University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization:	Spring semester, 20...... Open / Restricted access
Writer:	………………………………………… (Writer's signature)
Faculty supervisor: External supervisor(s):	
Thesis title:	
Credits (ECTS):	
Key words:	Pages: ………………… + enclosure: ………… Stavanger, ……………….. Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Resource Allocation for Vertical Industries in a Smart City Using Deep Reinforcement Learning

Master's Thesis in Computer Science
by

Brage Riis Gundersen

Internal Supervisors

Ali Gohar

Gianfranco Nencioni

Reviewers

To be decided later

June 8, 2023

Abstract

This Master's thesis addresses the problem of resource allocation in Network Function Virtualization (NFV) within the context of a smart city considering isolation. The objective of the thesis is to minimize the long-term cost and maximize the revenue and acceptance rate of the infrastructure provider. A resource allocation model is proposed, utilizing a state-of-the-art proximal policy optimization (PPO) training algorithm, and its performance is evaluated under decreasing available resources.

In this thesis, we first introduce the smart city and the basic NFV and reinforcement learning (RL) concepts. Then we review the existing literature on resource allocation in NFV. Further, we introduce our problem definition and solution as well as our proposed NFV-compatible isolation levels.

The results we produce demonstrate that the proposed solution exhibits increased performance overall compared to a leading heuristic algorithm called global resource capacity (GRC), especially in terms of the long-term revenue-to-cost ratio (LTRCR) of at the most 20%, and in terms of the 25x reduction in time spent allocating per slice request (SR).

Contents

Chapter 1

Introduction

The infrastructure provider (InP) is one of the major actors in modern networks and is responsible for providing and operating physical infrastructure. In modern cloud-based networks, the infrastructure is centered around data centers (DC) and multi-access edge computing (MEC). Resource allocation entails how these resources are allocated to provide networks to different kinds of areas and industries, such as in smart cities. Optimizations that reduce allocations' overhead and resource usage allow the InP to reduce costs and increase revenue as more tenants can be supported using fewer resources.

In smart cities, the tenants that the InP support are called smart city vertical industries (SCVIs), and they offer unique services that require networks with specific characteristics. Network slices are introduced in 5G networks as the *slicing* of a physical network into several virtual networks, where every slice is a combination of network functions (NFs) and configuration that provides specific characteristics. The tenants will interact with the slice broker (SB) actor, who will interpret their requirements. The SB will then request the resource allocation of network slices from the InP by sending SRs and pay for successful allocations.

Traditional NFs have been implemented using specialized hardware provided by the InP, which limits the flexibility and scalability of the network. To overcome these limitations, NFV has emerged as a technology that enables the virtualization of NFs on top of commodity hardware such as DCs and MECs. Virtualization also allows us to research the need for isolation between these virtual network functions (VNFs).

While the InP's goal is to support as many network slices as possible, isolating these network slices is paramount. This is because the operation of one network slice must not affect others maliciously or accidentally for the InP to be able to provide network slices as specified. Our thesis will distinguish the different ways that isolation can be enabled and provide our isolation levels.

This thesis focuses on the problem of resource allocation in NFV in a smart city scenario while considering isolation. Our goal is to reduce the number of resources allocated by the InP so that the cost of the InP is minimized and that the number of network slices that are successfully allocated is maximized to increase the generated revenue of the InP.

Research on the usefulness of RL in resource allocation is still needed. Deep reinforcement learning (DRL) has recently gotten much attention and has taken advantage of deep neural networks (DNN). Here it is reasonable to think that DRL is a field that can provide performant and scalable solutions to resource allocation due to the general and dynamic nature of the task. It has been shown in previous research that DRL methods are performant in large, complex, and dynamic environments where the state and action space is large.

Only a few papers propose DRL resource allocation solutions, and only some of them use gradient ascent-based methods. Still, none have been published yet where PPO has been used for allocating resources and embedding links while considering isolation. Gradient ascent-based methods optimize the policy parameters by iteratively updating them in the direction of the gradient of the expected reward. PPO is an algorithm in DRL that combines ideas from policy gradient methods and trust region methods to ensure stable and efficient learning. It balances the exploration of new policies with the exploitation of currently known good policies. We will use a state-of-the-art PPO DRL algorithm to tackle resource allocation in an SCVI scenario considering isolation and compare our results with previous research.

1.1 Objectives

1. Conduct a comprehensive review of the existing literature on resource allocation in NFV.
2. Define the levels of isolation required in NFV to ensure the security and reliability of the network.
3. Introduce the problem description.
4. Implement the solution model and test its performance.
5. Evaluate the solution model using appropriate metrics and compare the results to an existing solution.
6. Conclude the findings and discuss the implications of the results, highlighting the strengths and limitations of the proposed solution model.
7. Provide recommendations for future work, including potential extensions to the solution model and areas for further research.

1.2 Contributions

- Resource allocation model trained using the PPO training algorithm.
- Realistic simulation
  - Considers both ingress and egress. Ingress refers to the entry point of data or traffic into the network slice, while egress refers to the exit point.
  - Novel isolation definition that aligns with the principles of network slicing. Network slicing enables the creation of virtual networks tailored to specific requirements. The proposed isolation definition is compatible with NFV and ensures the effective isolation of network slices.
  - Considers various resources (CPU, RAM, storage, and bandwidth) and constraints (isolation, ingress, and egress).
- Demonstrating the competitive performance of the proposed solution compared to an existing approach.

The contributions of this thesis lie in the approach to the VNF-FGE problem while considering isolation and ingress and egress, the competitiveness of this approach, and how it can be used in relevant smart city environments. The thesis gives a systematic and clear definition of isolation levels and, importantly, distinguishes between using containers and virtual machines. The isolation levels presented are ETSI NFV compatible. The thesis implements the solution model and tests its performance under various scenarios, including network loads and resource availability. The evaluation methodology employs appropriate metrics, such as resource utilization and acceptance rate, and compares the results to existing research in the field. The thesis concludes the findings and discusses the implications of the results, highlighting the strengths and limitations of the proposed solution model. Finally, it provides recommendations for future work, including potential extensions to the solution model and areas for further research.

1.3 Outline

The second chapter will provide an overview of the essential components and terms of the thesis. The third chapter will categorize the related work and present the previous methods and their properties.

The fourth chapter will give the problem description and formulation, propose isolation levels in the NFV, and define each level. Further, it will explain the advantages and disadvantages of the different levels and the considerations that went into proposing these levels. How the isolation levels fit into the current NFV architecture will also be discussed. Then the chapter will discuss the chosen DRL approach and scenario, why the approach to the problem differs from previous work and the reasoning behind the related choices. In addition, the chapter will discuss how this thesis contributes to the current field of research.

The fifth chapter documents the simulation environment, related properties, and the evaluation methodology employed. The quantitative results are presented, compared to GRC, and finally analyzed. In the final chapter, the thesis will conclude the presented work and try to give some future directions.

Chapter 2

Background

This chapter provides an overview of essential components and terms related to the thesis topic. We explore fundamental concepts and technologies. Understanding these core elements lays the groundwork for further exploration and analysis of the following chapters. Table 2.1.

Acronym	Definition
NF	Network Function
VNF	Virtual Network Function
VNF-FG	VNF-Forwarding Graph
VNF-FGE	VNF-FG Embedding Problem
VNF-TR	VNF-Traffic Routing Problem
NFV	Network Function Virtualization
NFV MANO	NFV Management and Orchestration
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
VIM	Virtual Infrastructure Manager
EM	Element Manager
OSS/BSS	operational and business services
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DNN	Deep Neural Networks
PPO	Proximal Policy Optimization
SCVI	Smart City Vertical Industries
5G	Fifth-Generation Networking
eMBB	enhanced Mobile Broadband
URLLC	Ultra Reliable Low Latency Communication
IoT	Internet of Things
MIoT	Massive Internet of Things
V2X	Vehicle to Everything
SB	Slice Broker
SR	Slice Request
QoS	Quality of Service
SLA	Service Layer Agreement
InP	Infrastructure Provider
SN	Substrate Network
DC	Data Center
MEC	Mobile Edge Computing
VM	Virtual Machine

TABLE 2.1: Acronyms in Background

2.1 Smart cities

Smart cities are urban areas that leverage data collection and advanced technologies, including 5G networks, to enhance their citizens' quality of life, improve their operations' efficiency, and promote sustainable growth [1].

In a smart city, various critical and non-critical services require different communication technologies for data collection and operation.

2.2 Network Slices

A network slice combines NFs and configuration that provides certain characteristics. In fifth-generation networking (5G) the 3rd Generation Partnership Project [2] have defined four standard network slice types;

- Enhanced Mobile Broadband (eMBB) - Offers enhanced mobile networks over mobile networks currently seen in 4G, especially in terms of speed, capacity, and mobility in three dimensions.
- Ultra Reliable Low Latency Communication (URLLC)- Provides ultra-low latency and high-reliability communication for critical real-time applications.
- Massive Internet of Things (MIoT) - Enables communication between massive amounts of internet of things (IoT)-devices. Low data rates, low power consumption, and massive connectivity characterize MIoT.
- Vehicle to Everything (V2X) - Provides connectivity to and between vehicles in motion and the entities around the vehicles.

2.3 Smart City Vertical Industries

Smart city vertical industries (SCVI) [3] fit well with the network slice paradigm because vertical industries in a smart city require different network slices to be productive. This is because a vertical industry is a segment of the city that might offer a unique service or produce a specific product that requires specific network slice characteristics.

An example of an SCVI requiring specific characteristics in a network slice is the public transportation market, where airports and train stations require eMBB due to the high density of people and their mobile devices. Another example is industries where factories operate machines with real-time monitoring, control, and coordination, leading to improved operational efficiency and safety, which requires URLLC. The smart agriculture industry is an example of an SCVI that requires the MIoT network slice. It relies on many low-power, low-data-rate connections for crop, livestock, and environmental monitoring using sensors. Moreover, healthcare is an example where an ambulance might be required to communicate with every other vehicle and traffic light to reach the destination as fast and safely as possible. This can be accomplished by using the V2X slice type.

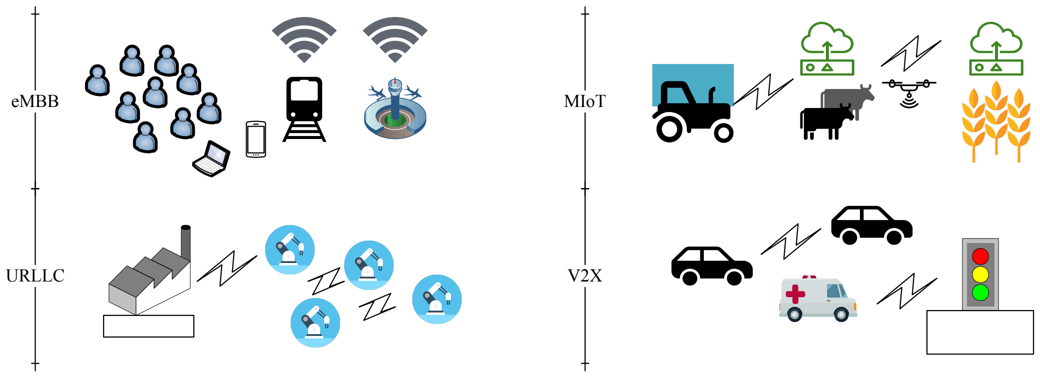

FIGURE 2.1: 5G-Slice Examples

2.4 Isolation

In our case, the performance, security, and reliability of our SCVI cannot be compromised. Each vertical industry might provide critical services, and as such, we must restrict network slices from influencing the operation of each other, both accidentally and maliciously.

Network slice isolation is discussed in several papers among Gonzalez et al. [4], which introduce this concept as the property that enables network slices to operate without the influence of others.

Network slice isolation can be defined as encompassing several levels. Wong et al. [5], for example, proposes eight levels of isolation.

Our isolation levels definition is given in Figure 4.2. In general, the network slice isolation concept can be thought of as the network segmentation of the slices, isolating the traffic of different slices. Then secondly, the virtualization of the hardware resources isolates software components from the underlying hardware and each other.

2.5 Infrastructure Provider

The InP is the entity responsible for providing the underlying physical infrastructure, such as servers, storage, and network resources, to support the allocation of VNFs. InP is typically a service provider or a cloud provider who owns and manages the infrastructure.

