure 43 – Basic Kubernetes concepts.



(a) Simple web application (b) Service example in Kubernetes

Source: adapted from (SANTOS et al., 2020b).

problem, Kubernetes provides an abstract way to access the pods named service[3]. It defines a logical set of pods and a policy by which to access them. A service exposes an application running on Kubernetes (in a pod, specified by a deployment) as a network service. Therefore, the user does not need to modify the application deployed in the Kubernetes to use a service discovery mechanism. Through the service, Kubernetes gives to the pods their own IP addresses and a DNS name, that allows them to be accessed. Usually, a label selector is defined for a deployment, and it can be used to access the associated pods. Figure 44(b) illustrates how the web application could be deployed as pods and exposed as different services. A front-end can call the API (running on Pod 1 or 2) through the back-end service, and Kubernetes will forward the request to the container of a pod.

Using the deployments to define how pods of an application will be launched and services to expose the pods, we can create container-based SFCs using the environment controller. For each VNF specified in JSON created by the Agent module, we create some files to launch the SFC.

[3] <https://kubernetes.io/docs/concepts/services-networking/service/>

The first file contains the application code for the implementation of the VNF itself. The implemented VNF examples were created using the Python language, but other programming languages could be used. In order to facilitate the development of VNFs, we implemented an auxiliary Python class that uses the Flask framework to provide the service to other VNFs. Hence, to implement a new VNF, a developer needs to create a class that extends our auxiliary Python class and implements an abstract method called *process_data*. This method receives as a parameter a *request* object from the Flask framework, which contains the data sent to the VNF to be processed. Therefore, the developer only needs to implement the data processing method (e.g. data inspection, data compression, etc.), while the auxiliary class handles receiving and forwarding the data from/to the next VNF.

The second file is a configuration (config) file that defines three information fields: the name of the next VNF service, a field that shows if the VNF is the last one of the SFC, and the port at which the VNF will listen for requests from other VNFs (as default we used port 5000). The auxiliary Python class (that the developer uses to implement a VNF) reads the information from the config file to setup the Kubernetes service to receive and send the data. When the SFC is created, the SPIDER builds automatically the VNF configuration file, defining which is the next VNF and the listening port (as default we used port 5000). Then, the auxiliary class (that the developer uses to implement a VNF) can listen for requests from other VNF and use the service mechanism of Kubernetes to access the next VNF of SFC.

The last file needed to create a VNFs is the Dockerfile that can be used to assemble a Docker image. The commands for defining the operating system, installing dependencies and starting the application that provides the VNF functionality are defined in the Dockerfile. A common approach is to create Docker images based on the Docker hub. In this case, we do not need to have a Dockerfile in our machine, since the Docker downloads the image from the Internet. However, we are assuming that the images of VNFs will be created from the Dockerfile and stored in the local Docker repository. For future implementations of SPIDER, we can consider that the Docker images for the VNFs could be downloaded from the Docker hub. All these files need to be located in the same path, which is defined in the VNF template, as described in Table 8.

Based on these files, the Environment Controller module creates the image of VNF in the local Docker repository and calls the Kubernetes API to create the deployment and the service of each VNF. The deployment and services can be defined following two different data modeling standards: YAML or JSON. We choose to use JSON since it is the main format

for exchanging information over the web (PEZOA et al., 2016), and we chose it to define the default communication data formatting among the other SPIDER modules.

An example of JSON to define the VNF deployment in Kubernetes is showed below:

```
1
2  {
3      "apiVersion": "apps/v1",
4      "kind": "Deployment",
5      "metadata":{
6          "name": "firewall",
7          "labels":{
8              "app":"firewall"
9          }
10     },
11     "spec": {
12         "replicas" : 2,
13         "selector": {
14             "matchLabels" : {
15                 "app":"firewall"
16             }
17         },
18         "template" : {
19             "metadata" : {
20                 "labels" : {
21                     "app":"firewall"
22                 }
23             },
24             "spec":{
25                 "containers":[
26                     {
27                         "name":"firewall",
28                         "image":"firewall:latest",
29                         "ports":[
```

```
                                {
                                    "containerPort": 5000
                                }
                            ],
                            "imagePullPolicy": "Never",
                            "limits":{
                                "cpu": 1,
                                "memory": 100Mib,
                                "storage": 10Gib
                            }
                        }
                    ],
                    "nodeSelector": {
                        "nodetype": "node_name"
                    }
                }
            }
        }
}
```

The example defines a VNF of a firewall. In addition to the information previously described, the *imagePullPolicy* is defined as *Never* because Kubernetes will create the container from the local repository. The *limits* field defines the amount of computing resources required by the VNF, and defined in its template. Finally the field *nodeSelector* defines the label of the physical node from the infrastructure where the VNF instance will be placed.

The JSON illustrated below is an example of service definition used to expose the firewall deployment:

```

{
    "apiVersion": "v1",
    "kind": "Service",
    "metadata":{
        "name": "firewall-service"
```

```
7       },
8       "spec":{
9           "selector":{
10              "app": "firewall-service"
11          },
12          "ports":[{"protocol": "TCP","port": 5000,"targetPort"
                : 5000}],
13          "type": "LoadBalancer"
14      }
15  }
```

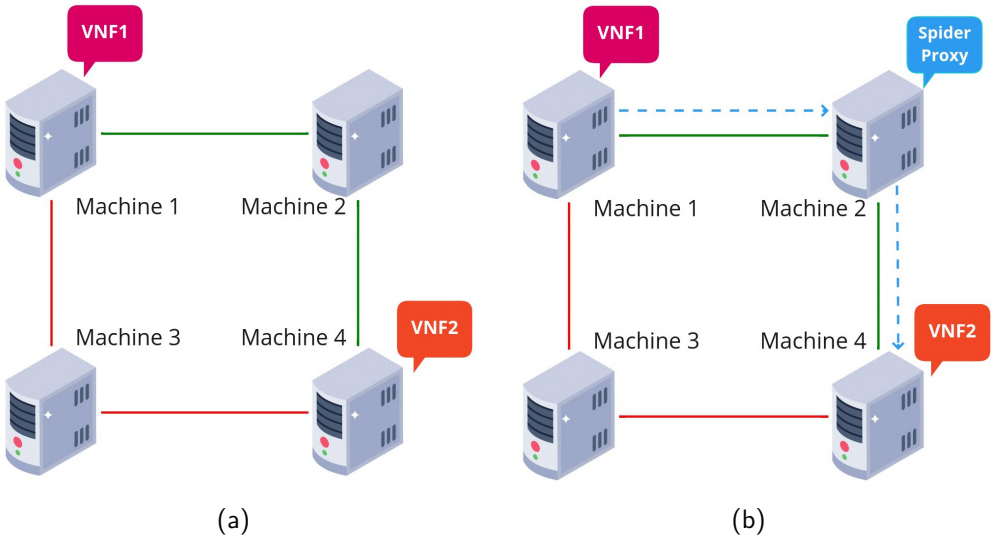
One can note that we only need to specify the name of the service (*firewall-service*), the protocol and ports where the service is listening for requests (TCP and 5000 respectively). Kubernetes allows defining different of services types. We used the *LoadBalancer* type, where Kubernetes tries to distribute network traffic across the pods equally, but other service types could also be used.

Therefore, when the Agent module sends the SFC placement to the Environment Controller module, the latter creates the Docker Image for each VNF and, based on the number of replicas and resources required, it creates a deployment and exposes it as a service. To access the SFC, it must send a request to the first VNF (using the service name, since it is exposed by service in Kubernetes) and, after processing data, the VNFs are then able to communicate with each other in order to provide the entire service.

In addition to the definition of VNFs, we need to define a mechanism to route the flow among them. For instance, consider the infrastructure shown in Figure 45(a), where the VNF1 is placed on Machine 1 and the VNF2 is placed on the Machine 4. We need to steer traffic between those machines, and decide how may one configure traffic to pass through the Machine 2 instead of Machine 3 for example.

By default, Kubernetes does not control traffic between the pods deployed in a cluster. To deal with this limitation, we implement the idea of place proxies (named Spider Proxy). They are placed on machines that compose the path between the machines on which the VNFs are placed. Figure 45(b) illustrates an example where a proxy is placed on Machine 2, composing the path between the VNF1 and VNF2. Therefore, instead the VNF1 sending data to the VNF2 directly, it sends data to Spider Proxy, which forwards to the VNF2. Spider Proxy does

Figure 44 – Traffic flow configuration.



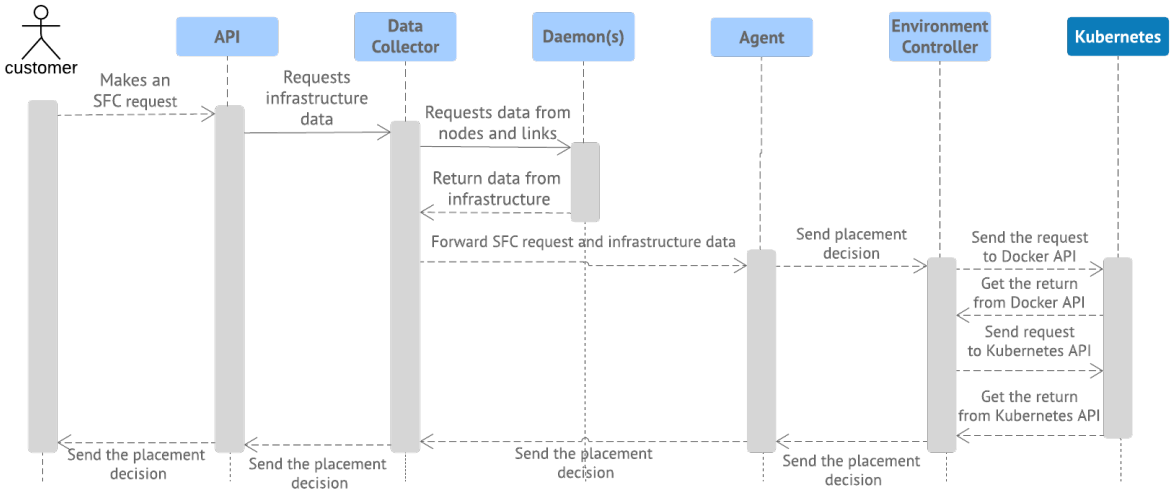
Source: the author (2023).

not perform any kind of processing, it just receives the data and forwards it to the defined destination. Please note that we can define many Spider Proxies in sequence, in order to define a path composed of many machines. In order to create the proxies, we use the virtual links defined by the Agent module. We deploy these proxies as a simple pod in Kubernetes, and expose them using services in order to be accessible by other VNFs of SFC.

Figure 45 shows the sequence diagram that summarizes the whole process adopted to make an SFC request to the SPIDER framework. The customer sends an SFC request to the SPIDER API, which forwards it to the Data Collector (first module of SPIDER core). The Data Collector gathers information about the infrastructure calling the Daemons that are running in the nodes. Afterwards, the Data Collector forwards both infrastructure data and the SFC request to the Agent module, which defines the SFC placement to ensure the availability required by the customer. Then, the Environment controller invokes the Docker API to create the docker image (if it does not exist in local repository) and next the Kubernetes API to create the deployment and the services for all VNFs.

In the next section, we will present the implementation of a proof-of-concept of SPIDER, as well as the results of its evaluation.

Figure 45 – SFC request sequence diagram.

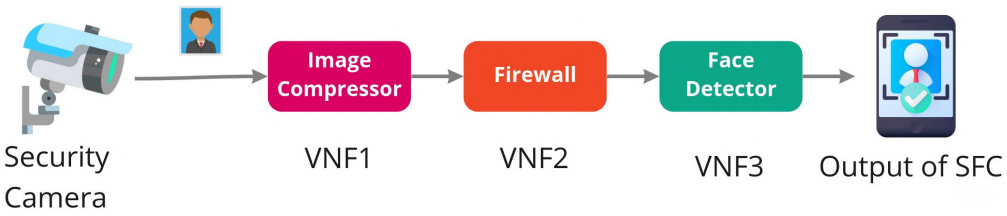


Source: the author (2023).

7.2 PROTOTYPE EVALUATION

In this section, we present a proof-of-concept of the SPIDER framework. We carried out experiments in order to assess the SPIDER performance and some aspects about the SFC placed. For the SFC example considered in the experiments, we consider a face detection service. Face detection systems can be applied in many contexts, such as gender classification, facial feature extraction, marketing, among others (KUMAR; KAUR; KUMAR, 2019). We design a simple system where a photograph must be sent to a server in order to check the number of faces to be detected within the image. This system can be considered in the context of smart home for access control or, more recently, in the context of COVID-19 pandemic, where it is important to control the number of people present in proximity in order to ensure social distancing.

Figure 46 – The face detection application represented as an SFC.



Source: the author (2023).

Figure 46 presents the application considered for the experiments. The first VNF is an image compressor, which reduces the original image dimension and resolution in order to

decrease the network consumption. In scenarios where the camera captures information in a high-resolution, the reduction of image dimension results in a smaller amount of image data, which in turn lowers the amount of bandwidth required. To implement the image compressor VNF, we use the OpenCV library[4]. This is a well-known and powerful library used for the analysis of images or videos. Next, the VNF receives the image, compresses and forwards it to the next VNF with a reduced dimension.