As the infrastructure owner, the InP also bears power usage costs, maintenance, and more. The InP is incentivized to keep its infrastructure operational and as well utilized as possible. This is because the revenue generated by the InP comes from selling the creation and operation of network slices, and this revenue can only be generated as long as the infrastructure is operational. Moreover, how well the InP allocates the resources determines how many network slices can fit and be sold.

2.6 Multi-tenancy

In the context of 5G network slicing, multi-tenancy refers to the ability of the network infrastructure to support multiple independent tenants within a shared physical infrastructure [6]. Each tenant can have its network slice, which provides dedicated network resources and services customized to their specific requirements.

Multi-tenancy allows for better resource utilization since resources can be dynamically allocated to different SCVIs based on their requirements, which means that the InP can fit and sell the creation of more network slices. For example, suppose one SCVI requires more processing power while another requires more storage. In that case, the InP can allocate the appropriate resources to each SCVI based on their needs and not reserve unnecessary resources that another SCVI can use.

2.7 Data Centers and Multi-Access Edge Computing

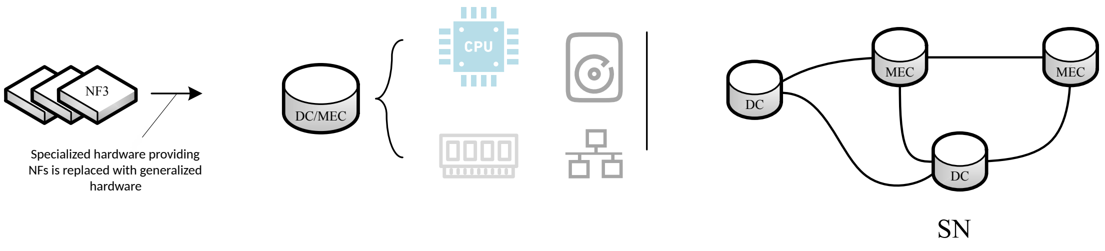
NF3
Specialized hardware providing NFs is replaced with generalized hardware
DC/MEC
CPU
DC
MEC
MEC
DC
SN

FIGURE 2.2: DC and MEC

Modern computers and data centers (DCs) have various hardware components, including general-purpose CPUs, memory, storage, and network cards. These components are designed to be scalable and interchangeable, resulting in cost-effective upgrades, maintenance, and replacements. Routers, firewalls, and load balancers can be replaced or complemented by DCs.

Multi-access edge computing (MEC) has been proposed to improve edge-cloud computing in networks. MEC hosts offer mostly the same features that the DCs offer, but closer to the edge of the network and end-user, but at a smaller scale. This allows the slices to reduce latency and improve QoS as the traffic moves through fewer links and DCs.

2.8 Substrate Network

The underlying infrastructure the InP manage and offer is called the substrate network (SN), where the network slices are allocated. Figure 2.3 shows the SN with its DCs, MECs, and physical links; in this case, there are two InPs that own different parts of the SN.

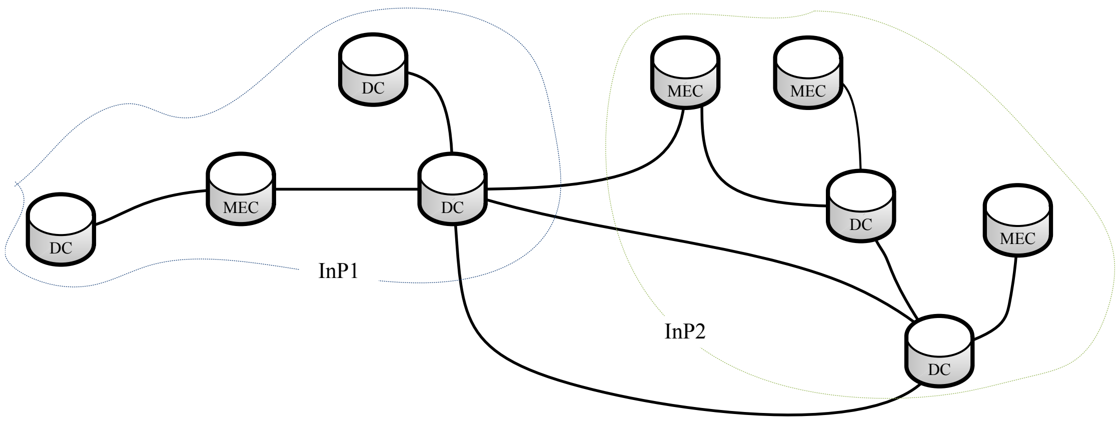

FIGURE 2.3: The SN

2.9 NFV

NFV moves the functionality of the network from specialized hardware into virtual network functions (VNFs) and centralizes the management and provisioning of the VNFs [7]. NFV is one of the ways that network slicing can be accomplished because it enables greater automation and orchestration and allows slices to be allocated and isolated in software and not hardware. The virtualization of the NFs to software-based VNFs reduces the cost of the network slices as software is more easily extensible and is easily placed on more generalized hardware like DCs and MECs.

Two primary virtualization technologies are used for implementing NFV: containerization and virtual machines (VMs). Containerization involves running applications and their dependencies in isolated containers sharing the same operating system kernel. This is an easy way to package and deploy network functions, making it ideal for VNFs. On the other hand, VMs run guest operating systems on top of a hypervisor, which isolates the VMs from the host operating system. While VMs provide greater isolation and security, they are less efficient and more resource-intensive than containers.

In Figure 2.4, we see how the standard ETSI NFV architecture is envisioned. Here the *Virtualization* block can be implemented with containerization or VM. In the NFV architecture, the management and orchestration (NFV MANO) control plane is separated from the data plane. The virtual infrastructure manager (VIM) manages the NFV infrastructure (NFVI). Also, importantly the VNF manager interfaces with the element manager (EM) and the VNF itself to deploy and keep the VNF operational. In our case, the NFV orchestrator (NFVO) works with creating and optimizing the allocation of network slices, while the operational and business services (OSS/BSS) provide an interface for requesting network slices.

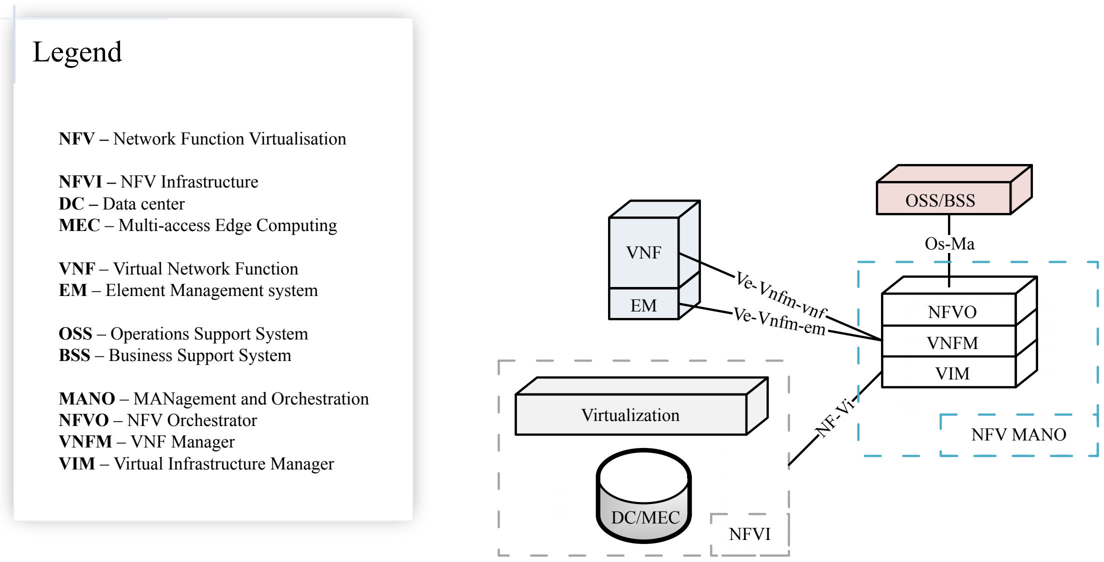
Legend
NFV – Network Function Virtualisation
NFVI – NFV Infrastructure
DC – Data center
MEC – Multi-access Edge Computing
VNF – Virtual Network Function
EM – Element Management system
OSS – Operations Support System
BSS – Business Support System
MANO – MANagement and Orchestration
NFVO – NFV Orchestrator
VNFM – VNF Manager
VIM – Virtual Infrastructure Manager
OSS/BSS
Os-Ma
VNF
EM
Ve-Vnfm-vnf
Ve-Vnfm-em
NFVO
VNFM
VIM
NFV MANO
Virtualization
DC/MEC
NFVI
NF-Vi

FIGURE 2.4: ETSI NFV architecture [8] (release 3)

2.10 Forwarding Graph

The blueprint of a network slice before it is allocated is in our thesis, the VNF forwarding graph (VNF-FG), and it describes the way traffic flows through a chain of VNFs and also contains the logical links(Figure 2.5). In the VNF-FG, an ingress and an egress VNF must exist that define the origin and destination of the traffic, which might then force a minimum amount of links that need to be embedded to complete the forwarding graph.

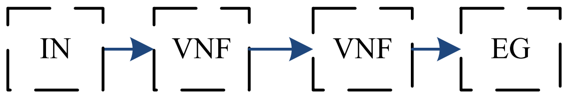

Figure 2.5: The VNF-FG

2.11 Placement Problem

The placement problem refers to placing the VNFs in the VNF-FG onto the SN and is referred to as VNF-P. The VNF-P affects the processing power and traffic used and, in turn, the cost of the service. Every placement incurs a cost, as the infrastructure to run the VNF is bought from the InP, and the cost of deploying and starting the VNF comes in addition to the resource and traffic used when running the VNF.

Many placements on a single SN node might benefit from lower overall costs due to sharing specific software components, i.e., when grouped as containers, and due to reduced network usage, as traffic must pass through fewer links. However, many placements on a single node will exceed the different resource capacities and overload and fail the node. In contrast, an entirely spread-out placement on many nodes is costly as traffic must traverse many links and will lead to poor utilization.

2.12 Forwarding Graph Embedding Problem

Individual VNFs cannot be applied interchangeably throughout the VNF-FG. The traffic must be routed in a specific order, often denoted by the traffic routing problem (VNF-TR). For example, if a firewall-VNF is present in the VNF-FG, it may be applied first to ensure the subsequent VNFs receive the firewall-filtered traffic [9]. When the VNF-P and T problems are combined, we call it the forwarding graph embedding problem (VNF-FGE)[7].

In figure Figure 2.6, we see how a VNF-FG might be allocated on the SN by the InP. The green straight arrows show the links of the VNF-FG, while the stippled ones show how the InP chose to embed the links in the SN, which implements the traffic routing of the VNF-FG. Following the NFV structure, here the VNFs are placed inside containers.

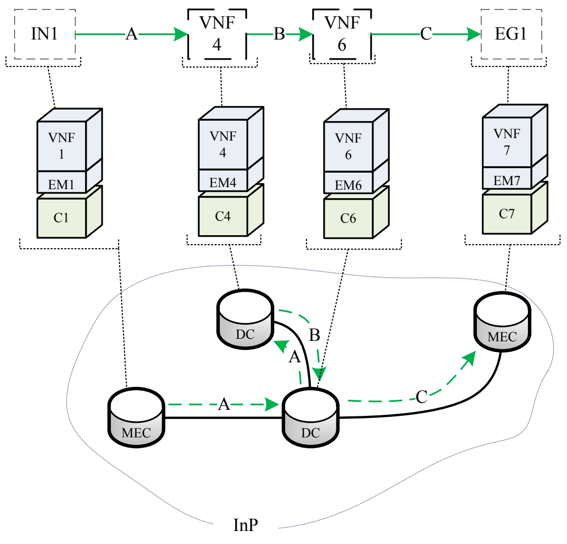

FIGURE 2.6: Placement and embedding of VNF-FG

2.13 Slice Broker

The slice broker (SB) is the entity that buys the allocation of network slices from the InP. The SCVIs interact with the SB, which interprets requirements and creates an SR that is forwarded to the InP. We use an *online* environment where the InP must allocate SRs as they arrive from the SB.