The second VNF is a firewall. We implemented a simple rule-based firewall, where the developer can define a set of rules to define if the traffic will be forwarded or excluded. For the sake of simplicity, we define three different rules to exclude traffic from the previous VNF: i) a list of prohibited IP addresses, ii) a list of denied ports, and iii) a list of excluded IP prefixes.

The last VNF is the face detector application itself. We also use the OpenCV library to implement the face detection application. The adopted algorithm to detect the faces in images is the Haar feature-based cascade classifier (VIOLA; JONES, 2001). The OpenCV function returns the boundary rectangles for the detected faces. Therefore, it is possible to know the quantity and the exact location of the faces in the image. Since the Haar feature-based cascade algorithm is a machine learning technique, we need to load the features generated by the algorithm training. To implement our face detection VNF, we download the XML file with the default features to detect the frontal face. In addition, OpenCV provides a set of different features to detect different objects in images[5], such as faces, eyes, full body, upper body, smile, and others.

7.2.1 Scenario Setup

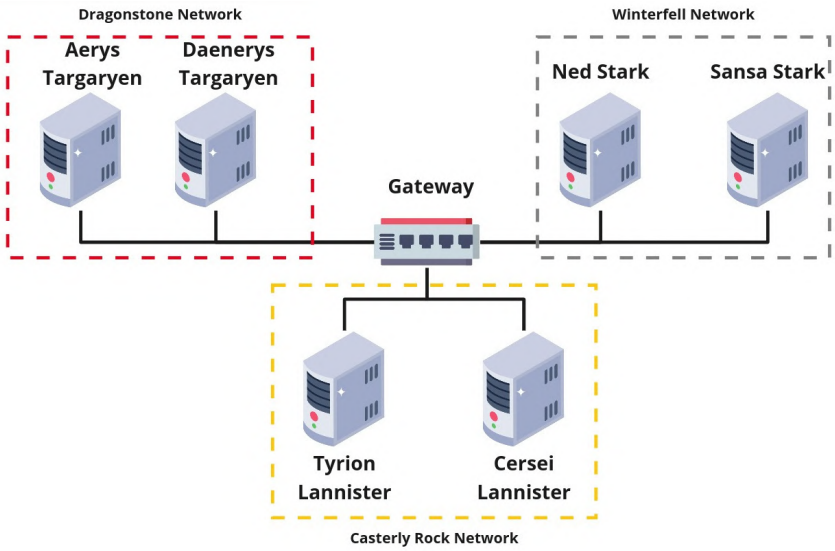
We create a network composed of several VMs using the VirtualBox hypervisor, since it is easy to replicate the machine configuration and change the network conditions. The VMs are running in a machine with an Intel(R) Core(TM) i7-7700 CPU processor with a 3.60GHz clock, 16 gigabytes of DD4 memory, and an SSD SATA with 480 gigabytes of storage space.

Figure 47 illustrates the network topology considered composed of six VMs. We considered that the machines are connected to a gateway which provides connectivity among them. Each VM has one CPU core, two gigabytes of memory, and a storage of 30 gigabytes. Each machine connects to the gateway through a link with 100 Mbps. We use the Ubuntu server 20.04 as

[4] <https://opencv.org/>
[5] <https://github.com/opencv/opencv/tree/master/data/haarcascades>

the operating system of all machines. Since SPIDER is based on Kubernetes and containers, in each VM we installed Docker version 20.10.7 and the Kubernetes version 23.3. Following the Kubernetes architecture, we need to define the main and worker nodes (SHAH; DUBARIA, 2019). The main node, which is responsible for the management of Kubernetes cluster (control plane), is the machine named Sansa Stark, while the remaining nodes are the worker nodes. As a requirement of Kubernetes, the node which runs the control plane must have at least two CPU cores. As a result, the Sansa Stark machine has two CPU cores, while the other ones have only one CPU core.

Figure 47 – Network topology considered for the experiments.



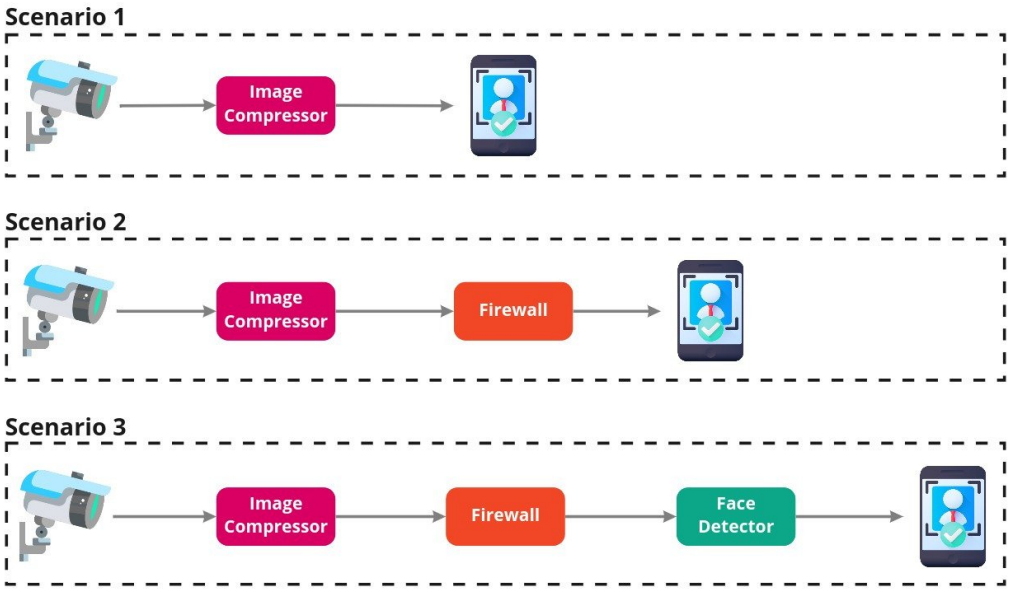
Source: the author (2023).

7.2.2 Placement Time

We evaluate the time to place an SFC in the infrastructure by measuring the time spent since sending a request to the API until the creation of the containers for all VNFs in the infrastructure. In order to assess the impact of SFC size on the placement time, we vary the number of VNFs from one to three, as shown in Figure 48. In Scenario 1, we consider only the compress image VNF, in the Scenario 2 we add the firewall VNF, and finally in Scenario 3, we also include the face detection VNF. We use images with width and height of 520x408 pixels, respectively. In our experiments, we reduce the original image to 224x224 pixels as part of its compression within the first VNF.

For these experiments, we establish the source and destination nodes randomly following

Figure 48 – Scenarios considered in the experiments.



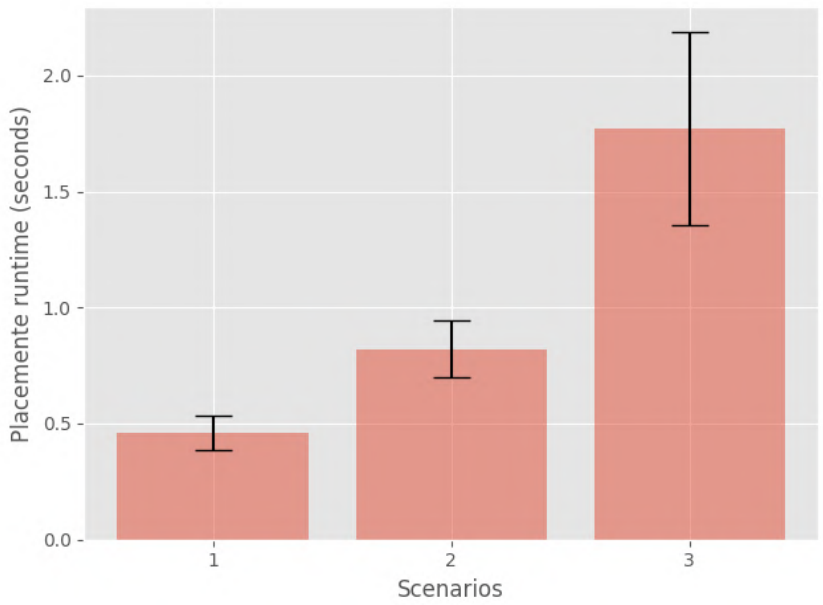
Source: the author (2023).

a uniform distribution. We defined an availability requirement of 99%. We also assume that all machines have containers images of all VNFs created in the local repository. This speeds up the downloading of their VNF packages and mitigates any Internet connectivity issues from affecting performance results. For each scenario, we call the API for the placement and calculate the runtime. Next, we invoke the API again to delete the SFC from the infrastructure, deleting the deployments, services, and pods. We repeat the process 30 times in order to calculate the average and standard deviation of SFC placement runtime. The runtime results are presented in Figure 49.

As expected, when we add VNFs in the SFC request, the average runtime increases. This happens because more containers need to be placed, leading to the creation of more image containers as well as having more containers launched. For Scenario 1, where only one VNF is considered, the average runtime is 0.3379 seconds. In scenario 2, with the addition of a second VNF, the average runtime increased to 0.6201 seconds, corresponding to an increase of 83.52% in comparison with scenario 1. In Scenario 3, with three VNFs, the achieved average runtime is 1.3877 seconds, showing a 123.79% increase when compared with Scenario 2, and a 310.68% increase when compared with Scenario 1.

One can also note that as we increase the number of VNFs in the scenario, the standard deviation increases. This happens because some containers may take longer to start than others, due to other operations that may be taking place at the time of allocation by the

Figure 49 – Placement runtime results.



Source: the author (2023).

operating system. Hence, when more containers need to be created, some can be created faster than others, which increases placement time variability. For the scenarios 1, 2, and 3 the standard deviation values of placement runtime are 0.1618, 0.2007, and 0.5673, respectively.

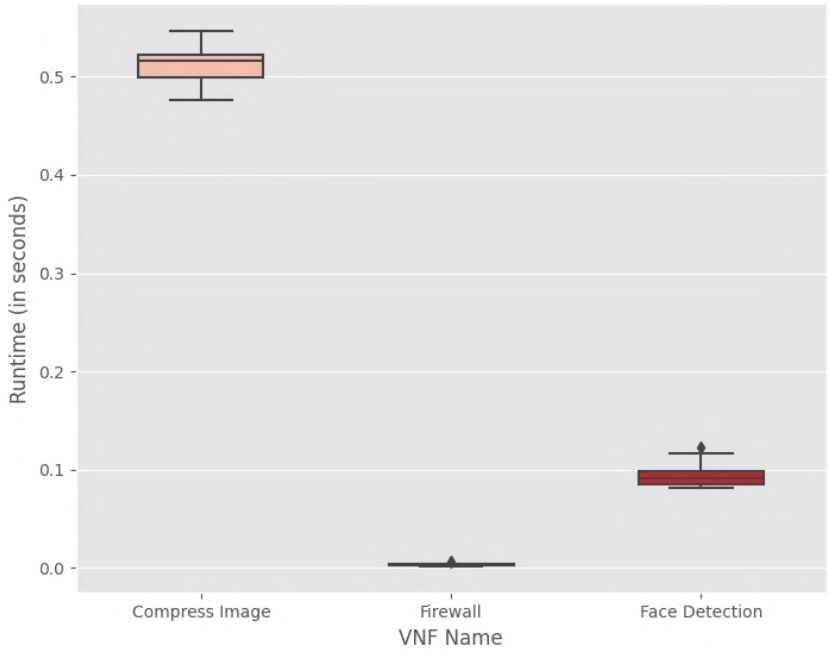
7.2.3 VNF processing Runtime

We also evaluate the processing runtime of the VNFs implemented in our prototype. The goal is to evaluate the performance of SFCs allocated by SPIDER using Kubernetes. Figure 50 shows the results. The VNFs that handles image processing presents a high runtime, i.e., the compress image and face detection functions. The compress image presents the highest runtime median, 0.5161 seconds, since this VNF processes the image in the original resolution. The face detection function processes the image in the reduced resolution, which results in a lower median of 0.0913 seconds delay.

The firewall function presents the lowest processing runtime, with a median of 0.0028 seconds. The firewall implementation is very simple, we only retrieve the IP and port from the data that arrives in VNF and check for the presence of a list of prohibitive IPs and port numbers. This justifies a lower processing runtime obtained when compared with others VNFs.

It is also important to highlight that the functions that deal with images present greater variability of the processing runtime. The compress image function has a standard deviation

Figure 50 – VNF processing runtime results.



Source: the author (2023).

of 0.0167, while the face detection function has a standard deviation of 0.0101. The firewall function has the smallest standard deviation among all VNFs: 0.0010.

7.2.4 Communication SFC Delay

We also evaluate the communication delay of the SFC for two different scenarios. In Scenario 1, all machines are in a local network, which means that the connection delay between the servers is too small. In Scenario 2, we use the Netem tool[6] to emulate a distributed network. Based on Figure 47, the different groups of servers are located in different cities, as illustrated in Table 19. The different delays are based on (SANTOS et al., 2018) and we emulate the delay on each server's link using the Netem tool. For these experiments, we define as source and destination nodes the servers Tyrion Lannister and Ned Stark, respectively. We selected these machines because they fall in the group of nodes with the highest delays. We consider the complete SFC illustrated in Figure 46 composed of the three VNFs, i.e., compress image, firewall, and the face detection application.