2.14 SRs

The SRs define the required VNF-FG, resources, and service level agreement (SLA) that the SCVI expects to use. The SLA specifies various performance metrics, availability targets, scalability requirements, and other QoS parameters that the InP commits to meeting. These metrics can include parameters like response time, throughput, latency, packet loss, availability percentage, and other relevant indicators.

Furthermore, the SLA also outlines the penalties or remedies that may be applicable if the service provider fails to meet the agreed-upon service level objectives. These penalties or remedies could involve financial compensation, service credits, additional support, or other remedies as agreed upon in the SLA. The VNF-FG describes the logical topology of the VNFs, their required interconnections, and, importantly, the ordering of the VNFs. The SR specifies the required resources, such as CPU and RAM, and the link bandwidth and isolation requirements.

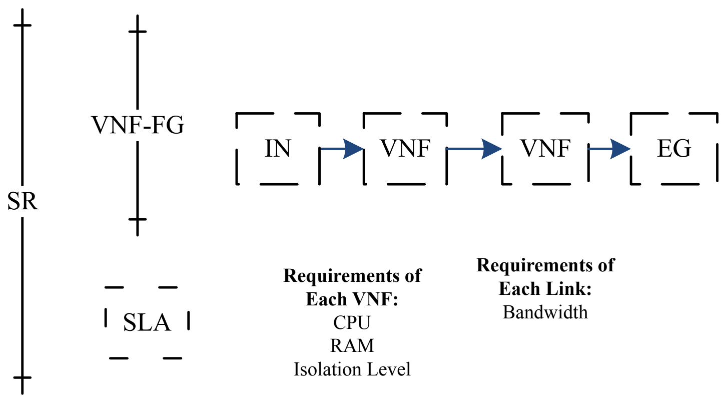

FIGURE 2.7: SR

2.15 Roles

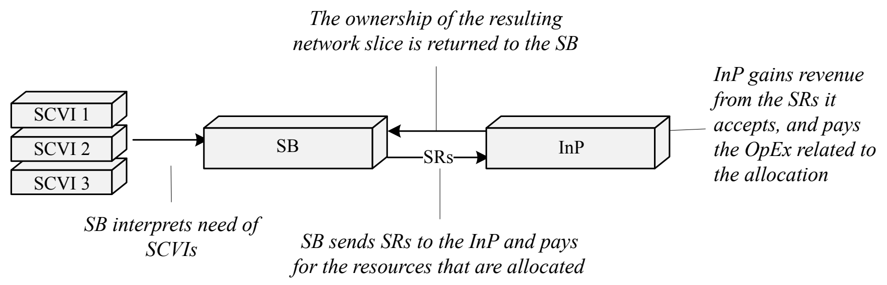
The ownership of the resulting network slice is returned to the SB
InP gains revenue from the SRs it accepts, and pays the OpEx related to the allocation
SCVI 1
SCVI 2
SCVI 3
SB
SRs
InP
SB interprets need of SCVIs
SB sends SRs to the InP and pays for the resources that are allocated

Figure 2.8: Relationship between the SCVI, SB, and InP

We see the relationship between the SCVI, SB, and InP in Figure 2.8. The InP manages the SN and its resources and allocates the SRs received from the SB. The SB is responsible for creating SRs by interpreting the needs of SCVIs. The SR requirements define the VNF-FG and the necessary service resources. The InP generates revenue from selling the allocations to the SB and pays the OpEx related to the allocation, which means that the InP has the incentive to maximize the number of allocated network slices and minimize the number of resources used.

In Figure 2.9, an overviewing model of our thesis problem is given. From the left side, we see the SB interpreting the needs of the SCVI shown with a downward arrow. The SB then interfaces with the InP's OSS/BSS layer and sends the SRs. The InPs NFV MANO stack is then responsible for allocating the requested network slices.

In this example, the three allocated network slices are shown represented by their VNF-FG at the top. The way each VNF has been placed is given by the line to the middle "NFV" layer. At the middle layer, either a container VM is used, and we will get back to the reasoning behind this in chapter 4.

At the bottom, the SN is shown along with its DCs and MECs, and the embedded links are indicated with the stippled arrows.

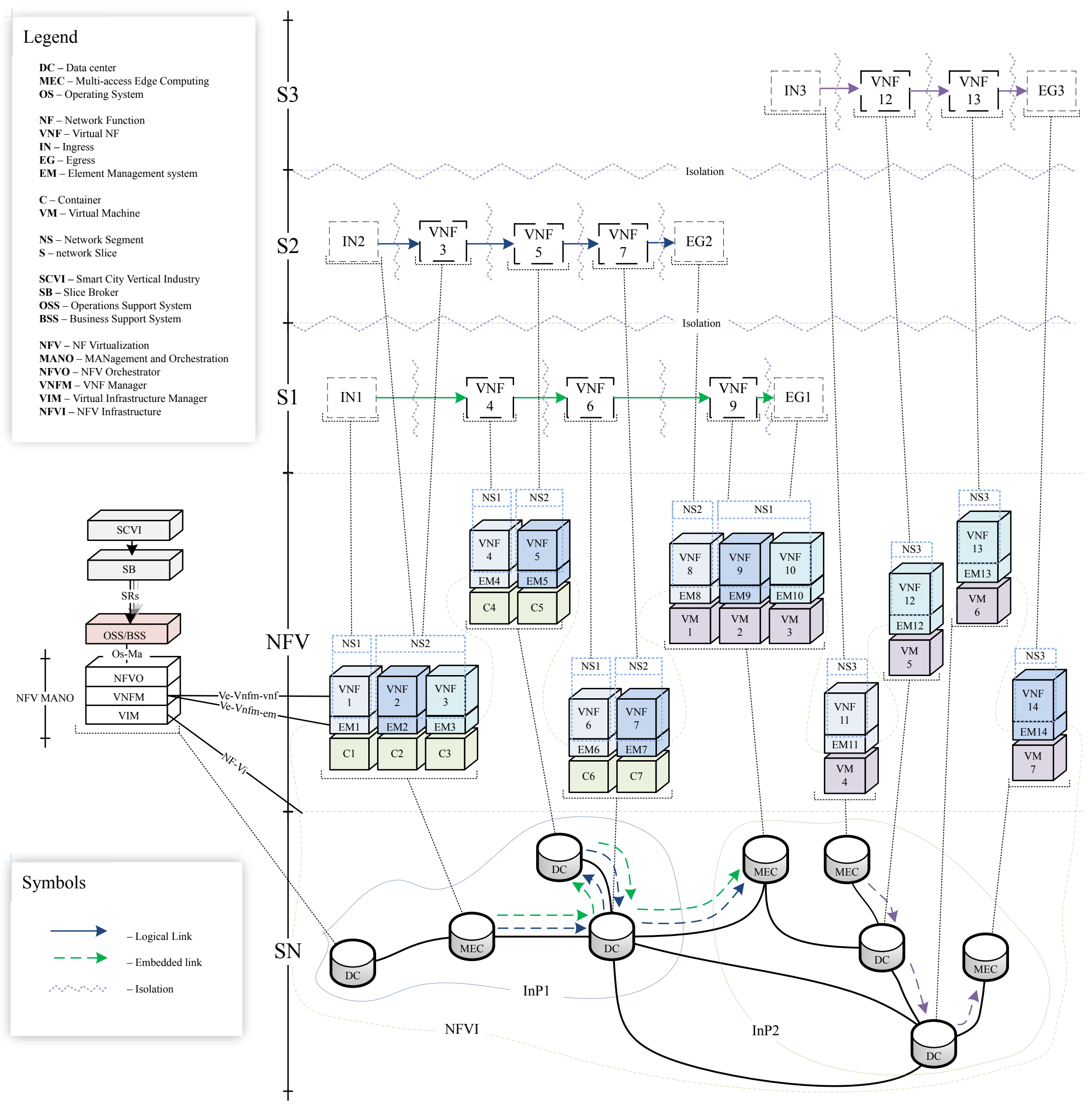

FIGURE 2.9: The Whole Picture

2.16 RL

Acronym	Definition
RL	Reinforcement Learning (An agent performing an action based on a policy receives a reward, which adjusts the policy of the agent)
DRL	Deep Reinforcement Learning (Deep neural network agent)
MC	Markov Chain (stochastic state model)
MRP	Markov Reward Process (Extension to MC adding rewards for good states)
MDP	Markov Decision Process (Extension to MRP adding decision-making to agents - policy and action)
STM	State Transition Matrix (Represents the possible transitions from one state to another with their associated probability)
Episode	Collection of samples of the state
Horizon	The maximum sample count of an episode
Epoch	Set of episodes after which the policy is updated
SVF	State Value Function (Value based on the discounted value of entering a state)
OVF	Optimal Value Function
OSVF	Optimal State Value Function
O(S)AVF	Optimal (State) Action Value Function ($q_*(s,a)$)
AVF	Action Value Function
SARSA	State–action–reward–state–action
On-policy	Update policy based on actions taken
Bootstrapping	An estimate of state or action value is updated based on experience

Table 2.2: Markov Chain Acronyms

Symbol	Definition
S	Set of States
$s \in S$	Particular state
A	Set of Actions
$a \in A$	Particular action
t	Time Step
r	Reward
γ	Discounting Factor
π	Policy
R_t	Discounted Reward
v_π	SVF
q_π	AVF

Table 2.3: Notation Used

2.16.1 MC

To understand RL properly, we must first look at the basic structure of a Markov chain (MC). An MC is a state model where the next state is given a stochastic probability. An important distinction here is that the likelihood in this chain is known and not "random", i.e., the following possible states are known as well as their transition probability; of course, the sum of probabilities must be one.

Every state must be unique in an MC, and a new state is transitioned for every step when a discrete time space is used. However, the probability of the following state can not depend on the previous state; this we call the Markov property (MP). The state transition matrix (STM) shows the possible transitions from one state to another with the associated probabilities while adhering to the MP. In addition, the STM has mathematical properties which make it useful for computations of the MC state.

2.16.2 MRP

The Markov reward process (MRP) adds state rewards to the MC, meaning transitioning into a state makes the agent receive an immediate reward. This reward will represent behavior that is defined as good. An episode is a collection of measured samples of our MC.

The return in a Markov Reward Process (MRP) is calculated as the sum of rewards in an episode, but it is useful to discount the reward received as we step forward in time.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

In the reward function (Equation 2.1)[10], R_t is the return at time step t, γ is the discount factor between zero and one, and r_{t+k} is the reward at time step $t + k$. A discount factor close to one will adjust the model towards receiving the most reward over time. However, when the discount factor is adjusted the other way, the model will prioritize the immediate rewards higher.

In the real world, reward-discounting can be analogous to the economy, where there is an incentive to spend or reward immediately due to the inflation of currencies, where the saved currency becomes less valuable with time. This is also very useful in mathematics and informatics because the sum of samples can be easily limited to achieve some specified accuracy.

We get the expected return if we average the discounted return in a Markov Reward Process (MRP) over a collection of episodes called an epoch. The state value function

(SVF) describes the long-term return of being in a certain state s[10].

$$v_\pi(s) = \mathbb{E}\left[R_t \mid s_t = s\right] \tag{2.2}$$

In the SVF, $v_\pi(s)$ is the expected return from state s, $\mathbb{E}$ is the expectation under policy π, and R is the reward function.

Using the Bellman equation, we can decompose the SVF into the immediate reward r_{t+1} and the discounted reward $\gamma v_\pi(s_{t+1})$ of the successor state s_{t+1}. This is useful as the Bellman equation can be solved, although the $O(n^3)$ time complexity means that solving it might only be useful for small perfect information models.

The Bellman equation for the SVF can be written as[10]:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.3}$$

where $\pi(a|s)$ is the probability of taking action a in state s, $p(s', r|s, a)$ is the probability of transitioning to state s' and receiving reward r after taking action a in state s.

2.16.3 MDP

The Markov decision process (MDP) adds decision-making to the model; the central concept is that the agent operates based on a policy that dictates which action the agent should take each episode. In the MDP, probability does not directly decide the state transition; instead, the agent might behave entirely deterministically using a policy that chooses only the action that will lead to the most valuable state or with a probabilistic policy where some distribution chooses the action to take.