[6] <https://wiki.linuxfoundation.org/networking/netem>

Table 19 – Network configuration of servers in Scenario 2.

Servers	Location	Delay (in ms)
Daenerys Targaryen	São Paulo (Brazil)	179
Aerys Targaryen		
Ned Stark	London (England)	427
Sansa Stark		
Tyrion Lannister	Tokyo (Japan)	567
Cersei Lannister		

Source: the author (2023).

Table 20 shows the results for the Scenarios 1 and 2. In Scenario 1, we can see that the communication SFC delay remains very small, below one second. The median of communication SFC delay is 0.0455 second, while the minimum and maximum values are 0.0359 and 0.0617 seconds, respectively. It is important to highlight that the standard deviation of communication SFC delay is also very small: 0.0073, which means that all SFCs allocated by the framework during the experiment had a very low delay.

Table 20 – Communication SFC delay results (in seconds).

Scenario	Min	Q1	Median	Q3	Max	Standard deviation
1	0.035918	0.040332	0.045478	0.052216	0.061684	0.007283
2	2.651789	2.681213	2.689708	2.705447	2.743413	0.019979

Source: the author (2023).

On the other hand, the communication SFC delay increased considerably in the distributed scenario. The median of communication SFC delay of Scenario 2 increases to 2.6897 seconds. This surge in the communication SFC delay can be explained by two main factors. The first one is due to having a higher delay in the links that connect the servers in the network. As a result, when a VNF sends the data to another one, it will take more time to arrive at the destination VNF. The second reason is due to the communication overhead between the network machines suffered to gather information about the VNFs. As mentioned in Subsection 7.1, after processing data, a VNF needs to forward it to the next VNF, and we use the Service resource provided by the Kubernetes to implement this feature. However, a VNF needs to discover the Service IP of the next VNF in order to forward data, and this discovery is achieved by communicating with the main Kubernetes node. Therefore, this new request to the main node increases the communication delay of SFC, which justifies the delay of approximately

two seconds, even though the longest link delay is just over 500ms.

Similar to the Scenario 1, Scenario 2 presents a lower variability in the communication SFC delay. The minimum and maximum delays are 2.6518 and 2.7434 seconds, respectively. The standard deviation is 0.0110, which shows that the framework is able to allocate SFCs with low delay variability.

7.2.5 Overall SFC delay

Table 21 shows the results of overall SFC delay, which is the sum of communication delay and the processing runtime of VNFs for the Scenarios 1 and 2. We also calculate the percentage of communication delay and VNFs processing delay from the overall SFC delay, in order to assess their impact.

Table 21 – Overall SFC delay results.

Scenario	Overall SFC delay (in seconds)	Impact of communication delay (in %)	Impact of VNFs processing delay (in %)
1	0.655255	7.062167	92.937833
2	3.300723	81.550097	18.449902

Source: the author (2023).

For the Scenario 1, the overall SFC delay is less than one second, or more exactly 0.6553 seconds. However, 92.94% of the delay in this scenario is due to the VNFs processing delay, while the communication delay represents only 7.06%. This behaviour is expected since the communication delay is very small in this scenario, due to the fact that the machines are located in the same local network, as explained in Subsection 7.2.4. On the other hand, in Scenario 2, we experience a different behaviour. The overall SFC delay is 3.3007 seconds, and the communication delay is responsible for as much as 81.55% of the overall delay, while the VNFs processing delay represents only 18.45% of the overall delay. Therefore, depending on network scenario, i.e., whether or not we have a distributed network, the communication delay may cause an expressive impact on the overall SFC delay. Nonetheless, for the scenario where the delay between the links is small, the VNFs processing delay can represent the biggest share of the total delay of the SFC.

7.3 CONCLUDING REMARKS

In this chapter, we implemented a proof-of-concept of SPIDER in order to conduct experiments considering a scenario where the SFC is a face detection application. Our results showed that SPIDER is able to place SFCs within around one second, and that this time is impacted by the number of VNFs that compose the requested SFC. We also evaluate the VNFs processing delay, communication delay, and overall SFC delay in a local network and in an emulated geographical distributed network. Our results showed that the communication and VNFs processing delays have different impact on the overall SFC delay depending on the network types (geographically distributed or centralized).

8 GENERAL CONSIDERATIONS

The virtualisation paradigm makes possible the sharing of computational resources, allowing to allocate different VMs at the same commodity hardware. This paradigm improves the hardware utilisation, flexibility management, can reduce the energy consumption, and increases the scalability (SAHOO; MOHAPATRA; LATH, 2010). Since the network managers suffer similar issues with network management, such as deploying service functions at proprietary expensive middle-boxes (MIRJALILY; ZHIQUAN, 2018), the NFV paradigm emerges as a solution, bringing virtualisation for the network environment. NFV and SDN allow to allocate SFCs in commercial-off-the-shelf devices, improving the network performance, security, as well as reducing the CAPEX and OPEX (PEI et al., 2018).

Besides all benefits of NFV paradigm, it is not without challenges. Considering distributed scenarios, where may there exists a high variety of hardware and software, the SFC placement is a challenge (MECHTRI et al., 2017). The SFC availability is an aspect that deserves attention by the network manager with the purpose to guarantee the downtime avoid and failure recovery (SOUALAH et al., 2017).

Against this backdrop, this thesis presents SPIDER, which is a framework for SFC placement in distributed scenarios with focus on optimise the availability. The SPIDER operation is based on infrastructure context information and ML algorithms in order to make the SFC placement with minimum human intervention as possible. SPIDER collects and stores context information from the infrastructure in order to choose the best solution of placement for an SFC request. The automation is based on DL models and RL agents that are used to map the VNFs and virtual links into the physical nodes and links of the infrastructure.

We presented the SPIDER architecture and its modules. We show the data modules used by SPIDER to store the context information about the physical infrastructure. Then, we present the SPIDER core, which contains the main modules of the framework. We describe how SPIDER can collect information about the nodes and use it to create the SFC placement in order to meet the customer requirements (specially the SFC availability).

The SFC placement is done by the agent module, the most important module of SPIDER. It is based on ML algorithms. We proposed DL models combined with clustering algorithms for the traffic prediction, which is used to predict traffic from the SFCs already placed in the infrastructure. Our results showed that the DL models outperformed traditional ML algorithms,

obtaining lower prediction error. Using the context information and the traffic prediction, the agent module uses RL to define the best SFC placement. We proposed the Cand-RL algorithm, that selects candidate nodes based on the computing requirements of the VNFs and the shortest path between the source and destination nodes of the SFC. Based on the candidate nodes, Cand-RL algorithm uses RL to define the best node and the redundancy strategy. Our studies showed that the Cand-RL (empowered by a PPO agent) outperformed greedy solutions in terms of acceptance rate, while provides best balance between availability, energy consumption, and cost.

We also presented a proof-of-concept of SPIDER using the Kubernetes. We showed how to use Kubernetes mechanisms to deploy the VNFs as containers and forward the traffic among them. We carried out experiments to evaluate the placement time of SPIDER considering a real network, composed of virtualized nodes and a real SFC implemented using the Python language. In addition, in order to demonstrate how the SFC allocated by the SPIDER using Kubernetes are functional, we evaluated the runtime and the communication delay considering a local network and a distributed network.

Following, we show how we answer the research questions considered in this thesis that are presented in Chapter 1:

- **How to allocate SFCs in distributed scenarios efficiently?** The answer of this question is presented in Chapter 3, where we present the main frameworks found in the literature for SFC placement. The related works were the result of a systematic review published in (SANTOS et al., 2022). We discussed briefly the frameworks and the algorithms that they used for the SFC placement. Also, we compared the solutions with the SPIDER, and how our proposal differs from the others.

- **How can contextual information about the infrastructure be organised and stored to enable quick consultation and updating?** In Chapter 4, specifically in Section 4.3.1, we present the data models and repositories that we used to represent the infrastructure (nodes and links), VNF, and SFC information. We show how the information are modelled in the SPIDER, describing each of the fields and data types.

- **How can computational algorithms be used to provide intelligent placement strategies, with a focus on SFC availability?** In Chapters 5 and 6 we present the ML algorithms for traffic prediction and SFC placement, respectively. These algorithms are

used to implement the Agent module of SPIDER (see Figure 14). In Chapter 5, we carried out experiments comparing two different DL techniques for traffic prediction, LSTM and GRU. The goal is to evaluate the technique that provides the lowest error prediction. In Chapter 6, we present the Cand-RL algorithm, which uses an agent implemented with the PPO algorithm for the SFC placement. The Cand-RL algorithm considers the availability requirement as well as the placement cost and energy consumption during the SFC placement decision.

- **How effective can an SFC availability-focused solution be?** In order to validate the different modules of SPIDER, we carried out several experiments considering real data, simulated scenario and real scenarios as shown in Chapters 5, 6, and 7.

8.1 LIMITATIONS

Although we have conducted studies to show that SPIDER can be implemented in real networks, and the proposed modules have results that surpass traditional algorithms in the literature, it has some limitations.

The first limitation is about the traffic prediction module. We use a specific data set on mobile networks, which has particular characteristics and can differ from traffic from other types of networks. Thus, depending on the type of SFC that will be allocated to the infrastructure and the respective traffic it will receive, the models may need to be retrained. However, this update can be done on a rolling basis in the model, as the traffic prediction module is exposed an API to the SPIDER core. Another limitation about the prediction module is that we assume that historical information about the traffic of each placed SFC is stored in the database and accessible through the repository.

Like the prediction module, the placement planner module, which is based on RL, needs to be retrained when network conditions change dramatically. This can happen in cases where new nodes are added to the network, and they have very different resources from the previous ones. However, like the traffic prediction models, the RL agent training can be done offline and replaced only by making the newly trained agent available through the placement planner module API.

Another limitation is about the SPIDER prototyping using the Kubernetes tool. Although Kubernetes is currently the most used tool for orchestrating containers, by default it is very

limited in handling network traffic. For the implementation of SFCs, we needed mechanisms for handling and discovering containers (in order to chain VNFs) and also mechanisms to direct traffic between nodes and physical links in the infrastructure. Although Kubernetes provides mechanisms for handling and discovering containers, it does not natively provide tools for directing traffic. For that, we had to implement the proxy which is a simple network function for directing the traffic.

8.2 SCIENTIFIC CONTRIBUTIONS

Table 22 presents the scientific papers produced in scope of this thesis, including papers published and submitted.

8.3 FUTURE WORKS

Many future works can be considered to improve the contribution of this thesis. We plan to integrate SPIDER with existing open source monitoring tools such as Zabbix[1] and Prometheus[2]. These tools allow the mapping and monitoring of the entire infrastructure, collecting and providing data about the devices. The purpose of such integration is due to the popularity of such tools, which would facilitate the implementation of SPIDER in different environments without major changes in the software solutions currently used by network managers. In addition, while we integrated SPIDER with Kubernetes in our proof of concept, we plan to integrate it with other management tools. Open Source MANO[3] and Tacker[4] are tools that are based on traditional virtual machines to allocate VNFs, which can be considered in future SPIDER implementations. Another important module to be developed in SPIDER is the one that monitors the traffic of each SFC to store the history in each database. As mentioned earlier, historical traffic information is important for the prediction module.

Taking into account the traffic prediction models, used in the Traffic Prediction module of SPIDER, we plan to compare additional machine learning and deep learning architectures including ensemble approaches and augmenting the model with longer-term historical trend data. Furthermore, we will extend the dataset with more heterogeneous data sources including

[1] <https://www.zabbix.com/>
[2] <https://prometheus.io/>
[3] <https://osm.etsi.org/>
[4] <https://wiki.openstack.org/wiki/Tacker>

SMS and voice call log data, amongst others, as well as other areas, e.g. Trentino, available in the Telecom Italia dataset. Improved mobile network prediction can be applied to a wide range of network planning and optimization use cases to optimise utilization, reduce cost and meet QoS. In future works, we will explore the efficacy of these models in a variety of use cases particularly where the faster training times of GRUs may provide advantages over LSTM, such as multi-step prediction and faster optimization time scales.

In order to improve the Placement Planner module of SPIDER, we plan to use different machine learning algorithms, including other deep model-free RL algorithms, in the agent module and compare these with the Cand-RL. We can also increase the system model complexity to consider more requirements about the flow entries. In this paper, we consider only the bandwidth requirements, but we can also define delay and cost requirements, which demand a different solution to select the candidate nodes to place the VNFs, considering only physical links that meet all requirements.

We also plan to extend the SPIDER to address multi-domain scenarios. Even we consider the SPIDER is able to deal with distributed infrastructure, we assume that all devices are managed by the same manager. However, many use cases consider that many networks are managed by different managers, such as Industrial IoT and autonomous vehicles (TOUMI et al., 2021). In this way, we plan to consider the challenges of these scenarios, such as forward traffic across different networks and security issues.

Table 22 – Scientific papers produced related to this thesis.