The MDP consists of a set of states S where each state $s \in S$, a set of actions A where each action $a \in A$, and a transition function $P(s'|s, a)$ that gives the probability of reaching state s' after taking action a in state s, and a reward function $R(s, a, s')$ that gives the immediate reward for this transition.

The action value function describes (AVF) the value of taking the action a in the state s adhering to policy π[10].

$$q_\pi(s, a) = \mathbb{E}\left[R_t \mid s_t = s, a_t = a\right] \tag{2.4}$$

The policy π maps each state to an action. The goal of an agent is to find an optimal policy π_* that maximizes the expected return, which is the sum of discounted rewards over time. The value function $v_\pi(s)$ measures the expected return from state s under policy π, and the action-value function $q_\pi(s,a)$ measures the expected return from taking action a in state s and then following policy π.

The optimal policy π_* is better than or equal to all other policies[10],

$$\pi_* \geq \pi, \forall \pi \tag{2.5}$$

The π_* achieves the optimal state value function v_* (OSVF) and optimal action value function q_* (OAVF).

$$\begin{aligned} v_{\pi_*}(s) &= v_*(s), \\ q_{\pi_*}(s,a) &= q_*(s,a) \end{aligned} \tag{2.6}$$

The OSVF $v_*(s)$ tells us the maximum SVF[10].

$$v_*(s) = \max_\pi v_\pi(s) = \max_a q_{\pi_*}(s,a) \tag{2.7}$$

The OAVF is the maximum SAVF[10].

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.8}$$

The Bellman equation for the AVF can be written as[10]:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')] \tag{2.9}$$

where $\pi(a|s)$ is the probability of taking action a in state s, $p(s',r|s,a)$ is the probability of transitioning to state s' and receiving reward r after taking action a in state s. And for the OAVF, the Bellman equation is[10]:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{2.10}$$

Model-based or *model-free* terms are used in the RL field to describe how the model perceives the environment states. Model-based methods give the agent perfect information about the environment. Many games provide the player with perfect information, including chess, and these games might be solved using model-based methods. Still, model-based methods become infeasible to compute when the state and action space are very large. Model-free methods make more sense in a more general environment as the methods can perceive parts of the environment and learn from experience rather than perfect information. This is also more analogous to human learning, where humans do not learn with perfect information about the environment but rather by perceiving the environment through senses, which are not entirely accurate.

2.16.4 Summary

Li [10] gives a great explanation and summary of the current different RL methods: dynamic programming, monte carlo, temporal difference learning, policy optimization, and DRL. The DRL and policy optimization methods are the most relevant for this thesis. Here Li [10] notes that compared to temporal difference learning policy optimization usually have better convergence properties. However, a big issue with these methods is that they might converge to local optimums and not global optimums, which in that case, means that the agent's learned policy may not reach close to the optimal policy in a reasonable timeframe.

2.16.5 DRL

In deep RL, deepness refers to the presence of multiple layers of nodes in the neural network. Deep RL methods have been shown to give better results than shallow models(see Table 3.3 paper [11] where Q-Learning loses to DRL methods).

Sun et al. [12] mentions current research on DRL and its problems. They note the curse of dimensionality as one of these problems. The curse refers to the nature of many-dimensional models and the exponential data required to use them when the dimensionality increases. This curse can be understood in this context as the problem of increased time and data spent when using increasingly complex models and is certainly a much-researched subject within DRL.

[12] also states that the "low convergence speed, and exploration and exploitation balance" as additional concerns in DRL, and here the low convergence speed is related to the implementation speed of the thesis, as the convergence can be practically understood as the time where there is little or nothing to gain from further learning, here there is significant variation between the different algorithms.

Exploration and exploitation balance goes into the same convergence concept but describes how the RL algorithm will learn and how greatly the algorithm will prioritize exploring new actions that might provide a different reward over exploiting the actions known to give the greatest reward. This concept is important, and we must ensure a good balance such that PPO converges to the optimal policy.

2.16.6 Policy Gradient Methods

For our thesis, policy gradient methods (PGMs) are the most important, Weng [13] summarizes a list of the current different PGMs. Considering the curse of dimensionality, it is clear that when the state space increases, the usefulness of model-based methods decreases because of the time complexity.

An advantage of the gradient methods is that we do not have to compute the entire AVF of the model; instead, using partial derivatives, we can approximate the SVF by using the sampled experience. We can instead sample episodes and sum the AVFs, which will approximate the actual AVF of the model, which we can then use to adjust our policy in the direction of the maximum SAVF (OAVF).

PGMs tend to converge faster than value-based methods [13]. Again the optimal policy will be to receive the maximum rewards, but the reward function(Equation 2.11) is different from in MRPs.

$$J(\theta) = \sum_{s \in S} d_\pi(s) v_\pi(s) = \sum_{a \in A} \pi_\theta(a|s) q_\pi(s, a) \tag{2.11}$$

Here the reward function $J(\theta)$ represents the expected discounted reward when following policy π.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s) \hat{A}_\pi(s, a)] \tag{2.12}$$

Further, the gradient can be represented as in Equation 2.12 where $\nabla_\theta J(\theta)$ θ is the expected reward gradient, which defines the reward that the agent will receive following policy $\pi_\theta(a|s)$ beginning from the current state. The first term gives the log probability of taking action a in state s, denoted as $\nabla_\theta \ln \pi_\theta(a|s)$. The policy gradient describes whether a local minimum has been achieved.

The second term describes the advantage function $\hat{A}_\pi(s, a)$, which is the difference between the AVF and SVF. This function represents the advantage of taking action a in state s compared to the average action under the policy $\pi_\theta(a|s)$. And so, it gives the value of the action relative to the average action for the current state.

Finally, the update rule can be written as:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k), \tag{2.13}$$

After computing the gradient of the expected return with respect to the policy parameters, we use Equation 2.13 to update the policy parameters iteratively during the learning process.

In this Equation 2.13, θ_k denotes the current values of the policy parameters at the k-th iteration, and θ_{k+1} denotes the updated values of the policy parameters at the $(k+1)$-th iteration. The update is performed by taking a step in the direction of the gradient, scaled by a learning rate α, which is a hyperparameter that determines the step size.

Chapter 3

Related Work

In this chapter, we present an overview of earlier research and findings, offering insights into the existing research in the field. Examining past endeavors gives us valuable insights into prevailing trends, challenges, and potential solutions researchers have explored.

3.1 NFV

Acronym	Definition
NF	Network Function
NFV	Network Function Virtualization
VNF	Virtual Network Function
VNF-FG	Virtual Network Function Forwarding Graph
VNF-P	Virtual Network Function Placement
VNF-TR	Virtual Network Function Traffic Routing
VNF-FGE	Virtual Network Function - Forwarding Graph Embedding problem

Table 3.1: NFV Acronyms

Sun et al. [12] shows a lot of the NFV-related acronyms used in their survey, but note that these acronyms and descriptions will vary slightly from other surveys like [7].

Understanding the evolution of network functions is important to the thesis, and [12] does a good job of describing the lineage of research around NFs. In this context, it is important to distinguish between the NFs, traditionally provided physically in specialized hardware, and the VNFs, provided entirely virtually in software. This evolution from NF to VNFs has reduced the cost of network slices and improved scalability, but there are more resources to be saved when solving the VNF-FGE problems; the first of these problems to solve is the VNF-P problem.

The routing model in [12] assumes that the service request that is made contains an ordering of VNF that is strict and that there is no concurrency in the way traffic flows through the VNF-FG. Therefore traffic must flow sequentially, even though there might exist VNFs that can be interchangeably computed in order and optimizations that take advantage of this.

Mirjalily and Luo [9] provides an overview of service function chaining and some good diagrams for understanding the standard ETSI NFV architecture. In this regard, it shows how the different parts of the architecture parts will interact, as well as some of the application scenarios.

Sun et al. [12] defines the different placement problem variant models: chaining, placing, embedding, and routing. The placing model is the most basic in terms of complexity, as it focuses on optimizing the VNF locations in the SN alone. We will, however, separate the placing and routing terms and tackle placing, embedding, and routing all at once, which we call the VNF-FGE problem.

As Sun et al. [12] notes, the placement problem research shows that no single algorithm fits every scenario, even when looking at only the P problem and not adding TR complexity. However, this is an area where we might see learning-based algorithms perform well because the unclear and complex scenarios suit a well-learned, complex, and generalist solution. RL and DRL solutions might become general enough to fit many scenarios.

Sun et al. [12] provides comprehensive tables encompassing papers describing parts of the VNF-FGE problem categorizing them in the cloud, backbone, mobile, IoT, and heterogeneous networks while describing the objectives, constraints, and algorithms used. The five network types will be important in our smart city environment, and the tables give good oversight into the five classes.

3.2 Isolation

As mentioned in the introduction and background, a table and diagram that introduce our isolation definition are provided in Figure 4.2 and Figure 4.3. The subject of isolation in NFV is not new, and as such, many papers are discussing the matter. Gonzalez et al. [4] provides a good overview of the current state of this isolation, and Wong et al. [5] goes into detail about the new concept of network slice isolation which is interesting for the thesis, as the different slices of our smart city network mustn't influence each other as mentioned in the introduction.

What is missing from current research is the combination of 5G network slice isolation, smart cities, and VNF-FGE with DRL; here, we can only find [3]. Gohar [3] provides a DRL solution to the VNF-FGE problem that beats some of the competition; we will try to improve this result by using a different DRL algorithm. The paper [3] uses an integer to represent the isolation level of a computational platform (CP), which we will also do. However, our isolation definition is different and takes a more systemic approach.

3.3 Solutions to VNF-FGE

3.4 Exact

Paper	Type	Algorithm	SN	Metrics	Performance
[14]	Exact	LBBD	Random	Runtime, SN Utilization	ILP < LBBD
[15]	Exact, Approximating, Heuristic	ILP, LP, HA	Random	Runtime, Cost	LP < ILP < HA
[16]	Meta Heuristic	MHA	Random Fat-Tree	Packet Loss, SN Utilization, Path Length	HA(CLBP) < MHA
[3]	DRL	DRL	Random	Acceptance Rate, Long Time Cost Ratio	MCT < DRL
[11]	DRL	A-DDPG	Random	Cost	Q-Learning < NFV-Deep < DDPG < A-DDPG
[17]	DRL	MADRL	Multiple	Delay, Cost, Acceptance Ratio, Throughput	HA < MADRL

Table 3.2: Simulation parameters - General

Paper	Algorithm	HW	Links	Nodes	SRs
[14]	LBBD	CPU or RAM	BW	30, 36, 40, 60	5-50
[15]	ILP, LP, HA	CPU	BW	25, 50, 75, 100, 125, 150, 175, 200	?
[16]	MHA	CPU	BW	250	500
[3]	DRL	CPU, RAM, Storage, Isolation	BW	110	2000
[11]	A-DDPG	CPU, Latency	BW, Latency	10, 20, 30, 40, 50	10-100
[17]	MADRL	CPU, RAM	BW	37, 65, 100, 225	up to 30000

Table 3.3: Simulation parameters - Specifics

The exact solutions optimally solve the VNF-FGE problem, finding the best placement and routing possible for each SR. So in the case of exact solutions, the problem is not to find a way to give the exact solution but to find one using the least amount of computational resources; as the VNF-P and T problems are themselves NP-hard, there can not be a solution that is computed in polynomial time that is $O(n^k)$. Often, solutions use Integer Linear Programming (ILP) and Mixed Integer Programming (MIP).

Ayoubi et al. [14] recently proposed an exact solution to solving the VNF-FGE problem using a Logic-Based Benders Decomposition (LBBD) Approach. Benders decomposition is a mathematical method that simplifies linear programming problems; in short, the technique allows for creating a smaller sub-problem which will, when checked, be the solution or reduce the original problem.

In [14], the physical network is represented as an undirected graph consisting of nodes and logical links with a finite computing and bandwidth capacity. This model is similar to many others solving the P and T problems, and we will use a similar setup. However, we will also model storage and the level of isolation of each node like in [3]. What Ayoubi et al. [14] finds is that their method performs better overall compared to an ILP solution up to 700 times in fact in terms of processing time, and compared to a heuristic k-shortest path, their solution gives better acceptance rate and processing time.