#	Reference	Type	Status	Qualis
1	**Santos, G. L.**, Bezerra, D., Rocha, E., Ferreira, L., Gonçalves, G., Moreira, A., ... and Endo, P. T. (2020, October). Analyzing the Impact of Micro Data Centers Failures in Cellular Networks: a Road Race Study. In 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 3096-3102). IEEE.	Conference	Published	A3
2	**Santos, G. L.**, Kelner, J., Sadok, D., and Endo, P. T. (2020, November). Using Reinforcement Learning to Allocate and Manage SFC in Cellular Networks. In 2020 16th International Conference on Network and Service Management (CNSM) (pp. 1-5). IEEE.	Conference	Published	A3
3	**Santos, G. L.**, Endo, P. T., Lynn, T., Sadok, D., Kelner, J. (2021, September). Automating the service function chain availability assessment. In 2021 IEEE Symposium on Computers and Communications (ISCC) (pp. 1-7). IEEE.	Conference	Published	A3
4	**Santos, G. L.**, Endo, P. T., Sadok, D., and Kelner, J. (2020). When 5G meets deep learning: a systematic review. Algorithms, 13(9), 208.	Journal	Published	A4
5	Bezerra, D. de F., **Santos, G. L.**, Gonçalves, G., Moreira, A.,da Silva L. G. F, Rocha E. S., Marquezini, M. V., Kelner, J., Sadok, D., Mehta A., Wildeman, M., and Endo, P. T. (2021). Optimizing NFV placement for distributing micro-data centers in cellular networks. Journal of Supercomputing.	Journal	Published	A2
6	**Santos, G. L.**, Lynn, T., Kelner, J., Endo, P. T. (2021). Availability-aware and energy-aware dynamic SFC placement using reinforcement learning. The Journal of Supercomputing, 77(11), 12711-12740.	Journal	Published	A2
7	**Santos, G. L.**, Rosati, P., Lynn, T., Kelner, J., Sadok, D., and Endo, P. T. (2022). Predicting Short-term Mobile Internet Traffic from Internet Activity using Recurrent Neural Networks. International Journal of Network Management.	Journal	Published	A4
8	**Santos, G. L.**, Bezerra, D. de F., Rocha. E., da Silva L. G. F, Moreira, A., Gonçalves, G., Marquezini, M. V., Mehta A., Kelner, J., Sadok, D., and Endo, P. T. (2022). Service Function Chain placement in distributed scenarios: a systematic review. Journal of Network and Systems Management.	Journal	Published	A3
9	**Santos, G. L.**, Endo, P. T., Lynn, T., Sadok, D., Kelner, J. (2022). A reinforcement learning-based approach for availability-aware service function chain placement in large-scale networks. Future Generation Computer Systems.	Journal	Published	A1
10	**Santos, G. L.**, Endo, P. T., Sadok, D., Kelner, J. (2022). SPIDER: A Framework for SFC Placement in Distributed Scenarios with Focus on Availability. Journal of Software Practice and Experience.	Journal	Published	A3

Source: the author (2023).

REFERENCES

ABDULJABBAR, R. L.; DIA, H.; TSAI, P.-W. Development and evaluation of bidirectional lstm freeway traffic forecasting models using simulation data. *Scientific reports*, Springer, v. 11, n. 1, p. 1–16, 2021.

ABOWD, G. D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a better understanding of context and context-awareness. In: SPRINGER. *International symposium on handheld and ubiquitous computing*. [S.l.], 1999. p. 304–307.

ABU-LEBDEH, M.; NABOULSI, D.; GLITHO, R.; TCHOUATI, C. W. On the placement of vnf managers in large-scale and distributed nfv systems. *IEEE Transactions on Network and Service Management*, IEEE, v. 14, n. 4, p. 875–889, 2017.

ALAMDARI, A. B.; SHARIFI, M. Solving a joint availability-redundancy optimization model with multistate components and metaheuristic approach. *International Journal of Industrial Mathematics*, v. 12, n. 1, p. 59–70, 2020.

ALAMEDDINE, H. A.; ASSI, C.; TUSHAR, M. H. K.; YU, J. Y. Low-latency service schedule orchestration in nfv-based networks. In: IEEE. *2019 IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2019. p. 378–386.

ALI, H. M. M.; LAWEY, A. Q.; EL-GORASHI, T. E.; ELMIRGHANI, J. M. Energy efficient disaggregated servers for future data centers. In: IEEE. *2015 20th European Conference on Networks and Optical Communications-(NOC)*. [S.l.], 2015. p. 1–6.

ALMURSHED, O.; RANA, O.; CHARD, K. Greedy nominator heuristic: Virtual function placement on fog resources. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 34, n. 6, p. e6765, 2022.

AMIN, F.; CHOI, G. S. Hotspots analysis using cyber-physical-social system for a smart city. *IEEE Access*, IEEE, v. 8, p. 122197–122209, 2020.

ANDRADE, E.; NOGUEIRA, B.; MATOS, R.; CALLOU, G.; MACIEL, P. Availability modeling and analysis of a disaster-recovery-as-a-service solution. *Computing*, Springer, v. 99, n. 10, p. 929–954, 2017.

ANSARI, M.; ALSAMHI, S.; QIAO, Y.; YE, Y.; LEE, B. Security of distributed intelligence in edge computing: Threats and countermeasures. In: LYNN JOHN MOONEY, P. E. T.; LEE, B. (Ed.). *The Cloud-to-Thing Continuum - Opportunities and Challenges in Cloud, Fog and Edge Computing*. Cham, Switzerland: Palgrave Macmillan, 2020. chap. 6.

ARAÚJO, I. M.; NATALINO, C.; CHEN, H.; ANDRADE, M. D.; CARDOSO, D. L.; MONTI, P. Availability-guaranteed service function chain provisioning with optional shared backups. In: IEEE. *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*. [S.l.], 2020. p. 1–6.

ARAUJO, J.; MACIEL, P.; ANDRADE, E.; CALLOU, G.; ALVES, V.; CUNHA, P. Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models. *Journal of Cloud Computing*, Springer, v. 7, n. 1, p. 7, 2018.

ARAUJO, J.; MACIEL, P.; TORQUATO, M.; CALLOU, G.; ANDRADE, E. Availability evaluation of digital library cloud services. In: IEEE. *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. [S.l.], 2014. p. 666–671.

ARORA, P.; VARSHNEY, S. et al. Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, Elsevier, v. 78, p. 507–512, 2016.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

ASLANPOUR, M. S.; GHOBAEI-ARANI, M.; TOOSI, A. N. Auto-scaling web applications in clouds: A cost-aware approach. *Journal of Network and Computer Applications*, Elsevier, v. 95, p. 26–41, 2017.

ATTAR, A.; RAISSI, S.; KHALILI-DAMGHANI, K. A simulation-based optimization approach for free distributed repairable multi-state availability-redundancy allocation problems. *Reliability Engineering & System Safety*, Elsevier, v. 157, p. 177–191, 2017.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004.

AZARI, A.; PAPAPETROU, P.; DENIC, S.; PETERS, G. Cellular traffic prediction and classification: A comparative evaluation of lstm and arima. In: SPRINGER. *Discovery Science: 22nd International Conference, DS 2019, Split, Croatia, October 28–30, 2019, Proceedings 22*. [S.l.], 2019. p. 129–144.

BARLACCHI, G.; NADAI, M. D.; LARCHER, R.; CASELLA, A.; CHITIC, C.; TORRISI, G.; ANTONELLI, F.; VESPIGNANI, A.; PENTLAND, A.; LEPRI, B. A multi-source dataset of urban life in the city of milan and the province of trentino. *Scientific data*, Nature Publishing Group, v. 2, p. 150055, 2015.

BAUSE, F.; KRITZINGER, P. S. *Stochastic petri nets*. [S.l.]: Citeseer, 2002.

BHAMARE, D.; ERBAD, A.; JAIN, R.; ZOLANVARI, M.; SAMAKA, M. Efficient virtual network function placement strategies for cloud radio access networks. *Computer Communications*, Elsevier, v. 127, p. 50–60, 2018.

BHAMARE, D.; JAIN, R.; SAMAKA, M.; ERBAD, A. A survey on service function chaining. *Journal of Network and Computer Applications*, Elsevier, v. 75, p. 138–155, 2016.

BHAMARE, D.; SAMAKA, M.; ERBAD, A.; JAIN, R.; GUPTA, L.; CHAN, H. A. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications*, Elsevier, v. 102, p. 1–16, 2017.

BINZ, T.; BREITENBÜCHER, U.; KOPP, O.; LEYMANN, F. Tosca: portable automated deployment and management of cloud applications. In: *Advanced Web Services*. [S.l.]: Springer, 2014. p. 527–549.

BOLCH, G.; GREINER, S.; MEER, H. D.; TRIVEDI, K. S. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. [S.l.]: John Wiley & Sons, 2006.

BOUBENDIR, A.; BERTIN, E.; DIAZ, G.; SIMONI, N. Flexible and dynamic network-as-a-service for next generation internet. *Network as a Service for Next Generation Internet*, IET, v. 73, p. 51, 2017.

BOUTABA, R.; SALAHUDDIN, M. A.; LIMAM, N.; AYOUBI, S.; SHAHRIAR, N.; ESTRADA-SOLANO, F.; CAICEDO, O. M. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, Springer, v. 9, n. 1, p. 16, 2018.

CAI, J.; HUANG, Z.; LUO, J.; LIU, Y.; ZHAO, H.; LIAO, L. Composing and deploying parallelized service function chains. *Journal of Network and Computer Applications*, Elsevier, p. 102637, 2020.

CAO, J.; LI, Z.; LI, J. Financial time series forecasting model based on ceemdan and lstm. *Physica A: Statistical mechanics and its applications*, Elsevier, v. 519, p. 127–139, 2019.

CAO, S.; LIU, W. Lstm network based traffic flow prediction for cellular networks. In: SPRINGER. *International Conference on Simulation Tools and Techniques*. [S.l.], 2019. p. 643–653.

CARDOSO, J.; BARROS, A.; MAY, N.; KYLAU, U. Towards a unified service description language for the internet of services: Requirements and first developments. In: IEEE. *2010 IEEE International Conference on Services Computing*. [S.l.], 2010. p. 602–609.

CARPIO, F.; BZIUK, W.; JUKAN, A. Replication of virtual network functions: Optimizing link utilization and resource costs. In: IEEE. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.], 2017. p. 521–526.

CAUX, M. de; BERNARDINI, F.; VITERBO, J. Short-term forecasting in bitcoin time series using lstm and gru rnns. In: SBC. *Anais do VIII Symposium on Knowledge Discovery, Mining and Learning*. [S.l.], 2020. p. 97–104.

CHAI, H.; ZHANG, J.; WANG, Z.; SHI, J.; HUANG, T. A parallel placement approach for service function chain using deep reinforcement learning. In: IEEE. *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. [S.l.], 2019. p. 2123–2128.

CHAI, T.; DRAXLER, R. R. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, Copernicus GmbH, v. 7, n. 3, p. 1247–1250, 2014.

CHEN, H.; WANG, X.; ZHAO, Y.; SONG, T.; WANG, Y.; XU, S.; LI, L. Mosc: A method to assign the outsourcing of service function chain across multiple clouds. *Computer Networks*, Elsevier, v. 133, p. 166–182, 2018.

CHEN, J.; CHENG, X.; CHEN, J.; ZHANG, H. A lightweight sfc embedding framework in sdn/nfv-enabled wireless network based on reinforcement learning. *IEEE Systems Journal*, IEEE, 2021.

CHEN, L.; HA, W. Reliability prediction and qos selection for web service composition. *International Journal of Computational Science and Engineering*, Inderscience Publishers (IEL), v. 16, n. 2, p. 202–211, 2018.

CHEN, L.; YANG, D.; ZHANG, D.; WANG, C.; LI, J. et al. Deep mobile traffic forecast and complementary base station clustering for c-ran optimization. *Journal of Network and Computer Applications*, Elsevier, v. 121, p. 59–69, 2018.

CHEN, X.; LI, Z.; ZHANG, Y.; LONG, R.; YU, H.; DU, X.; GUIZANI, M. Reinforcement learning–based qos/qoe-aware service function chaining in software-driven 5g slices. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 29, n. 11, p. e3477, 2018.

CHEN, Y.-T.; LIAO, W. Mobility-aware service function chaining in 5g wireless networks with mobile edge computing. In: IEEE. *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. [S.l.], 2019. p. 1–6.

CHNITI, G.; BAKIR, H.; ZAHER, H. E-commerce time series forecasting using lstm neural network and support vector regression. In: *Proceedings of the international conference on big data and Internet of Thing*. [S.l.: s.n.], 2017. p. 80–84.

CHUNG, J.; GULCEHRE, C.; CHO, K.; BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

CHUNG, J.; GULCEHRE, C.; CHO, K.; BENGIO, Y. Gated feedback recurrent neural networks. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015. p. 2067–2075.

COMPUTING, A. et al. An architectural blueprint for autonomic computing. *IBM White Paper*, Citeseer, v. 31, n. 2006, p. 1–6, 2005.

COSTA, I.; ARAUJO, J.; DANTAS, J.; CAMPOS, E.; SILVA, F. A.; MACIEL, P. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, Wiley Online Library, v. 32, n. 7, p. 2191–2205, 2016.

DÂMASO, A.; ROSA, N.; MACIEL, P. Reliability of wireless sensor networks. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 9, p. 15760–15785, 2014.

DÂMASO, A.; ROSA, N.; MACIEL, P. Integrated evaluation of reliability and power consumption of wireless sensor networks. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 11, p. 2547, 2017.

DEEZER. *Frases dos Racionais MC's: conheça as letras mais famosas do grupo*. 2023. Https://www.deezer-blog.com/br/frases-racionais/. Accessed in: 12 may 2023.