Xu et al. [15] proposes using a more traditional ILP-based exact solution than [14], and the authors state that this exact solution is not scaleable to large problems due to the fact the placement problem NP-hard, but [14] show that an LBBD solution can be improved.

3.5 Meta Heuristic

Meta-heuristic algorithms work by iteratively evaluating solutions and then modifying them to improve their quality. These modifications are guided by rules or heuristics that determine which solutions to accept or reject based on some evaluation criteria.

Zhou et al. [16] distinguishes between two types of scalability, vertical and horizontal. Vertical scaling refers to VNFs being duplicated or otherwise allowed to consume more of the resources of a single network node, while horizontal scaling is when the VNFs are instead placed across the network. This distinction is important since scaling up vertically is cheaper and faster, as mentioned in the introduction, and is a concern that we will consider. But as Zhou et al. [16] notes, the resilience to burst traffic depends on the idle resources on the node in question, and many times, horizontal placement must be used even though vertical placement might be preferred.

Zhou et al. [16] also discovers that prioritizing burst resilience means that placing the VNFs from the same SFC (Service Function Chain) onto the same node is worse than placing VNFs from different SFCs onto the same hardware. This is because the probability that the VNF will receive burst traffic is related to SFC burst probability. Therefore it follows that if one VNF in the SFC gets burst traffic, so will other VNFs too, and this combined need for vertical scaling will more likely overflow into forced horizontal scaling when it first happens.

However, this burst resilience might come at the cost of isolation, which [16] has not discussed. Zhou et al. [16] propose their own binary search and bisection algorithm and compares it to a cluster-based heuristic algorithm and finds that their algorithm performs better when considering packet-loss in a "bursty" environment

3.6 RL

```
(("reinforcement learning" AND NOT "deep reinforcement learning")
AND "VNF" AND NOT "survey" AND NOT "arxiv")
AND ("VNF-FGE" OR "VNF-FG embedding"
OR "VNF*P*R" OR
(("placement" OR "deployment") AND ("routing" OR "scheduling")))
```

FIGURE 3.1: Dimensions RL only search query

Using an advanced search query (Figure 3.1) in the search engine Dimensions, we find published articles using only RL methods and not DRL. Here we only want to find published and peer-reviewed articles that solve the VNF-FGE problem. From the query used on the 10th of February 2023, we can gather that this specific area of research has only been active since 2020, with only seven published articles returned.

Cai et al. [18] considers a parallelized SFC (PSFC) where traffic can be routed in the VNF-FG such that the ordering is not forced. Two VNFs can be computed in parallel if none are dependent on the other, and the VNFs are not writing and reading the packets passing through them, similar to the concepts and problems of multi-threading in CPUs. This can be useful in URLLC as it reduces the latency of the SFC.

However, the parallelization induces a higher cost of the SFC due to the memory used to hold the packets of the node that finishes first and, eventually, the computation used to merge the packets of both nodes at the dependent node. It is possible that a DRL method could take advantage of PSFC, but then the complexity of the model would be increased as what VNFs could be parallelized would have defined, as well as the wanted trade-off between cost and latency based on the slice type.

Cai et al. [18] use a Q-learning-based RL method to optimize the P and R problems. Q-learning has become a standard RL that finds the optimal policy that gives the OSVF. Deep Q-learning improves on this method, but PPO methods are known to out-compete the deep Q-learning methods. Cai et al. [18] show that their algorithm performed best when the learning rate was set to 0.6, but as we are using a DRL method which is quite different, it is unlikely that this is the best learning rate for our purposes.

3.7 DRL

```
(("DRL" OR "deep RL" OR "deep reinforcement learning")
AND "VNF" AND NOT "survey" AND NOT "arxiv")
AND ("VNF-FGE" OR "VNF-FG embedding"
OR "VNF*P*R" OR
(("placement" OR "deployment") AND ("routing" OR "scheduling")))
```

FIGURE 3.2: Dimensions DRL search query

Paper	Algorithm	Hidden Layers	Learning Rate	Discount	Episodes	Batch Size	Epochs
[3]	DRL	?	0.005	0.998	100	100	?
[11]	A-DDPG	3	0.01	0.8	3000	64	?
[17]	MADRL	?	0.05, 0.01	0.99	500	256	10000

TABLE 3.4: DRL Hyperparameters

Similarly to in Figure 3.1, we find published articles using DRL methods with Figure 3.2. From the query used on the 10^{th} of February 2023, again, we see that this specific area of research has only been active since 2020, with only ten published articles returned. Considering the significant development in RL and DRL in general in the last few years, it is reasonable to think that more will be published in the next years.

Gohar [3] uses a DRL method to optimize the network broker's cost but also considers the isolation aspect of NFV. The performance of the DRL solution is found to be 12% better than other algorithms, including monte-carlo tree search, which is a classic RL method. The paper uses a four-layer neural network as the learning agent to create the policy network, similar to what we will present in this thesis. Gohar [3] uses a policy gradient ascent-based algorithm using the common softmax method.

He et al. [11] uses an Attention mechanism-based deep deterministic policy gradient (ADDP) method to train their agent. Their paper seeks to solve the VNF-FGE like ours and notably considers cost and QoS. He et al. [11] takes care not only to consider the cost of the "NFV-operator", or as we have called it the SB, but also the network revenue and cost. Interestingly the paper uses an attention model in the learning process to focus on the learning to more important nodes to increase the learning efficiency. A drawback of [11] is that the paper only considers the processing capacity of the nodes and delay and link capacity and delay, not the storage capacity or isolation level.

He et al. [11] find that they achieve the best performance with a learning rate of 0.01. Further, they compare their A-DDPG algorithm to DDPG, NFVdeep [26], and Q-learning methods. They find that the A-DDPG method returns the greatest reward, even more than DDPG, while also taking less time to compute.

He et al. [11] also find that the performance of the A-DDPG fluctuates less than the others during the learning process but that they have to ensure the right batch size, which is the number of samples processed before the weights are updated. This is especially important as small batch sizes tend to increase the learning rate. However, too small a size will cause the loss function to fluctuate greatly as the gradient estimate is inaccurate. But again, a too-large batch size becomes too memory intensive and slow and worse, leading to reaching a local optimum, not the global.

Very recently Wang et al. [17] showed how a multi-agent DRL could be used to solve the VNF-FGE. Wang et al. [17] claim to be the first to use a DRL solving the VNF-FGE while also considering a dynamic topology, that is, a network topology where new nodes and links might become a part of the network while old ones might stop working and disappear. This is important as the solution will allow the same trained network to be applied to multiple different topologies without incurring the cost of training anew. As the name suggests, the method is based on training multiple agents, but the training is conducted using the DDPG method. Wang et al. [17] does model memory capacity in addition to processing and link capacity.

Chapter 4

Problem Model

4.1 Description

The SRs that are simulated are received in an online manner and are thus not known to the agent beforehand. The ingress and egress nodes, as well as DC and MEC nodes, are modeled, giving a more realistic simulation as the origin and destination of the VNF-FGs are restricted to nodes close to the user.

Due to our novel approach to the isolation levels in NFV, we will give comprehensive diagrams, definitions, explanations, and considerations of isolation later. For the simulation, the isolation will be described as a constraint in the SR and a property of the nodes in the SN.

4.1.1 Actors

There are several important parts to the simulation model; firstly, there are two actors: the SB and the other the InP. The InP is responsible for managing the SN and its resources and allocating SRs that it receives from the SB. Conversely, the SB is responsible for buying allocations from the InP by sending SRs.

SB

The SB interprets the needs of SCVI tenants and creates SRs. Then the SB pays for the resources defined in the SRs, which are sent to the InP if accepted.

The SR (Figure 2.7) requirements define the VNF-FG, which contains the ingress and egress and the required resources needed to provide the service. It is up to the SB to interpret the request, allocate the resources needed, and do so efficiently so that more SRs can be allocated. If the SB allocates an SR, we say that the SR is accepted.

InP

In our simulation model, the InP owns the DC, MEC, and links that make up the SN and allocates resources as efficiently as possible according to the received SRs. To achieve this goal, the InP needs to manage the SN's resources, including processing power, memory, and storage capacity, and ensure that they are allocated in a way that meets the SRs requirements while considering the isolation levels. The InP sells allocations to the SB and pays for the OpEx.

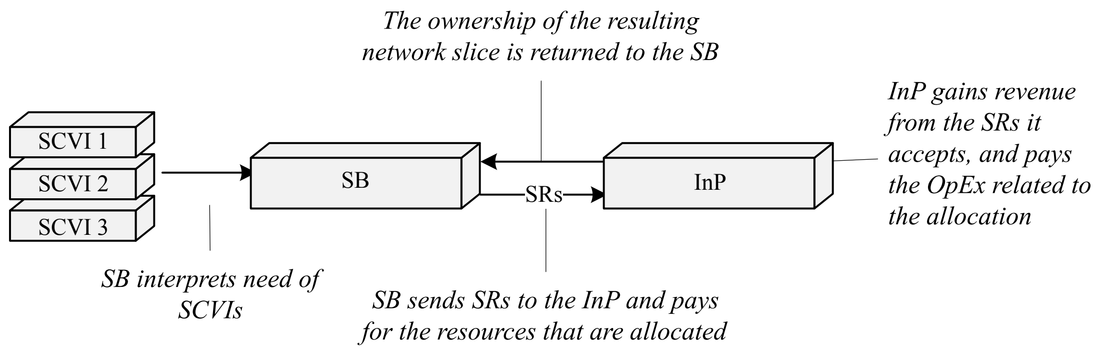
The ownership of the resulting network slice is returned to the SB
InP gains revenue from the SRs it accepts, and pays the OpEx related to the allocation
SCVI 1
SCVI 2
SCVI 3
SB
SRs
InP
SB interprets need of SCVIs
SB sends SRs to the InP and pays for the resources that are allocated

FIGURE 4.1: Relationship between the SCVI, SB, and InP

Due to considering the VNF-FGE, SRs are only accepted when the InP completes both the allocation of resources and the embedding of links in the SN. The embedding of the links is based on the VNF-FG given in the SR and is the corresponding path that connects the allocated resources in the SN such that the network slice satisfies the requirements of the SR.

The Waxman network topology is chosen for the SN as it provides a natural structure seen in many types of networks, which is more realistic than a random topology. The SN consists of DCs and MECs, which are connected with links.

4.1.2 Objectives

The InP is tasked with allocating the received SRs. The InP's allocation objectives are:

- Maximizing long-term Profit of the InP
  - *Maximizing long term Revenue* – Generated from accepting SRs.
    - *Maximizing acceptance rate* – By accepting as many SRs as possible.
  - *Minimizing long term Cost* – Incurred from allocating resources.

The revenue gained by the InP is defined by the number of resources requested in the SRs that the InP manages to accept. The cost, however, is measured by the resources allocated by the InP consistent with OpEx. Thus we are, in effect, maximizing the profit of the InP.

4.1.3 Assumptions

In our model, we make the following assumptions:

- *Online* – SRs are received at any time.
  - *Lifetime* – SRs have any lifetime. The lifetime of an SR is defined by its arrival and departure time. Therefore the InP will not only allocate but also deallocate the resources.
- *Acceptance* – SRs are accepted or rejected. The acceptance is based on whether or not the InP manages to allocate the SB.

Ingress and Egress

The ingress and egress must be allocated on a MEC in the SN. This constraint complicates the InPs' task as it must now also ensure that the MEC is available to facilitate further allocation in the future.

In addition, the allocation becomes dependent on the location of the MEC as they define the available ingress and egress points of the service, which can force the use of more links if the only possibility left is to allocate ingress and egress on different MECs.

Consolidation

A one-to-one relationship between the VNFs and DC or MEC to restrict the influence of one slice to another, or to reduce the impact of a single VNF failure that will fail multiple VNFs because of failing the underlying DC or MEC. However, when using the proposed isolation levels we can guarantee the isolation of all slices even if some VNFs are consolidated on the same DC or MEC. Consolidation means that VNFs can be allocated on the same DC or MEC. Due to the isolation being enabled by virtualization technology, we can also guarantee that a single VNF will not fail others and can be quickly reinstated.