DU, W.; DING, S. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. *Artificial Intelligence Review*, Springer, v. 54, n. 5, p. 3215–3238, 2021.

ELLIOTT, A. C.; HYNAN, L. S. A sas® macro implementation of a multiple comparison post hoc test for a kruskal–wallis analysis. *Computer methods and programs in biomedicine*, Elsevier, v. 102, n. 1, p. 75–80, 2011.

ELMOKASHFI, A.; KVALBEIN, A.; DOVROLIS, C. On the scalability of bgp: The role of topology growth. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 28, n. 8, p. 1250–1261, 2010.

ENDO, P. T.; SANTOS, G. L.; ROSENDO, D.; GOMES, D. M.; MOREIRA, A.; KELNER, J.; SADOK, D.; GONÇALVES, G. E.; MAHLOO, M. Minimizing and managing cloud failures. *Computer*, IEEE, v. 50, n. 11, p. 86–90, 2017.

ERAMO, V.; AMMAR, M.; LAVACCA, F. G. Migration energy aware reconfigurations of virtual network function instances in nfv architectures. *IEEE Access*, IEEE, v. 5, p. 4927–4938, 2017.

ERAMO, V.; MIUCCI, E.; AMMAR, M.; LAVACCA, F. G. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking*, IEEE, v. 25, n. 4, p. 2008–2025, 2017.

ESPOSITO, F. Catena: A distributed architecture for robust service function chain instantiation with guarantees. In: IEEE. *2017 IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2017. p. 1–9.

EUROSTAT. *Population on 1 January by age groups and sex - cities and greater cities*. 2020. <https://bit.ly/2YqqJeW>. Accessed: April, 2020.

FAN, J.; GUAN, C.; ZHAO, Y.; QIAO, C. Availability-aware mapping of service function chains. In: IEEE. *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. [S.l.], 2017. p. 1–9.

FAN, J.; JIANG, M.; QIAO, C. Carrier-grade availability-aware mapping of service function chains with on-site backups. In: IEEE. *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. [S.l.], 2017. p. 1–10.

FRANK, R. J.; DAVEY, N.; HUNT, S. P. Time series prediction and neural networks. *Journal of intelligent and robotic systems*, Springer, v. 31, p. 91–103, 2001.

GENS, F. The 3rd platform: Enabling digital transformation. *USA: IDC*, v. 209, 2013.

GERMAN, R. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. New York, NY, USA: John Wiley & Sons, Inc., 2000. ISBN 0471492582.

GHAZOUANI, S.; SLIMANI, Y. Towards a standardized cloud service description based on usdl. *Journal of Systems and Software*, Elsevier, v. 132, p. 1–20, 2017.

GHRADA, N.; ZHANI, M. F.; ELKHATIB, Y. Price and performance of cloud-hosted virtual network functions: Analysis and future challenges. In: IEEE. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. [S.l.], 2018. p. 482–487.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. *Deep learning*. [S.l.]: MIT press Cambridge, 2016.

GREFF, K.; SRIVASTAVA, R. K.; KOUTNÍK, J.; STEUNEBRINK, B. R.; SCHMIDHUBER, J. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, IEEE, v. 28, n. 10, p. 2222–2232, 2017.

GROUP, E. N. I. S. Network functions virtualisation (NFV)-network operator perspectives on industry progress. *ETSI*, 2013.

GUARINI, E.; MAGLI, F.; NOBOLO, A. Accounting for community building: the municipal amalgamation of milan in 1873–1876. *Accounting History Review*, Taylor & Francis, v. 28, n. 1-2, p. 5–30, 2018.

GUIDOTTI, R.; MONREALE, A.; RUGGIERI, S.; TURINI, F.; GIANNOTTI, F.; PEDRESCHI, D. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 51, n. 5, p. 1–42, 2018.

GUO, S.; DAI, Y.; XU, S.; QIU, X.; QI, F. Trusted cloud-edge network resource management: Drl-driven service function chain orchestration for iot. *IEEE Internet of Things Journal*, IEEE, 2019.

GUPTA, L.; SAMAKA, M.; JAIN, R.; ERBAD, A.; BHAMARE, D.; METZ, C. Colap: A predictive framework for service function chain placement in a multi-cloud environment. In: IEEE. *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.], 2017. p. 1–9.

GYŐRÖDI, C.; GYŐRÖDI, R.; PECHERLE, G.; OLAH, A. A comparative study: Mongodb vs. mysql. In: IEEE. *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. [S.l.], 2015. p. 1–6.

HAN, S.-Y.; LIANG, T. Reinforcement-learning-based vibration control for a vehicle semi-active suspension system via the ppo approach. *Applied Sciences*, MDPI, v. 12, n. 6, p. 3078, 2022.

HASSELT, H. P. van; GUEZ, A.; HESSEL, M.; MNIH, V.; SILVER, D. Learning values across many orders of magnitude. *Advances in Neural Information Processing Systems*, v. 29, 2016.

HASSELT, H. V. Double q-learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2010. p. 2613–2621.

HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, World Scientific, v. 6, n. 02, p. 107–116, 1998.

HONG, J.-y.; SUH, E.-h.; KIM, S.-J. Context-aware systems: A literature review and classification. *Expert Systems with applications*, Elsevier, v. 36, n. 4, p. 8509–8522, 2009.

HU, Y.; LOU, S.; WU, S.; YANG, L. Service function chain embedding framework for nfv-enabled iot application. In: IEEE. *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. [S.l.], 2020. p. 80–84.

HUA, Y.; ZHAO, Z.; LIU, Z.; CHEN, X.; LI, R.; ZHANG, H. Traffic prediction based on random connectivity in deep learning with long short-term memory. In: IEEE. *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. [S.l.], 2018. p. 1–6.

HUSSAIN, B.; DU, Q.; ZHANG, S.; IMRAN, A.; IMRAN, M. A. Mobile edge computing-based data-driven deep learning framework for anomaly detection. *IEEE Access*, IEEE, v. 7, p. 137656–137667, 2019.

INSTITUTE, E. T. S. *Network Functions Virtualization*. 2018. <https://bit.ly/3enE2mB>. Accessed: November, 2020.

IQBAL, M. F.; ZAHID, M.; HABIB, D.; JOHN, L. K. Efficient prediction of network traffic for real-time applications. *Journal of Computer Networks and Communications*, Hindawi, v. 2019, 2019.

JIM, M. *NFV Applications - Key Considerations for Profitability*. [S.l.]: Dialogic, 2015. <https://web.dialogic.com/making-nfv-profitable>.

JIN, H.; ZHU, X.; ZHAO, C. Computation offloading optimization based on probabilistic sfc for mobile online gaming in heterogeneous network. *IEEE Access*, IEEE, v. 7, p. 52168–52180, 2019.

JOZEFOWICZ, R.; ZAREMBA, W.; SUTSKEVER, I. An empirical exploration of recurrent network architectures. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015. p. 2342–2350.

KAISER, Ł.; SUTSKEVER, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

KANE, M. J.; PRICE, N.; SCOTCH, M.; RABINOWITZ, P. Comparison of arima and random forest time series models for prediction of avian influenza h5n1 outbreaks. *BMC bioinformatics*, BioMed Central, v. 15, n. 1, p. 1–9, 2014.

KARASU, S.; ALTAN, A. Recognition model for solar radiation time series based on random forest with feature selection approach. In: IEEE. *2019 11th international conference on electrical and electronics engineering (ELECO)*. [S.l.], 2019. p. 8–11.

KATSAROS, G.; MENZEL, M.; LENK, A.; REVELANT, J. R.; SKIPP, R.; EBERHARDT, J. Cloud application portability with tosca, chef and openstack. In: IEEE. *2014 IEEE international conference on cloud engineering*. [S.l.], 2014. p. 295–302.

KAUR, K.; MANGAT, V.; KUMAR, K. A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture. *Computer Science Review*, Elsevier, v. 38, p. 100298, 2020.

KAYEDPOUR, F.; AMIRI, M.; RAFIZADEH, M.; NIA, A. S. Multi-objective redundancy allocation problem for a system with repairable components considering instantaneous availability and strategy selection. *Reliability Engineering & System Safety*, Elsevier, v. 160, p. 11–20, 2017.

KHEZRI, H. R.; MOGHADAM, P. A.; FARSHBAFAN, M. K.; SHAH-MANSOURI, V.; KEBRIAEI, H.; NIYATO, D. Deep reinforcement learning for dynamic reliability aware nfv-based service provisioning. In: IEEE. *2019 IEEE Global Communications Conference (GLOBECOM)*. [S.l.], 2019. p. 1–6.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KOUAH, R.; ALLEG, A.; LARABA, A.; AHMED, T. Energy-aware placement for iot-service function chain. In: IEEE. *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. [S.l.], 2018. p. 1–7.

KRAUS, M.; FEUERRIEGEL, S.; OZTEKIN, A. Deep learning in business analytics and operations research: Models, applications and managerial implications. *European Journal of Operational Research*, Elsevier, v. 281, n. 3, p. 628–641, 2020.

KUBER, T.; SESKAR, I.; MANDAYAM, N. Traffic prediction by augmenting cellular data with non-cellular attributes. In: IEEE. *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.], 2021. p. 1–6.

KUMAR, A.; KAUR, A.; KUMAR, M. Face detection techniques: a review. *Artificial Intelligence Review*, Springer, v. 52, n. 2, p. 927–948, 2019.

KUMAR, U. System maintenance: Trends in management and technology. In: *Handbook of performability engineering*. [S.l.]: Springer, 2008. p. 773–787.

LANGE, S.; TU, N. V.; JEONG, S.-Y.; LEE, D.-Y.; KIM, H.-G.; HONG, J.; YOO, J.-H.; HONG, J. W.-K. A network intelligence architecture for efficient vnf lifecycle management. *IEEE Transactions on Network and Service Management*, IEEE, 2020.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LI, D.; HONG, P.; XUE, K.; PEI, J. Virtual network function placement and resource optimization in nfv and edge computing enabled networks. *Computer Networks*, Elsevier, v. 152, p. 12–24, 2019.

LI, G.; ZHOU, H.; FENG, B.; LI, G. Context-aware service function chaining and its cost-effective orchestration in multi-domain networks. *IEEE Access*, IEEE, v. 6, p. 34976–34991, 2018.

LI, G.; ZHOU, H.; FENG, B.; ZHANG, Y.; YU, S. Efficient provision of service function chains in overlay networks using reinforcement learning. *IEEE Transactions on Cloud Computing*, IEEE, 2019.

LI, H.; KORDI, M. E. Dsppv: Dynamic service function chains placement with parallelized virtual network functions in mobile edge computing. *Internet of Things*, Elsevier, v. 22, p. 100733, 2023.

LIMA, P. A.; NETO, A. S. B.; MACIEL, P. Data centers' services restoration based on the decision-making of distributed agents. *Telecommunication Systems*, Springer, p. 1–12, 2020.

LIN, H.-Y.; HSUEH, Y.-L.; LIE, W.-N. Abnormal event detection using microsoft kinect in a smart home. In: IEEE. *Computer Symposium (ICS), 2016 International*. [S.l.], 2016. p. 285–289.

LINGUAGLOSSA, L.; LANGE, S.; PONTARELLI, S.; RÉTVÁRI, G.; ROSSI, D.; ZINNER, T.; BIFULCO, R.; JARSCHEL, M.; BIANCHI, G. Survey of performance acceleration techniques for network function virtualization. *Proceedings of the IEEE*, IEEE, v. 107, n. 4, p. 746–764, 2019.

LIRA, V.; TAVARES, E.; OLIVEIRA, M.; SOUSA, E.; NOGUEIRA, B. Virtual network mapping considering energy consumption and availability. *Computing*, Springer, v. 101, n. 8, p. 937–967, 2019.

LIU, X.; CHENG, B.; WANG, S. Availability-aware and energy-efficient virtual cluster allocation based on multi-objective optimization in cloud datacenters. *IEEE Transactions on Network and Service Management*, IEEE, 2020.

LUO, Z.; WU, C.; LI, Z.; ZHOU, W. Scaling geo-distributed network function chains: A prediction and learning framework. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 37, n. 8, p. 1838–1850, 2019.

LYNN, T.; GOURINOVITCH, A.; SVOROBEH, S.; ENDO, P. T. *Software Defined Networking and Network Functions Virtualization - Market Briefing*. 2018. <https://recap-project.eu/media/market-briefings/>.

MA, B.; GUO, W.; ZHANG, J. A survey of online data-driven proactive 5g network optimisation using machine learning. *IEEE Access*, IEEE, v. 8, p. 35606–35637, 2020.

MARSAN, M. A.; BALBO, G.; CONTE, G.; DONATELLI, S.; FRANCESCHINIS, G. *Modelling with Generalized Stochastic Petri Nets*. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN 0471930598.

MARTÍN-PÉREZ, J.; BERNARDOS, C. J. Multi-domain vnf mapping algorithms. In: IEEE. *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. [S.l.], 2018. p. 1–6.

MASOUDI, R.; GHAFFARI, A. Software defined networks: A survey. *Journal of Network and computer Applications*, Elsevier, v. 67, p. 1–25, 2016.