Domain	Advantages	Disadvantages
Resource Utilization	– Efficient use of SN resources reduces CapEx and OpEx. – Less links and traffic required reduces OpEx. – Greater flexibility.	– Increased complexity of allocation.
Performance	– Lower latency due to closer proximity of VNFs.	– Greater risk of performance degradation due to resource contention.
Management	– Simplified deployment and operation.	– Increased complexity in troubleshooting and diagnosis due to shared resources.
Reliability	– Virtualization enables faster restoration of VNFs in case of VNF failure	– Failure of a single DC MEC fails multiple VNFs.

Table 4.1: Advantages and disadvantages of consolidation.

4.1.4 Parameters

In our model, we consider the following parameters:

- *CPU* – Processing power in units
- *RAM* – Random access memory in GiB
- *Storage* – Storage in GiB
- *Isolation Level* – Level of isolation in units
- *Bandwith* – Bandwidth capacity in Gib/s

4.1.5 Isolation

Traditionally security in network infrastructure was inherent due to using a single domain and single trusted operator approach, but when considering slicing in modern networks, multi-tenancy requires considering security and isolation in the design of the system [5].

Inter-slice-isolation ensures that tenants' slices will be unaffected by other tenants' slices. Firstly slices will not be affected in regards to performance and reliability, such that the failure of a slice will not spread to other slices or consume more resources than allocated, and so that other slices do not experience reduced performance. Secondly, in regard to security, the misoperation whether intentional or not, will not affect other slices.

To achieve inter-slice-isolation it is possible to reserve each DC or MEC to a single slice, but this is quite an inefficient approach as this does not necessarily fully utilize the DC or MEC. Moreover, such an approach makes placing less flexible which might lead to suboptimal placements and routing which impact the performance of the slice and the cost for the INP. Instead, we will in our definition consolidate VNFs from different slices onto the same DC or MEC by adopting intra-slice-isolation such that each placed VNF is isolated from all other VNFs on the same DC or MEC, regardless of the parent slice.

In our definition network slice traffic is isolated such that different network slices can only receive traffic designated to the network slice. This type of isolation can be implemented by segmenting the network logically using existing technologies such as subnetting, access control lists (ACLs), virtual local area networks (VLANs), or the extended VLAN version called VXLANS. We see this network segmentation in Figure 4.3 and Figure 4.4 in the stippled box denoted "NS".

Secondly, we consolidate (section 4.1.3) VNFs on the same DC or MEC with the advantages that brings. However, this leads to the problem of isolation between VNFs as VNFs are no longer inherently physically isolated. Our isolation levels solve this problem by using virtualization to provide isolation between the VNFs and enable intra-slice-isolation.

We define three isolation levels where the third level is the most isolated. In our definition, we place containerization at the lowest level of isolation, because containerization provides sufficient isolation for most applications. Here each VNF is deployed in a container alongside its required services. Importantly this level shares a host operating system (OS) on top of the DC or MEC, which means that VNFs placed with this level of isolation on the same DC or MEC will share the same kernel but will have separate services.

The definition leaves out a potential level zero where there would be no isolation and where VNFs would be placed together on "bare metal" without any virtualization layer.

However, such a level would indeed not be isolated and therefore feature the disadvantages of not isolating such as only OS-level security and isolation and increased difficulty of management, deployment, and redeployment. Moreover, we see that the general cloud infrastructure and NFV are moving towards using virtualization not only for isolation but for ease of management.

We place the *Dedicated VM* level over *Containerization* as it is more isolated. This is because at this level each VNF placed on the same DC or MEC is deployed within a separate virtual machine (VM) so that each VNF operates on its own kernel with its own services, whereas the "Containerization" level would share the kernel between VNFs. The isolation of containers can be improved by using isolation methods or with intrusion detection systems and kernel security hardening, but the isolation remains inherently weaker than for VMs [19].

Finally, we define the third and most isolated level as dedicating a DC or MEC as this level features even stronger hardware-level isolation than the second level. Here each VNF reserves a DC or MEC and is physically separated from other VNFs.

In Figure 4.2 we see the isolation level definitions and in Figure 4.3 we see the different levels visualized at the bottom together with the axis of the level of isolation. On the top of Figure 4.3 we see the difference between levels 1 and 2 visually.

In Figure 4.4 the isolation of each slice and VNF is visually represented as the purple stippled jagged lines. In this diagram, we also visualize the consolidation of different VNFs on the same node but note that the consolidated VNFs are always of the same level and that for the third level, each DC and MEC is reserved for a single VNF.

Aspect	Containerization	VMs
Isolation Method	OS-level virtualization (Using namespaces, cgroups, host based intrusion detection and kernel security hardning)	Hardware-level virtualization (Using hypervisor, contextualization and filters)
Resource Sharing	Shared OS kernel	Separate OS instances
Performance	Lightweight and efficient	Overhead due to hardware virtualization
Startup Time	Fast startup	Slow startup
Footprint	Smaller footprint	Larger footprint
Isolation	Less isolation due to shared OS kernel Vulnerabilities can spread	More isolation no sharing and hypervisor Vulnerabilities are contained to VM

Table 4.2: Containerization Vs. VMs

Level	Type	Description	Implementation
3	Dedicated DC/MEC	VNFs have a one-to-one relationship with the underlying DC or MEC and are physically separated from other VNFs, this is considered the most isolated level.	Achieved by reserving dedicated DC or MEC to VNF.
2	Dedicated VM	Each VNF runs in a dedicated virtual machine (VM), with its own separate virtualized operating system, services, and kernel. No part of the VM is shared between two VNFs.	Achieved by deploying each VNF in its own VM using virtualization technology such as VMWare, KVM, or Hyper-V. Each VM is assigned dedicated resources such as CPU, memory, and storage.
1	Containerization	Each VNF runs in a dedicated container with its own services, but the VNFs share a common virtualized operating system and kernel. [20].	Achieved by deploying each VNF in its own container using a containerization technology such as Docker or Kubernetes. Each container is assigned dedicated resources such as CPU, memory, and storage.

FIGURE 4.2: Levels of isolation
Level 3 is the most isolated

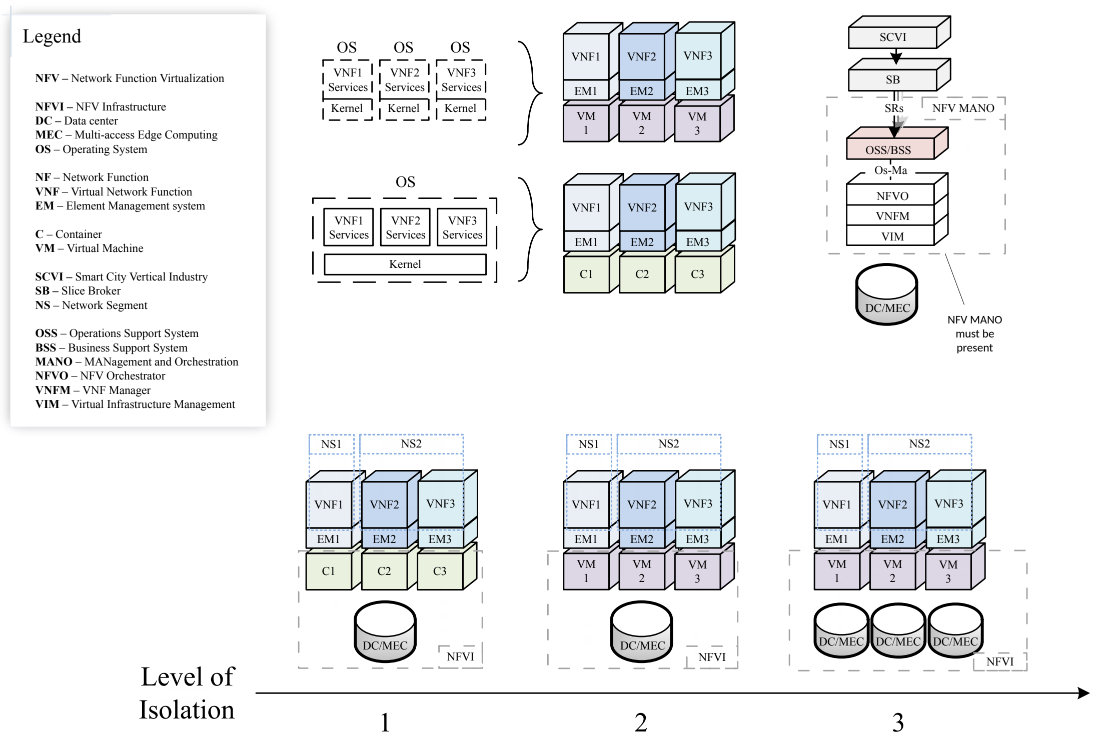

FIGURE 4.3: Levels of Isolation: Anatomy

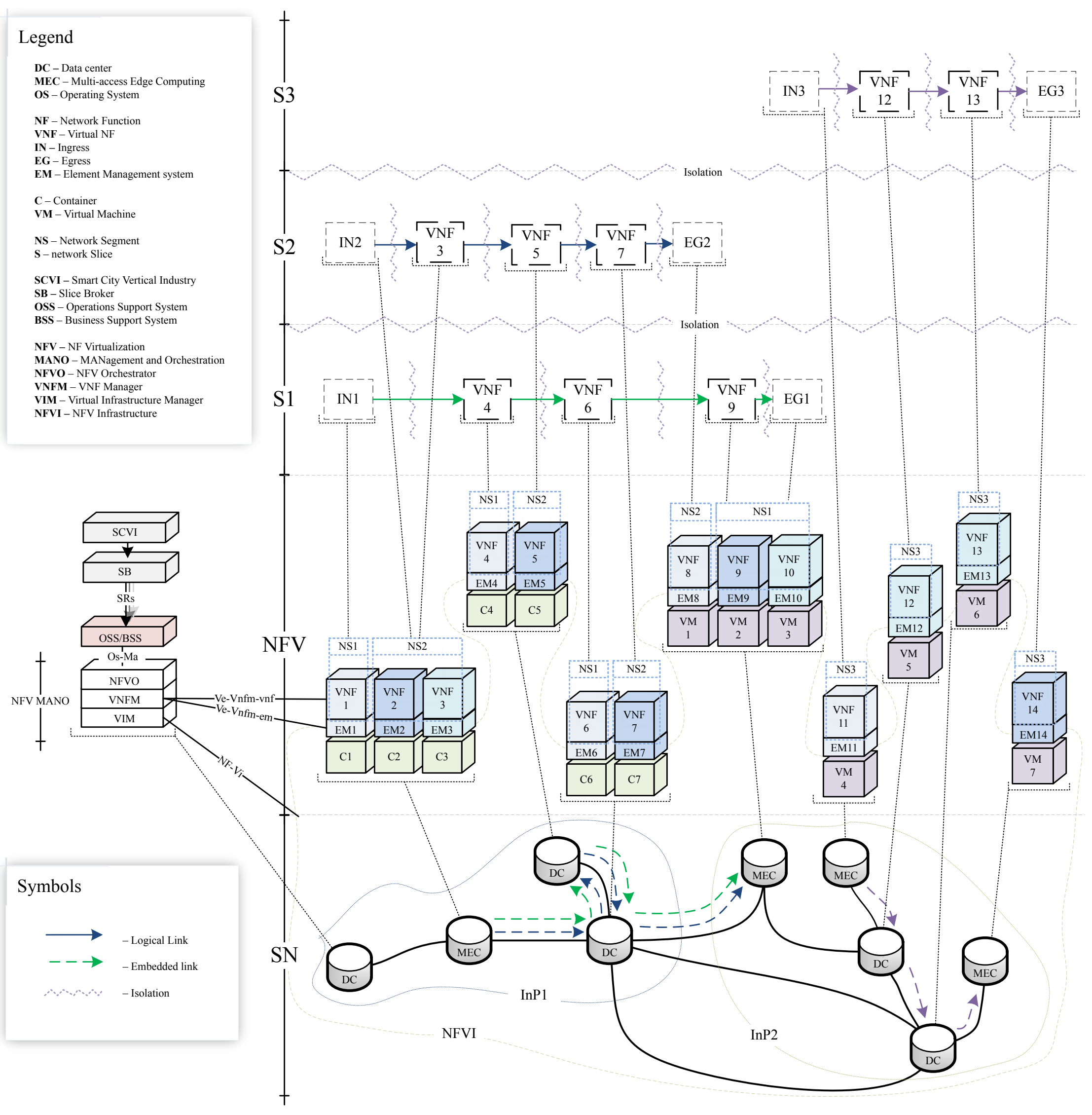

Figure 4.4: Isolation in NFV

4.2 Definition

4.2.1 SN

The parameters are given in Table 4.4 and Table 4.5.

Parameter	Unit	Distribution	Type	Data Type
CPU	Units	Uniform	Resource	Float
RAM	GiB	Uniform	Resource	Float
STO	GiB	Uniform	Resource	Float
ISL	Level	Uniform	Constraint	Int

Table 4.4: DC/MEC parameter value ranges.

Parameter	Unit	Distribution	Type	Data Type
BW	Mib/s	Uniform	Resource	Float

Table 4.5: Link parameter value ranges.

Symbol	Definition
N^S	Set of SN nodes
$n^s \in N^S$	Individual SN node
$M^S \subset N^S$	Subset of MEC nodes
$m^s \in M^S$	Individual MEC node
P_N^S	Set of n^s configurations
$PAR(n^s)$	Parameter value for a n^s,
	i.e. $CPU(n^s)$ gives the CPU value
L^S	Set of logical links
$l^s \in L^S$	Individual SN link
P_L^S	Set of l^s configurations
$PAR(l^s)$	Parameter value for a l^s,
	i.e. $BW(n^s)$ gives the bandwidth value
$G^S = \{N^S, L^S, P_N^S, P_L^S\}$	Undirected graph representation of SN

Table 4.6: Symbol Definitions for the SN

The SN is represented as an undirected graph (Equation 4.1) of SN nodes and logical links.

$$G^S = \{N^S, L^S, P_N^S, P_L^S\} \tag{4.1}$$

The set of SN nodes is defined as N^S and individual nodes are defined as n_s.

$$n_s \in N^S \tag{4.2}$$

The resources available for every n_s are the processing power (CPU in units), RAM available (RAM in GiB), and the level of isolation provided (*ISL* in units [1-3]) and whether or not the node is a MEC if not it is DC.

For the set of links L^S the bandwidth capacity (BW in Mib/s) is available

$$l_s \in L^S \tag{4.3}$$

P_N^S is denoted as the set of n_s configurations. The configuration value of a specific parameter can be retrieved with $PAR(n^s)$ for the node or $PAR(l^s)$ for links.

4.2.2 SR

The parameters are given in Table 4.7 and Table 4.13.

Parameter	Unit	Distribution	Type	Data Type
CPU	Units	Uniform	Resource	Float
RAM	GiB	Uniform	Resource	Float
STO	GiB	Uniform	Resource	Float
ISL	Level	Uniform	Constraint	Int
IOE	Boolean	N/A	Constraint	Int

Table 4.7: VNF Parameters

Parameter	Unit	Distribution	Type	Data Type
BW	Mb/s	Uniform	Resource	Float

Table 4.8: Logical link parameters

Symbol	Definition
D^V	Directed graph representation of the VNF-FG
$n^v \in N^V$	Individual VNF
P_N^V	Set of n^v configurations
$PAR(n^v)$	Parameter value for a n^v,
	i.e. $CPU(n^s)$ gives the CPU value
L^V	Set of logical links
$l^v \in L^S$	Individual SN link
P_L^V	Set of l^V configurations
$PAR(l^v)$	Parameter value for a l^v,
	i.e. $BW(n^v)$ gives the bandwidth value
$D^V = \{N^V, L^V, P_N^V, P_L^V\}$	Directed graph representation of SR

Table 4.9: Symbol Definitions for the SR

The VNF-FG is represented as a directed graph (Equation 4.4) of VNFs and logical links.

$$D^V = \{N^V, L^V, P_N^V, P_L^V\} \tag{4.4}$$

Due to the direction of the graph, the ordering of the VNFs comes naturally. The set of VNFs is defined as N^V and individual VNFs are defined as n^v(Equation 4.5).

$$n^v \in N^V \tag{4.5}$$

The resources demanded by every n^v are the processing power (CPU in units), RAM (RAM in GiB), and the level of isolation provided (*ISL* in units [1-3]) and whether or not the VNF is an ingress or egress (*IOE*, 1 or 0).

For every link l^v the demanded bandwidth capacity (BW in Mib/s.)

$$L^v \in L^V \tag{4.6}$$

P_N^V is denoted as the set of n^v configurations and P_L^V as the set of l^v configurations. The configuration value of a specific parameter can be retrieved with $PAR(n^s)$ for the node or $PAR(l^s)$ for links.

4.3 Solution

It is important that the DRL model learns and is optimized toward lowering cost and increasing the revenue for the InP. This will be done by rewarding the model for allocating as many SRs as possible while using as few resources of the SN as possible.

The VNF-FGE solution can be understood as a mapping process that maps the VNFs of the VNF-FG of an SR to appropriate SN nodes.

For the allocation of resources in the physical network, SRs that meet certain conditions will be accepted. Specifically, the resources of the n^s must have greater or equal resources compared to the requested resources in the n^v. These resources include CPU, RAM, storage (STO), and isolation level (ISL). Additionally, the constraints of the physical network must be satisfied.

Similarly, for logical links, the same conditions apply. The bandwidth in the SN must be of greater or equal capacity to the requested bandwidth in the VNF-FG.

Once an SR is accepted, the resource parameters in the SN will be updated to reflect the utilization of resources. However, there is an exception when the node has an isolation level of 3. In this case, the resources of the n^s will be fully reserved and set to zero.

Finally, when the lifetime of a service request is over, the resources of the SN will be restored to their previous values.

4.3.1 Parameter Specification

Parameter	Distribution	Data Type	Minimum	Maximum
CPU	Uniform	Float	50	100
RAM	Uniform	Float	50	100
STO	Uniform	Float	50	100
ISL	Uniform	Int	1	3

Table 4.10: DC/MEC parameter value ranges.

Parameter	Distribution	Data Type	Minimum	Maximum
BW	Uniform	Float	50	100

Table 4.11: Link parameter value ranges.

Parameter	Distribution	Data Type	Minimum	Maximum
CPU	Uniform	Float	1	50
RAM	Uniform	Float	1	50
STO	Uniform	Float	1	50
ISL	Uniform	Int	1	3
IOE	N/A	Bool	0	1

Table 4.12: SR parameter value ranges.

Parameter	Distribution	Data Type	Minimum	Maximum
BW	Uniform	Float	1	50

Table 4.13: Logical Link parameter value ranges.

4.3.2 Evaluation

Acceptance Rate

The acceptance rate (AR) is the ratio of accepted to received SRs. This ratio is an important measure of the performance of the InP, as the revenue comes directly from accepting SRs. Therefore, the revenue is maximized when the AR approaches one. The AR is the ratio of accepted to arrived SRs and is calculated as follows:

$$AR = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} Accepted(D^V, t)}{\sum_{t=0}^{T} Arrived(D^V, t)} \tag{4.7}$$

Long-Term Average Revenue

Long-term average revenue (LTAR) is the limit of the total revenue from allocating SRs divided by the time as the time approaches infinity. Thus it is a measure of how much revenue the InP can generate from allocating SRs over a long period. LTAR is written as:

$$LTAR = \lim_{T\to\infty} \frac{1}{T} Revenue(D^V, t) \tag{4.8}$$

Long-Term Average Cost

Similarly to LTAR Long-Term Average Cost (LTAC) is the limit of the total cost of leasing resources from the InP.

$$LTAC = \lim_{T\to\infty} \frac{1}{T} Cost(D^V, t) \tag{4.9}$$

LTRCR

The LTRCR considers both LTAR and LTAC and so it provides a full picture of the feasibility of the InP. LTRCR considers the ratio between LTAR and LTAC and is written as follows:

$$LTRCR = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} Revenue(D^V, t)}{\sum_{t=0}^{T} Cost(D^V, t)} \tag{4.10}$$

Objective

The objective of this thesis will be to maximize AR and LTAR and minimize LTAC jointly.

4.3.3 SN Generation

The SN Generator generates a virtual SN following a JSON configuration (Table 4.14) using the Networkx python library. The seed is chosen to generate a specific SN every time the configuration is used, if set to -1 a new seed is chosen each time.

The size of the SN can be any positive integer; however, scaling the size up reduces the performance of running simulations and training greatly as the increased amount of nodes and edges has to be loaded in memory and processed. A larger SN will be easier to embed nodes and edges on but will lead to an increased state space for the policy and value function networks of the DRL model, which require more computation and will lead to slower training.

On the other hand, decreasing the size of the SN makes it more likely to not be a fully connected graph due to the fact that edges between nodes are not guaranteed but instead given by the alpha and beta parameters. The alpha parameter determines the impact of the distance between two nodes on the connection likelihood, and a smaller value decreases the impact so that it is more likely for nodes farther away to connect. A higher Beta parameter gives a higher randomness of the graph generation. Because of the likelihood that the SN will not be connected for smaller SNs, we guarantee a connected graph by retrying the generation until it qualifies as a connected graph, but this only applies if the seed is set to -1.

The MEC parameter specifies the number of nodes that will be randomly chosen as MECs and must therefore be less or equal to the total number of SN nodes; the rest of the nodes will be DCs.

Parameter	Data Type	Minimum	Maximum
Seed	Integer	-1	Int Max
Nodes	Integer	1	Int Max
Alpha	Float	0	1
Beta	Float	0	1
MEC	Integer	0	$\lvert N^S \rvert$

Table 4.14: SN Generation Parameters

Figure 4.5 shows an example SN with only 20 nodes for display purposes. Here the nodes are colored based on the isolation level, and the parameters of the node are displayed at the bottom right of each node.