MECHTRI, M.; GHRIBI, C.; SOUALAH, O.; ZEGHLACHE, D. Nfv orchestration framework addressing sfc challenges. *IEEE Communications Magazine*, IEEE, v. 55, n. 6, p. 16–23, 2017.

MEDEDOVIC, E.; DOUROS, V. G.; MÄHÖNEN, P. Node centrality metrics for hotspots analysis in telecom big data. In: IEEE. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.], 2019. p. 417–422.

MEHRAGHDAM, S.; KARL, H. Placement of services with flexible structures specified by a yang data model. In: IEEE. *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. [S.l.], 2016. p. 184–192.

MENDIS, H. K.; LI, F. Y. Achieving ultra reliable communication in 5g networks: A dependability perspective availability analysis in the space domain. *IEEE Communications Letters*, IEEE, v. 21, n. 9, p. 2057–2060, 2017.

MIJUMBI, R.; SERRAT, J.; GORRICHO, J.-L.; BOUTEN, N.; TURCK, F. D.; DAVY, S. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In: IEEE. *Proceedings of the 2015 1st IEEE conference on network softwarization (NetSoft)*. [S.l.], 2015. p. 1–9.

MIRJALILY, G.; ZHIQUAN, L. Optimal network function virtualization and service function chaining: A survey. *Chinese Journal of Electronics*, IET, v. 27, n. 4, p. 704–717, 2018.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In: *International conference on machine learning*. [S.l.: s.n.], 2016. p. 1928–1937.

MOLLOY, M. K. Performance analysis using stochastic petri nets. *IEEE Transactions on computers*, IEEE Computer Society, v. 31, n. 09, p. 913–917, 1982.

MOLLOY, M. K. Performance analysis using stochastic petri nets. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 31, p. 913–917, September 1982. ISSN 0018-9340.

MOUALLA, G.; TURLETTI, T.; SAUCEZ, D. An availability-aware sfc placement algorithm for fat-tree data centers. In: IEEE. *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. [S.l.], 2018. p. 1–4.

MOUSAVI, S. S.; SCHUKAT, M.; HOWLEY, E. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, IET, v. 11, n. 7, p. 417–423, 2017.

MUNDIE, C.; VRIES, P. de; HAYNES, P.; CORWINE, M. *Trustworthy computing*. [S.l.], 2002.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, Apr 1989. ISSN 0018-9219.

NAREJO, S.; PASERO, E. An application of internet traffic prediction with deep neural network. *Multidisciplinary Approaches to Neural Computing*, Springer, p. 139–149, 2018.

NGUYEN, D. T.; PHAM, C.; NGUYEN, K. K.; CHERIET, M. Placement and chaining for run-time iot service deployment in edge-cloud. *IEEE Transactions on Network and Service Management*, IEEE, v. 17, n. 1, p. 459–472, 2019.

NING, K.; WANG, H.; ZHANG, Z.; XU, Z.; SHU, X. Parallel deployment of vnfs in service function chain: Benefit or not? In: IEEE. *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. [S.l.], 2022. p. 628–635.

OCHODEK, M.; KOPCZYŃSKA, S. Perceived importance of agile requirements engineering practices–a survey. *Journal of Systems and Software*, Elsevier, v. 143, p. 29–43, 2018.

ORDÓÑEZ, F. J.; ROGGEN, D. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 16, n. 1, p. 115, 2016.

OSBAND, I.; BLUNDELL, C.; PRITZEL, A.; ROY, B. V. Deep exploration via bootstrapped dqn. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2016. p. 4026–4034.

PALHARES, A.; SANTOS, M.; ENDO, P.; VITALINO, J.; RODRIGUES, M.; GONÇALVES, G.; SADOK, D.; SEFIDCON, A.; WUHIB, F. Joint allocation of nodes and links with load balancing in network virtualization. In: IEEE. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. [S.l.], 2014. p. 148–155.

PAN, J.; WANG, X.; CHENG, Y.; YU, Q. Multisource transfer double dqn based on actor learning. *IEEE transactions on neural networks and learning systems*, IEEE, v. 29, n. 6, p. 2227–2238, 2018.

PANDEY, D.; SUMAN, U.; RAMANI, A. K. An effective requirement engineering process model for software development and requirements management. In: IEEE. *2010 International Conference on Advances in Recent Technologies in Communication and Computing*. [S.l.], 2010. p. 287–291.

PANDEY, S.; NGUYEN, T. V.; YOO, J.-H.; HONG, J. W.-K. Edgedqn: Multiple sfc placement in edge computing environment. In: IEEE. *2021 17th International Conference on Network and Service Management (CNSM)*. [S.l.], 2021. p. 301–309.

PEI, J.; HONG, P.; XUE, K.; LI, D. Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 30, n. 10, p. 2179–2192, 2018.

PENG, B.; LI, X.; GAO, J.; LIU, J.; CHEN, Y.-N.; WONG, K.-F. Adversarial advantage actor-critic model for task-completion dialogue policy learning. In: IEEE. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2018. p. 6149–6153.

PEREIRA, P.; MELO, C.; ARAUJO, J.; DANTAS, J.; SANTOS, V.; MACIEL, P. Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. *The Journal of Supercomputing*, Springer, p. 1–28, 2022.

PEZOA, F.; REUTTER, J. L.; SUAREZ, F.; UGARTE, M.; VRGOČ, D. Foundations of json schema. In: *Proceedings of the 25th International Conference on World Wide Web*. [S.l.: s.n.], 2016. p. 263–273.

PONTES, F. J.; AMORIM, G.; BALESTRASSI, P. P.; PAIVA, A.; FERREIRA, J. R. Design of experiments and focused grid search for neural network parameter optimization. *Neurocomputing*, Elsevier, v. 186, p. 22–34, 2016.

POURSAFAR, N.; ALAHI, M. E. E.; MUKHOPADHYAY, S. Long-range wireless technologies for iot applications: A review. In: IEEE. *2017 Eleventh International Conference on Sensing Technology (ICST)*. [S.l.], 2017. p. 1–6.

PRAJAM, S.; WECHTAISONG, C.; KHAN, A. A. Applying machine learning approaches for network traffic forecasting. *Indian Journal of Computer Science and Engineering*, v. 13, n. 2, p. 324–335, 2022.

QIANG, W.; ZHONGLI, Z. Reinforcement learning model, algorithms and its application. In: IEEE. *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*. [S.l.], 2011. p. 1143–1146.

QIU, C.; ZHANG, Y.; FENG, Z.; ZHANG, P.; CUI, S. Spatio-temporal wireless traffic prediction with recurrent neural network. *IEEE Wireless Communications Letters*, IEEE, v. 7, n. 4, p. 554–557, 2018.

QU, L.; ASSI, C.; SHABAN, K. Delay-aware scheduling and resource optimization with network function virtualization. *IEEE Transactions on Communications*, IEEE, v. 64, n. 9, p. 3746–3758, 2016.

RAVICHANDIRAN, S. *Hands-on Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. [S.l.]: Packt Publishing Ltd, 2018.

RAY, P. P.; KUMAR, N. Sdn/nfv architectures for edge-cloud oriented iot: A systematic review. *Computer Communications*, Elsevier, v. 169, p. 129–153, 2021.

RECSE, A.; SZABO, R.; NEMETH, B. Elastic resource management and network slicing for iot over edge clouds. In: *Proceedings of the 10th International Conference on the Internet of Things*. [S.l.: s.n.], 2020. p. 1–8.

REDDY, R. V.; MURALI, D.; RAJESHWAR, J. Context-aware middleware architecture for iot-based smart healthcare applications. In: *Innovations in Computer Science and Engineering*. [S.l.]: Springer, 2019. p. 557–567.

REN, W.; SUN, Y.; LUO, H.; OBAIDAT, M. S. A new scheme for iot service function chains orchestration in sdn-iot network systems. *IEEE Systems Journal*, IEEE, v. 13, n. 4, p. 4081–4092, 2019.

RIERA, J. F.; ESCALONA, E.; BATALLE, J.; GRASA, E.; GARCIA-ESPIN, J. A. Virtual network function scheduling: Concept and challenges. In: IEEE. *2014 international conference on smart communications in network technologies (SaCoNeT)*. [S.l.], 2014. p. 1–5.

SAHOO, J.; MOHAPATRA, S.; LATH, R. Virtualization: A survey on concepts, taxonomy and associated security issues. In: IEEE. *2010 Second International Conference on Computer and Network Technology*. [S.l.], 2010. p. 222–226.

SANTOS, G. L.; BEZERRA, D. d. F.; ROCHA, E. d. S.; FERREIRA, L.; MOREIRA, A. L. C.; GONÇALVES, G. E.; MARQUEZINI, M. V.; RECSE, Á.; MEHTA, A.; KELNER, J. et al. Service function chain placement in distributed scenarios: a systematic review. *Journal of Network and Systems Management*, Springer, v. 30, n. 1, p. 1–39, 2022.

SANTOS, G. L.; ENDO, P. T.; LISBOA, M. F. F. da S.; SILVA, L. G. F. da; SADOK, D.; KELNER, J.; LYNN, T. et al. Analyzing the availability and performance of an e-health system integrated with edge, fog and cloud infrastructures. *Journal of Cloud Computing*, Springer, v. 7, n. 1, p. 16, 2018.

SANTOS, G. L.; ENDO, P. T.; LYNN, T.; SADOK, D.; KELNER, J. Automating the service function chain availability assessment. In: IEEE. *2021 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2021. p. 1–7.

SANTOS, G. L.; ENDO, P. T.; LYNN, T.; SADOK, D.; KELNER, J. A reinforcement learning-based approach for availability-aware service function chain placement in large-scale networks. *Future Generation Computer Systems*, Elsevier, 2022.

SANTOS, G. L.; ENDO, P. T.; SADOK, D.; KELNER, J. When 5g meets deep learning: a systematic review. *Algorithms*, MDPI, v. 13, n. 9, p. 208, 2020.

SANTOS, G. L.; ENDO, P. T.; SADOK, D.; KELNER, J. Spider: An availability-aware framework for the service function chain placement in distributed scenarios. *Journal of Software: Practice and Experience*, Elsevier, 2022.

SANTOS, G. L.; LYNN, T.; KELNER, J.; ENDO, P. T. Availability-aware and energy-aware dynamic sfc placement using reinforcement learning. *The Journal of Supercomputing*, Springer, p. 1–30, 2021.

SANTOS, G. L.; ROSATI, P.; LYNN, T.; KELNER, J.; SADOK, D.; ENDO, P. T. Predicting short-term mobile internet traffic from internet activity using recurrent neural networks. *arXiv preprint arXiv:2010.05741*, 2020.

SANTOS, J.; WAUTERS, T.; VOLCKAERT, B.; TURCK, F. D. Towards delay-aware container-based service function chaining in fog computing. In: IEEE. *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. [S.l.], 2020. p. 1–9.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

SHAH, H. A.; ZHAO, L. Multi-agent deep reinforcement learning based virtual resource allocation through network function virtualization in internet of things. *IEEE Internet of Things Journal*, IEEE, 2020.

SHAH, J.; DUBARIA, D. Building modern clouds: using docker, kubernetes & google cloud platform. In: IEEE. *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.], 2019. p. 0184–0189.

SHANG, X.; LI, Z.; YANG, Y. Placement of highly available virtual network functions through local rerouting. In: IEEE. *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. [S.l.], 2018. p. 80–88.

SHEN, W.; ZHANG, H.; GUO, S.; ZHANG, C. Time-wise attention aided convolutional neural network for data-driven cellular traffic prediction. *IEEE Wireless Communications Letters*, IEEE, 2021.

SHEN, Z.; ZHANG, Y. An nfv framework for supporting elastic scaling of service function chain. In: IEEE. *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. [S.l.], 2018. p. 566–571.

SIMONINI, T. *Proximal Policy Optimization (PPO) with Sonic the Hedgehog 2 and 3*. [S.l.]: Towards Data Science, 2018. <https://towardsdatascience.com/proximal-policy-optimization-ppo-with-sonic-the-hedgehog-2-and-3-c9c21dbed5e>.

SOUALAH, O.; MECHTRI, M.; GHRIBI, C.; ZEGHLACHE, D. A link failure recovery algorithm for virtual network function chaining. In: IEEE. *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. [S.l.], 2017. p. 213–221.

SOUZA, R.; DIAS, K.; FERNANDES, S. Nfv data centers: A systematic review. *IEEE Access*, IEEE, v. 8, p. 51713–51735, 2020.

SUBRAMANYA, T.; HARUTYUNYAN, D.; RIGGIO, R. Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks. *Computer Networks*, Elsevier, v. 166, p. 106980, 2020.

SUN, G.; LI, Y.; YU, H.; VASILAKOS, A. V.; DU, X.; GUIZANI, M. Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Generation Computer Systems*, Elsevier, v. 91, p. 347–360, 2019.

SUN, L.; DONG, H.; ASHRAF, J. Survey of service description languages and their issues in cloud computing. In: IEEE. *2012 Eighth International Conference on Semantics, Knowledge and Grids*. [S.l.], 2012. p. 128–135.

SUN, P.; LAN, J.; LI, J.; GUO, Z.; HU, Y. Combining deep reinforcement learning with graph neural networks for optimal vnf placement. *IEEE Communications Letters*, IEEE, 2020.

SUN, X.; MA, S.; LI, Y.; WANG, D.; LI, Z.; WANG, N.; GUI, G. Enhanced echo-state restricted boltzmann machines for network traffic prediction. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 2, p. 1287–1297, 2019.

SURIANO, A.; STRICCOLI, D.; PIRO, G.; BOLLA, R.; BOGGIA, G. Attestation of trusted and reliable service function chains in the etsi-nfv framework. In: IEEE. *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2020. p. 479–486.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

SYAKUR, M.; KHOTIMAH, B.; ROCHMAN, E.; SATOTO, B. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In: IOP PUBLISHING. *IOP Conference Series: Materials Science and Engineering*. [S.l.], 2018. v. 336, n. 1, p. 012017.

SYARIF, I.; PRUGEL-BENNETT, A.; WILLS, G. Svm parameter optimization using grid search and genetic algorithm to improve classification performance. *Telkomnika*, Ahmad Dahlan University, v. 14, n. 4, p. 1502, 2016.

TASHTARIAN, F.; ZHANI, M. F.; FATEMIPOUR, B.; YAZDANI, D. Codec: a cost-effective and delay-aware sfc deployment. *IEEE Transactions on Network and Service Management*, IEEE, v. 17, n. 2, p. 793–806, 2019.

TAVAKOLI-SOMEH, S.; REZVANI, M. H. Multi-objective virtual network function placement using nsga-ii meta-heuristic approach. *The Journal of Supercomputing*, Springer, v. 75, n. 10, p. 6451–6487, 2019.

TORQUATO, M.; TORQUATO, L.; MACIEL, P.; VIEIRA, M. Iaas cloud availability planning using models and genetic algorithms. In: IEEE. *2019 9th Latin-American Symposium on Dependable Computing (LADC)*. [S.l.], 2019. p. 1–10.

TORQUATO, M.; UMESH, I.; MACIEL, P. Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration. *The Journal of Supercomputing*, Springer, v. 74, n. 9, p. 4817–4841, 2018.

TOUMI, N.; BAGAA, M.; KSENTINI, A. Hierarchical multi-agent deep reinforcement learning for sfc placement on multiple domains. In: IEEE. *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. [S.l.], 2021. p. 299–304.

TOUMI, N.; BAGAA, M.; KSENTINI, A. On using deep reinforcement learning for multi-domain sfc placement. In: IEEE. *2021 IEEE Global Communications Conference (GLOBECOM)*. [S.l.], 2021. p. 1–6.

TOUMI, N.; BERNIER, O.; MEDDOUR, D.-E.; KSENTINI, A. On using physical programming for multi-domain sfc placement with limited visibility. *IEEE Transactions on Cloud Computing*, IEEE, 2020.

TOUMI, N.; BERNIER, O.; MEDDOUR, D.-E.; KSENTINI, A. On cross-domain service function chain orchestration: An architectural framework. *Computer Networks*, Elsevier, v. 187, p. 107806, 2021.

TRIVEDI, K. S. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. New York: John Wiley and Sons, 2001.

TROFIMOVICH, J. Comparison of neural network architectures for sentiment analysis of russian tweets. In: *Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue*. [S.l.: s.n.], 2016. p. 50–59.

TROIA, S.; ALVIZU, R.; MAIER, G. Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks. *IEEE Access*, IEEE, v. 7, p. 167944–167957, 2019.

TROIA, S.; ALVIZU, R.; ZHOU, Y.; MAIER, G.; PATTAVINA, A. Deep learning-based traffic prediction for network optimization. In: IEEE. *2018 20th International Conference on Transparent Optical Networks (ICTON)*. [S.l.], 2018. p. 1–4.

TYRALIS, H.; PAPACHARALAMPOUS, G. Variable selection in time series forecasting using random forests. *Algorithms*, MDPI, v. 10, n. 4, p. 114, 2017.

VENTRE, P. L.; PISA, C.; SALSANO, S.; SIRACUSANO, G.; SCHMIDT, F.; LUNGARONI, P.; BLEFARI-MELAZZI, N. Performance evaluation and tuning of virtual infrastructure managers for (micro) virtual network functions. In: IEEE. *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. [S.l.], 2016. p. 141–147.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Designing context-sensitive systems: An integrated approach. *Expert Systems with Applications*, Elsevier, v. 38, n. 2, p. 1119–1138, 2011.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. [S.l.], 2001. v. 1, p. I–I.

WANG, L.; MAO, W.; ZHAO, J.; XU, Y. Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement. *IEEE Transactions on Network and Service Management*, IEEE, v. 18, n. 1, p. 118–132, 2021.

WANG, M.; CHENG, B.; WANG, S.; CHEN, J. Availability-and traffic-aware placement of parallelized sfc in data center networks. *IEEE Transactions on Network and Service Management*, IEEE, v. 18, n. 1, p. 182–194, 2021.

WANG, S.; CAO, H.; YANG, L. A survey of service function chains orchestration in data center networks. In: IEEE. *2020 IEEE Globecom Workshops (GC Wkshps*. [S.l.], 2020. p. 1–6.

WANG, W.; LU, Y. Analysis of the mean absolute error (mae) and the root mean square error (rmse) in assessing rounding model. In: IOP PUBLISHING. *IOP Conference Series: Materials Science and Engineering*. [S.l.], 2018. v. 324, n. 1, p. 012049.

WANG, Y.; HE, H.; TAN, X. Truly proximal policy optimization. In: PMLR. *Uncertainty in Artificial Intelligence*. [S.l.], 2020. p. 113–122.

WANG, Y.; VELSWAMY, K.; HUANG, B. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. *Processes*, MDPI, v. 5, n. 3, p. 46, 2017.

WÓJCIK, P. I.; KURDZIEL, M. Training neural networks on high-dimensional data using random projection. *Pattern Analysis and Applications*, Springer, v. 22, n. 3, p. 1221–1231, 2019.

XIAO, Y.; ZHANG, Q.; LIU, F.; WANG, J.; ZHAO, M.; ZHANG, Z.; ZHANG, J. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In: *Proceedings of the International Symposium on Quality of Service*. [S.l.: s.n.], 2019. p. 1–10.

XIAO, Y.; ZHANG, Q.; LIU, F.; WANG, J.; ZHAO, M.; ZHANG, Z.; ZHANG, J. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In: *Proceedings of the International Symposium on Quality of Service*. [S.l.: s.n.], 2019. p. 1–10.

XIE, Y.; LIU, Z.; WANG, S.; WANG, Y. Service function chaining resource allocation: A survey. *arXiv preprint arXiv:1608.00095*, 2016.

XIE, Y.; WANG, S.; DAI, Y. Revenue-maximizing virtualized network function chain placement in dynamic environment. *Future Generation Computer Systems*, Elsevier, 2020.

XU, Q.; GAO, D.; LI, T.; ZHANG, H. Low latency security function chain embedding across multiple domains. *IEEE Access*, IEEE, v. 6, p. 14474–14484, 2018.

XU, S.; LIAO, B.; HU, B.; HAN, C.; YANG, C.; WANG, Z.; XIONG, A. A reliability-and-energy-balanced service function chain mapping and migration method for internet of things. *IEEE Access*, IEEE, v. 8, p. 168196–168209, 2020.

XU, Z.; ZHANG, X.; YU, S.; ZHANG, J. Energy-efficient virtual network function placement in telecom networks. In: IEEE. *2018 IEEE International Conference on Communications (ICC)*. [S.l.], 2018. p. 1–7.

YAN, S. *Understanding LSTM and its diagrams*. 2016. <https://bit.ly/2JvRwhr>. Accessed: August, 2018.

YAN, S. *Understanding LSTM and its diagrams*. [S.l.]: ML Review, 2016. <https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714>.

YAO, Y.; GUO, S.; LI, P.; LIU, G.; ZENG, Y. Forecasting assisted vnf scaling in nfv-enabled networks. *Computer Networks*, Elsevier, v. 168, p. 107040, 2020.

YUAN, C.; YANG, H. Research on k-value selection method of k-means clustering algorithm. *J—Multidisciplinary Scientific Journal*, Multidisciplinary Digital Publishing Institute, v. 2, n. 2, p. 226–235, 2019.

ZANG, Y.; NI, F.; FENG, Z.; CUI, S.; DING, Z. Wavelet transform processing for cellular traffic prediction in machine learning networks. In: IEEE. *2015 IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*. [S.l.], 2015. p. 458–462.

ZENG, Q.; SUN, Q.; CHEN, G.; DUAN, H.; LI, C.; SONG, G. Traffic prediction of wireless cellular networks based on deep transfer learning and cross-domain data. *IEEE Access*, IEEE, v. 8, p. 172387–172397, 2020.

ZHANG, C.; PATRAS, P. Long-term mobile traffic forecasting using deep spatio-temporal neural networks. In: *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. [S.l.: s.n.], 2018. p. 231–240.

ZHANG, C.; PATRAS, P.; HADDADI, H. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, IEEE, v. 21, n. 3, p. 2224–2287, 2019.

ZHANG, C.; ZHANG, H.; YUAN, D.; ZHANG, M. Citywide cellular traffic prediction based on densely connected convolutional neural networks. *IEEE Communications Letters*, IEEE, v. 22, n. 8, p. 1656–1659, 2018.

ZHANG, D.; LIU, L.; XIE, C.; YANG, B.; LIU, Q. Citywide cellular traffic prediction based on a hybrid spatiotemporal network. *Algorithms*, Multidisciplinary Digital Publishing Institute, v. 13, n. 1, p. 20, 2020.

ZHANG, X.; SHEN, F.; ZHAO, J.; YANG, G. Time series forecasting using gru neural network with multi-lag after decomposition. In: SPRINGER. *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V 24*. [S.l.], 2017. p. 523–532.

ZHANG, X.; XU, Z.; FAN, L.; YU, S.; QU, Y. Near-optimal energy-efficient algorithm for virtual network function placement. *IEEE Transactions on Cloud Computing*, IEEE, 2019.

APPENDIX A – SFC AVAILABILITY ANALYSIS

We conducted experiments to evaluate the SFC availability for different placement configurations. We assume an SFC with four VNF types as per Table 23. We assume that the VNF will be deployed as virtual machines based on the parameters reported in (GHRADA; ZHANI; ELKHATIB, 2018). These parameters were based on Amazon EC2, one of the leading Infrastructure as a Service (IaaS) providers. Each VNF type is allocated a computing requirement in terms of vCPU and memory. The amount of resources and the respective price (in $/hour) are based on the EC2 service.

Table 23 – Parameters of different VNF types

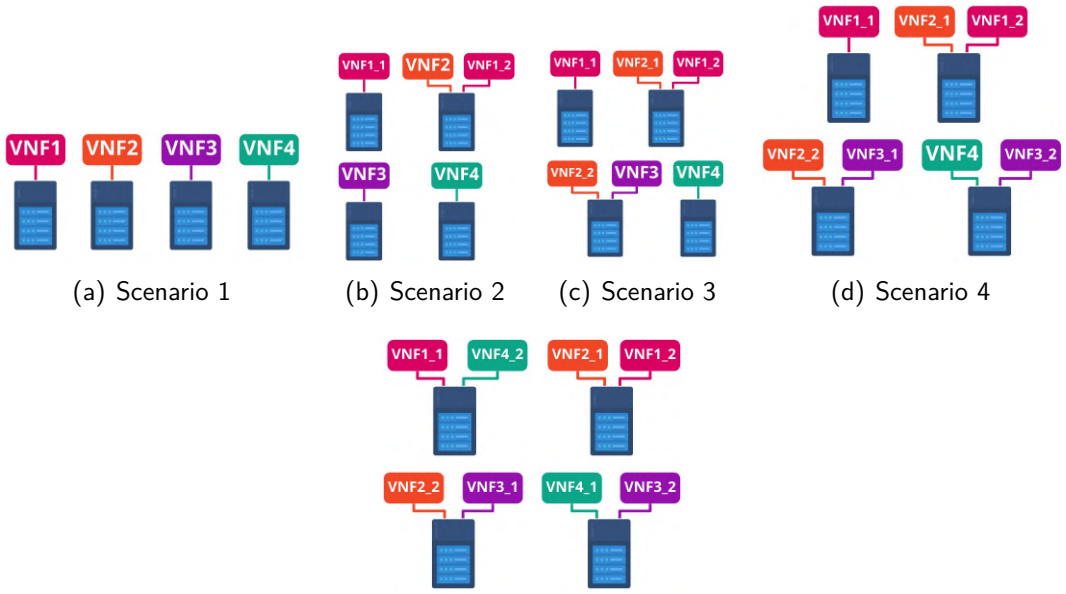
VNF type	Instance Type	vCPU Cores	Memory (GiB)	Price ($/hour)	MTTF (hours)
1	t2.micro	1	1	0.012	2880
2	t2.small	1	2	0.023	3024
3	t2.medium	2	4	0.047	3175.2
4	t2.xlarge	4	16	0.188	3333.96

Source: the author (2023).

We also assume that the servers and VNFs failure and repair events follow an exponential distribution (ANDRADE et al., 2017). Therefore, the stochastic transition that the SPN models receive as input is an MTTF value, for failure events, or an MTTR value, for repair events. We assume baseline MTTF and MTTR values for virtual machines of 2880 hours and 0.5 hours, respectively (TORQUATO; UMESH; MACIEL, 2018). As the virtual machine price increases, we assume that the MTTF increases 5%, in order to give different reliability characteristics for the different VNF types. The MTTR is the same for all VNF types (0.5 hours) since it is related to the maintenance strategies used by a firm, such as the time for finding and subsequently repairing a fault (KUMAR, 2008). We also assume that the MTTF and MTTR values for a server are 8760 hours and 1.67 hours, respectively (ARAUJO et al., 2014). These values can be adapted for different scenarios according to the network manager requirements.

Figure 51 presents the SFC placement configurations evaluated in this study. The basic scenario illustrated in Figure 52(a) presents an SFC placement without redundancy. A replica of each type of VNF is added in the other scenarios, always on a different server than the primary VNF, and in this way increase the availability. For example, in Scenario 2 (Figure 52(b)), a replica of VNF1 (VNF1_2) is added on a server other than the main VNF (VNF1_1). Scenario

Figure 51 – SFC placement scenarios



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3 (d) Scenario 4

(e) Scenario 5

Source: the author (2023).

5 (Figure 52(e)) presents a scenario where the whole SFC is redundant.

We used the GSPN Framework[1] to create, visualise, execute, and analyse SPN models. This framework is written in Python and solves the SPN models in an analytical way. It generates the respective CTMC and calculates the probability of tokens being in the places needed to calculate the overall SFC availability.

A.1 RESULTS

All experiments were run on a machine with an Intel(R) Core(TM) i5-3470 CPU at 3.20GHz and 12GB of RAM running Ubuntu 18.04.5 LTS. Table 24 shows the results for all scenarios presented in Figure 51. The availability is calculated using the SPN models, the downtime (considering the period of one year) estimated using Equation A.1, and the placement cost is calculated as the sum of price of all VNF instances (illustrated in Table 23) for each scenario multiplied by 8760 (number of hours in one year).

$$downtime = 8760 \times (1 - availability) \tag{A.1}$$

In general, as redundant VNFs are added in each scenario, the overall SFC availability

[1] <https://github.com/cazevedo/gspn-framework>

Table 24 – Availability, downtime, and placement cost results for all scenarios

Scenario	Availability (%)	Downtime (hours/year)	Placement Cost ($/year)
1	0.998593	12.321	2365.20
2	0.998956	9.138	2470.32
3	0.999312	6.026	2671.80
4	0.999659	2.983	3083.52
5	0.999999	0.004	4730.40

Source: the author (2023).

increases considerably, which results in a reduction in downtime. For Scenario 1 (no redundancy), the availability is 0.998593%, which results in a downtime of 12.321 hours. With the addition of one replica of VNF1 (Scenario 2), the availability increases to 0.998956%, reducing SFC downtime to 9.138 hours, a difference of 3.183 hours. When half of the SFC is replicated (Scenario 3), the availability increases to 0.999312%, resulting in downtime of 6.026 hours, a reduction of 51.09% in comparison to Scenario 1. For the Scenario 4, the availability increases to 0.999659%, with the respective downtime of 2.983 hours. Finally, when the SFC is fully replicated (Scenario 5), availability reaches 0.999999%, with the lowest downtime: 0.004 hours.

Although the addition of redundant VNFs in each scenario increases the SFC availability, it also increases the SFC placement cost. The cost of Scenario 1 is $2365.20; the addition of a VNF of type 1 (Scenario 2) increases the cost to $2470.32, a difference of $105.12. For Scenario 3, where half of SFC presents redundancy, the placement cost is $2671.80, an increase of 13% in comparison with Scenario 1. The cost of Scenario 4 is $3083.52, an increase of 30.37% from the Scenario 1. In Scenario 5, with redundancy in all VNFs of the SFC, the placement cost is 4730.40$, an increase of 77% on Scenario 3 and double of cost of Scenario 1.

As redundant VNF instances are added, the availability increases while the downtime decreases. These aspects must be considered by the network manager, since different customers can have different requirements. For instance, traditional IT applications have availability in the order of 2'9s to 3'9s (i.e., 0.99% and 0.999%), while telecommunication service providers require their network service to be “always on" with an availability of 0.99999% or 0.999999% (FAN; JIANG; QIAO, 2017). Therefore, based on Table 24, only Scenario 5 is capable of meeting the requirements of Telecommunications services, i.e. full redundancy of the SFC. Other

critical applications can require a high availability level, such as traffic systems and critical healthcare (POURSAFAR; ALAHI; MUKHOPADHYAY, 2017). For scenarios with lower availability requirements, other redundancy configurations could be used. It is important highlight the trade-off between availability and placement cost. To achieve the availability required for critical applications and telecommunications services, the SFC must be duplicated, which doubles the placement cost. However, for applications with lower availability requirements, other redundancy strategies can be used based on cost considerations. For example, if the client's requirement is two 9's, the placement cost will be around $2365.20 (Scenario 1). On the other hand, if the customer requirements increase to three 9's, the allocation cost will be at least $2671.80 (Scenario 3).

As discussed we created an algorithm that created comparatively smaller SPN models to assess the SFC availability. To evaluate the performance gain of this algorithm, we carried out experiments comparing the algorithm with the baseline SPN models. For each scenario presented in Figure 51, we created an SPN with building blocks for each component presented in the scenarios. We then calculated the SFC availability using our proposed algorithm 1. We performed the experiments 30 times to calculate the runtime. The average results are presented in Table 25.

Table 25 – Runtime comparison of baseline SPN models and proposed algorithm

Scenario	Baseline SPN model (in seconds)	Proposed Algorithm (in seconds)
1	0.68	0.02
2	2.32	0.05
3	9.21	0.28
4	40.22	0.49
5	196.34	0.89

Source: the author (2023).

The runtime increases across the scenarios as more components are considered in the models generated by our algorithm and in the baseline SPN models. However, the runtime of the baseline SPN models are higher than the proposed algorithm for all scenarios. One can note that the runtime of the baseline SPN models increases significantly across the scenarios, reaching 196.34 seconds on average. In contrast, the proposed algorithm never exceeds one second on average to calculate the availability of the SFC; the highest runtime reported was 0.89 seconds for Scenario 5. This runtime reduction can be explained by the smaller SPNs generated by the proposed algorithm, and the subsequent performance gain in calculating

the availability of each type of VNF separately. Since the SPNs are smaller, the respective CTMCs have fewer states and therefore can be generated and resolved more quickly. This has practical implications. Where placement decisions must be made quickly, the baseline models are not suitable due to the substantially longer runtime. The significantly faster runtime of the proposed algorithm suggests it represents a potentially good solution for optimisation algorithms, where the availability need to be evaluated several times, such as meta heuristics.

APPENDIX B – SFC REQUEST DETAILS

Endpoint 10, listed in Table 10, is used to create a new SFC request. It implements a POST method. A user must pass the information about the new SFC as JSON data, as illustrated below:

```
{
    "id": 0,
    "name": "My SFC",
    "destination": "master-node",
    "source": "minion-1",
    "VNFs": [
        {
            "_id": "1",
            "name": "compress-image",
            "id": "1",
            "resources": {
                "cpu": 1,
                "memory": 1,
                "storage": 1
            },
            "mttf": 1,
            "mttr": 1,
            "availability": 1,
            "path_to_files": "/home/guto/vnf_catalog/compress
               -image"
        },
        {
            "_id": "0",
            "name": "firewall",
            "id": "0",
            "resources": {
                "cpu": 1,
```

```
                "memory": 1,
                "storage": 1
            },
            "mttf": 1,
            "mttr": 1,
            "availability": 1,
            "path_to_files": "/home/guto/vnf_catalog/firewall
                "
        }
    ],
    "flow_entries": [
        {
            "source": "Source",
            "destination": "compress-image",
            "resources": {
                "bandwidth": 1,
                "cost": 1
            }
        },
        {
            "source": "compress-image",
            "destination": "firewall",
            "resources": {
                "bandwidth": 1,
                "cost": 1
            }
        },
        {
            "source": "firewall",
            "destination": "destination",
            "resources": {
                "bandwidth": 1,
```

```
58                  "cost": 1
59              }
60          }
61      ],
62      "requirements": {
63          "availability": 0.99
64      }
65  }
```

One may note that a user needs to specify some basic information about the SFC, such as its ID, name, source and destination nodes (see Figure 10). The VNF list needs to be defined as well. Each VNF has the information defined in the VNF template, as illustrated in Table 8. The order of VNFs specified in the list is not relevant for the SFC composition, since the virtual links define how they are connected. In addition, a user also must define the flow entries list. Each virtual link defines the source and the destination among the VNFs; and the source and the destination physical nodes.

From the SFC request showed in the list defined in line 6, we can see that there are two VNFs: a firewall and a compress image function. According to the virtual links list, the first VNF is the compress image function, since it connects to the source node. Both VNFs are connected by the second virtual link of the list, and finally the firewall is connected to the destination.

A user can specify the requirements for the SFC and define these in a specific field. In the example illustrated above, we define the availability requirement of the SFC in line 63. Although the focus of the SPIDER is on the availability, other requirements could be considered in the future for the placement and inserted through their JSON specification.

APPENDIX C – DAEMON CONFIGURATION

The daemon module is implemented using the Python language. To monitor the computing resources usage, we use the psutil tool[1], which is a cross-platform library designed for monitoring information related to running processes and system utilization, such as CPU, memory, and disks. Therefore, we may collect information about the number of CPU cores used, the amount of memory usage, and the total storage consumption. To obtain data on network usage, the daemon uses the vnfStat tool[2], which is a network traffic monitor that uses the network interface statistics provided by the kernel as information source. Using vnfStat, the daemon is able to collect link usage on a given interface in the last hour (this time is configurable, and the collection time can be increased).

Based on these two tools, the daemon is able to collect updated information about computing and network resources used by a node. When a node is included in the SPIDER framework, a user must specify a set of data fields in a JSON file, as illustrated below:

```
1
2  {
3      "node":{
4          "id": "node_1",
5          "name": "node_1",
6          "latitude": 19.994326843893894,
7          "longitude": 73.78965468852559,
8          "resources": {"cpu": 10, "memory": 10, "storage": 10}
               ,
9          "available_resources": {"cpu": 10, "memory": 10, "
               storage": 10},
10         "node_cost": 500,
11         "mttf": 2880,
12         "mttr": 1.5,
13         "availability": 0.9994794377928162,
14         "energy": 20,
15         "metadata": [],
```

[1] <https://psutil.readthedocs.io/en/latest/>
[2] <https://github.com/vergoh/vnstat>

```
        "capabilities":
            {"supported_VNFs":
                [
                    {"id": 0, "type": "compress-image"},
                    {"id": 1, "type": "firewall"}
                ]
            },
        "ports": ["wlo1"],
    },


    "links":[
        {
            "id":0,
            "destination":{
                            "id":"router-1",
                            "name":"router-1",
                            "ip":"http://192.168.0.1",
                            "port":"router-1-p1",
                            "latitude": 19.994326843893894,
                            "longitude": 73.78965468852559
                        },
            "interface":"wlo1",
            "resources": {"delay": 0.5564765526145907, "
               bandwidth": 10, "cost_link": 10}
        }
    ]
}
```

The user needs to specify basic information of a node, such as its ID, name, latitude, and longitude. It is important to highlight that we can add geographically distributed nodes in the SPIDER, since we have information about their latitude and longitude. Additional information is needed for the placement algorithm, such as MTTF (in hours), MTTR (in hours), availability

(in %), cost (in $), and energy consumption (in Watts). We also consider a field metadata, where additional information about the node can be added (e.g. manufacturer). Next, the user must specify the VNFs that are accepted on the node. We use this field in JSON to define which VNFs the node supports for possible placement. The goal is to avoid placing VNFs which are not supported by a node (due to hardware restrictions, for example). Therefore, a user defines a list of VNFs that are supported by the node and this information is considered during the placement by the Agent module. The last information about the node is a list of ports, which defines all network interfaces of a given node.

APPENDIX D – SFC PLACEMENT DECISION BY THE AGENT

Once the Agent defines the best placement strategy for a given request, it creates a JSON representation that describes the obtained placement strategy, as illustrated below:

```
1
2  {
3      "id": 0,
4      "name": "my-sfc",
5      "source": "minion-1",
6      "destination": "master-node",
7      "VNFs": [
8          {
9              "name": "compress-image",
10             "node_name": "minion-1",
11             "replicas": 2,
12             "resources": {
13                 "cpu": 1,
14                 "memory": 1,
15                 "storage": 1
16             }
17         },
18         {
19             "name": "firewall",
20             "node_name": "master-node",
21             "replicas": 1,
22             "resources": {
23                 "cpu": 1,
24                 "memory": 1,
25                 "storage": 1
26             }
27         }
28     ],
```

```
    "flow_entries": [
        {
            "source": "minion-1",
            "destination": "master-node",
            "path": ["minion-1_router-1"]
        }
    ]
}
```

The JSON data contains all information needed for the placement SFC in the infrastructure. We add the name of SFC, the source and destination nodes (this information is defined in the request). Then a list of VNFs is defined. Each element of the list has the name of the VNF, the name of the physical node where the VNF will be placed, the number of replicas of VNF, and the amount of resources that this VNF requires (defined in the VNF template as shown in Table 8). We also specify the list of virtual links that connect nodes where the VNFs are placed. For each virtual link, we specify the source node, the destination node, and the path, which is composed of the links that connect those nodes.