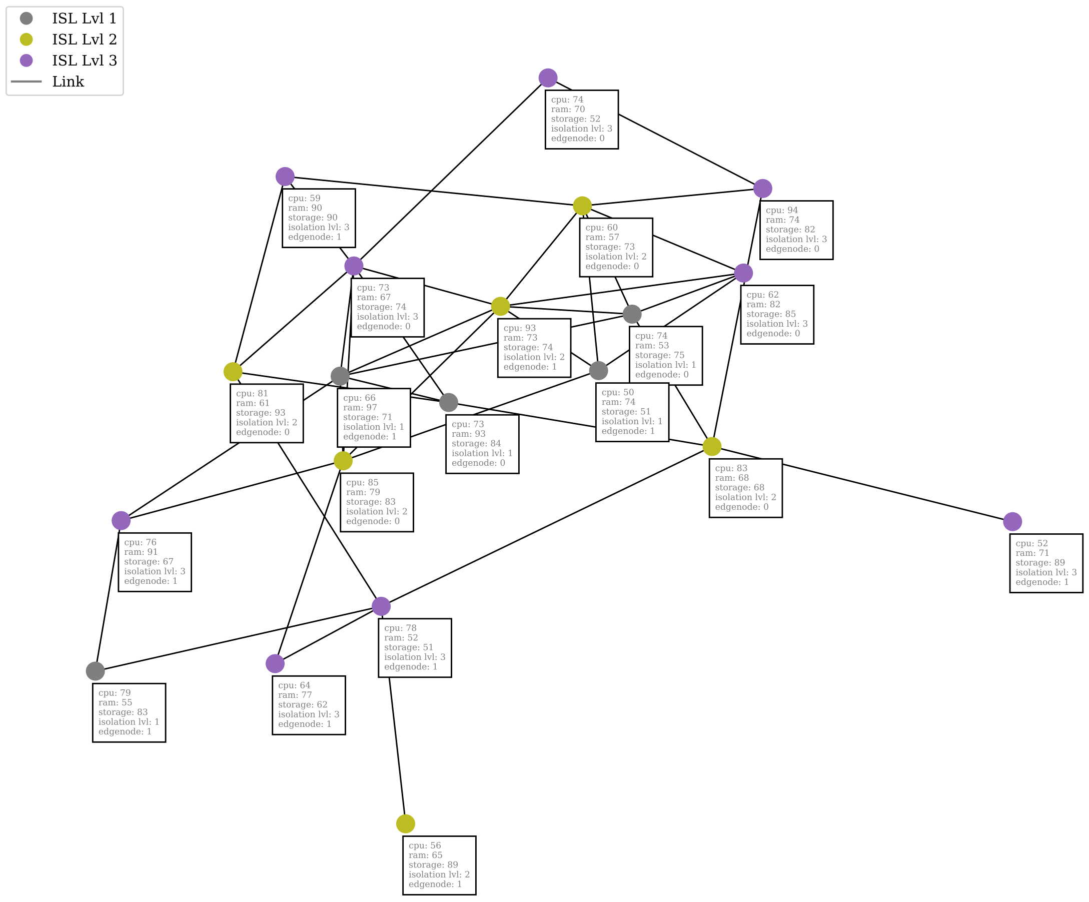
ISL Lvl 1
ISL Lvl 2
ISL Lvl 3
Link
cpu: 74 ram: 70 storage: 52 isolation lvl: 3 edgenode: 0
cpu: 59 ram: 90 storage: 90 isolation lvl: 3 edgenode: 1
cpu: 94 ram: 74 storage: 82 isolation lvl: 3 edgenode: 0
cpu: 60 ram: 57 storage: 73 isolation lvl: 2 edgenode: 0
cpu: 73 ram: 67 storage: 74 isolation lvl: 3 edgenode: 0
cpu: 62 ram: 82 storage: 85 isolation lvl: 3 edgenode: 0
cpu: 93 ram: 73 storage: 74 isolation lvl: 2 edgenode: 1
cpu: 74 ram: 53 storage: 75 isolation lvl: 1 edgenode: 0
cpu: 81 ram: 61 storage: 93 isolation lvl: 2 edgenode: 0
cpu: 66 ram: 97 storage: 71 isolation lvl: 1 edgenode: 1
cpu: 50 ram: 74 storage: 51 isolation lvl: 1 edgenode: 1
cpu: 73 ram: 93 storage: 84 isolation lvl: 1 edgenode: 0
cpu: 85 ram: 79 storage: 83 isolation lvl: 2 edgenode: 0
cpu: 83 ram: 68 storage: 68 isolation lvl: 2 edgenode: 0
cpu: 76 ram: 91 storage: 67 isolation lvl: 3 edgenode: 1
cpu: 52 ram: 71 storage: 89 isolation lvl: 3 edgenode: 1
cpu: 78 ram: 52 storage: 51 isolation lvl: 3 edgenode: 1
cpu: 79 ram: 55 storage: 83 isolation lvl: 1 edgenode: 1
cpu: 64 ram: 77 storage: 62 isolation lvl: 3 edgenode: 1
cpu: 56 ram: 65 storage: 89 isolation lvl: 2 edgenode: 1

FIGURE 4.5: Example of small SN

Algorithm 4.1 SubstrateGenerator

```

function GENERATESUBSTRATENETWORK($generationAttempts = 0$)
    $generationAttempts \leftarrow generationAttempts + 1$
    $S^N \leftarrow$ None
    $|N^S| \leftarrow$ configuration[$'nodes'$]
    $S^N \leftarrow$ generateNetwork(configuration, $|N^S|$)

    if not networkx.is_connected($S^N$) then
        return generateSubstrateNetwork($generationAttempts$)
    end if

    setSNParameters($S^N$, $P^S_N$)
    if configuration['includeIngressEgress'] then
        setSNEdgeNodes($S^N$)
    end if

    return $S^N$

end function


function GENERATENETWORK(parameters, nodecount)
    GLOBAL_CURRENT_SEED ← CURRENT_SEED
    networkType ← parameters[$'topology\_type'$].upper()

Match networkType
  Case'WAXMAN'
    return networkx.waxman_graph($n =$ nodecount, $\beta =$ parameters['beta'], $\alpha =$ parameters[$'alpha'$], seed = getAndUpdateSeed())

  Case'ERDOS-RENYI'
    return networkx.erdos_renyi_graph($n =$ nodecount, $p =$ parameters[$'p'$], seed = getAndUpdateSeed(), directed = parameters[$'directed'$])

end function
```

4.3.4 SR Generation

A virtual SR is generated following a JSON configuration (Table 4.15) using the Networkx python library. As the training and simulation will need a large number of unique SRs, compared to only needing one SN, the SR generator has been built to be able to generate large amounts of SRs efficiently.

Parameter	Data Type	Minimum	Maximum
Seed	Integer	-1	Int Max
Number of SRs	Integer	1	Int Max
Nodes	Integer	1	Int Max
Directed	Boolean	0	1
Branching Probability	Float	0	1

TABLE 4.15: SR Generation Parameters

The seed is derived from the SN generation seed and as such choosing a seed value of -1 will give randomized sets of SRs every time while any other seed will generate the same set every time.

The SR generator outputs sets of randomized SRs and the generation of any non-zero positive integer amount is possible.

If the "directed" value is set to 1 the SR will be considered as a linear directed acyclic graph (LDAG) where ordering is considered.

The "Branching Probability" parameter defines the likelihood of branching of the VNF-FG and in our case, we only consider one branch, but in the case of parallelization, a greater probability can be chosen to allow branching.

Algorithm 4.2 SR Generator

```

function GenerateServiceRequest
    getAndUpdateSeed()

    $chainLength \leftarrow$ np.floor(getParameterValue(configuration["request_attributes"]))
    $splitProb \leftarrow$ configuration["branch_probability"]

    $DAG \leftarrow$ generateBranchingDAG(chainLength, splitProb)
    setVirtualParameters(DAG, configuration)

    $D^V \leftarrow ServiceRequest(DAG, VNID)$
    $VNID \leftarrow VNID + 1$
    return $D^V$

end function


function GenerateRequests
    $requests \leftarrow \{\}$
    $count \leftarrow$ configuration["number_of_instances"]

    for in range(count)
        $D^V \leftarrow$ generateServiceRequest()
        $requests[ID(D^V)] \leftarrow D^V$

    arrivals, departures, requests $\leftarrow$ generateLifeTimes(requests, startTime)
    return arrivals, departures, requests

end function

```

4.4 DRL Training

4.4.1 Environment

Overview

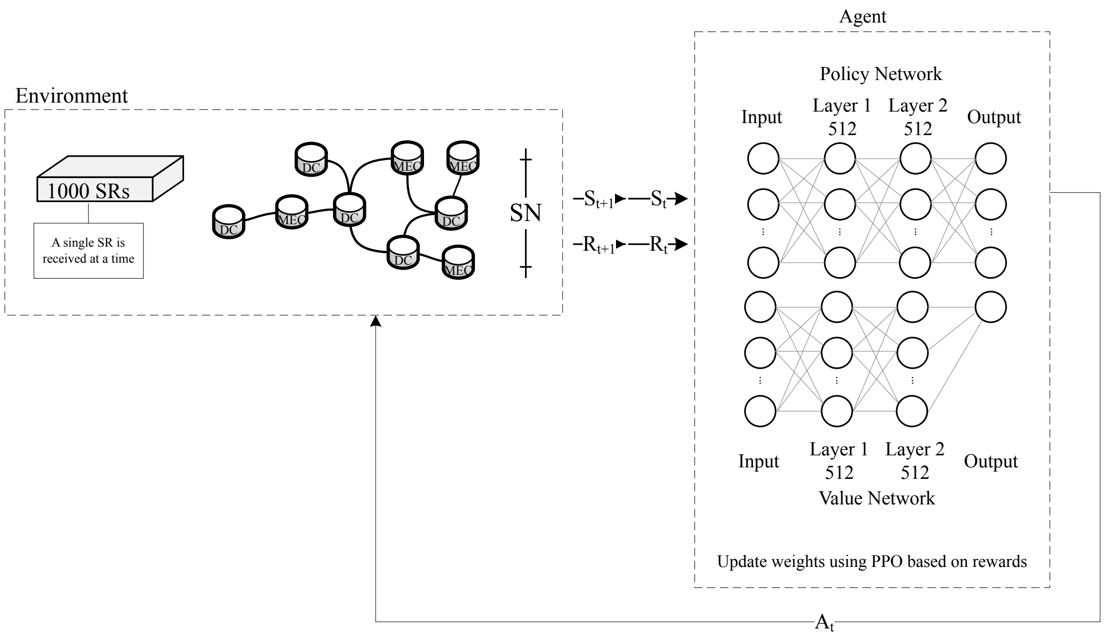

FIGURE 4.6: Overview of DRL environment

We train the agent in a Gym 0.21.0 environment that interfaces with our simulator. The weights of the agent's policy and value network are updated using the PPO algorithm. We provide the PPO algorithm with the necessary functionality to train by implementing a custom environment.

State Space

The input layer of the agent takes in a state from the environment and is what the agent observes about the environment. The state space might be modeled as an MDP, but when only partial information about the true state is observable by the agent we can model the state using a partial observable MDP (POMDP). In our testing, we try both a complete state space and a partial version.

When using a POMDP we only perceive information that is necessary for the completion of its task as a larger than necessary state space will slow down the learning and operation of the agent due to increasing the computations needed to update weights of the layers of the policy network and value function network while learning and needed for the operation of the agent when learning is completed.

Therefore we want to minimize the state space while providing the necessary information to the agent in the case of the POMDP. The agent must know the resources and constraints of VNFs and substrate nodes to be able to make a good judgment about where to place VNFs, in addition, the edges of the SR and SN will complicate the solution as the agent must perceive information about the edges or at least the proximity of the nodes to be able to reduce the cost of the accepted SRs as the edges play a part in this calculation.

- **Edge Information -** In this complete state space we provide the state of each node in the SN including the current value of each resource, and the static constraint values as well as IDs for all the nodes in the SN. In addition, we provide IDs of the nodes from which the edge is between for all edges in the SN. Finally, the request is fully represented with all the requirements and constraints of the VNFs as well as an index that states the current VNF for placement. The size of the space is given as:

$$\begin{aligned}|\mathrm{SP}_{\mathrm{Edge}}| = \\ &|N^S| \cdot (|p_n^s| + 1) \\ +& \\ &|L^S| \cdot 2 \\ +& \\ &\mathrm{maxLength}(N^V) \cdot |p_n^v| + 1\end{aligned} \tag{4.11}$$

- **Distance Information** In this POMDP model we remove information that is less important. We still provide the state of each node in the SN including the current value of each resource, and the static constraint values but instead of using IDs about nodes to provide information about edges, we give the normalized shortest distance between the SN node where the previous VNF of the current request was placed, and in the case of the first VNF of a request where there is no previous placement the value will be set to zero. In addition, we only provide the requirements of the current VNF to be placed. The size of the space is given as:

$$\begin{aligned} |\mathrm{SP}_{\mathrm{Distance}}| = \\ & |N^S| \cdot (|p_n^s| + 1) \\ + \\ & |p_n^v| \end{aligned} \tag{4.12}$$

The agent will receive an up-to-date vector with the current state where all values are normalized based on their respective maximum and minimum values. As previously seen in the "Edge Information" compared to the "Distance Information" the state space is a lot larger and therefore the algorithm for producing it is more complex.

Algorithm 4.3 EdgeInformation

```
1: $i \leftarrow 0$
2: vectorLength $\leftarrow |SP_{Edge}|$
3: environmentVector $\leftarrow$ np.zeros(shape=(vectorLength))
4: lastSNNodeID $\leftarrow |N^S| - 1$
5:
6: for $n^s \in N^S$ do
7:     environmentVector[$i$] $\leftarrow$ ID($n^s$)/lastSNNodeID)
8:     $i \leftarrow i + 1$
9:     for $parval \in PAR(n^s)$ do
10:        environmentVector[$i$] $\leftarrow$ normalizeParameterAccordingToMax(parval)
11:        $i \leftarrow i + 1$
12:    end for
13: end for
14:
15: for $l^s \in L^S$ do
16:    environmentVector[$i$] $\leftarrow$ fromNodeID($l^s$)
17:    $i \leftarrow i + 1$
18:    environmentVector[$i$] $\leftarrow$ toNodeID($l^s$)
19:    $i \leftarrow i + 1$
20: end for
21:
22: for $n^v \in N^V$ do
23:    for $parval \in PAR(n^v)$ do
24:        environmentVector[$i$] $\leftarrow$ normalizeParameterAccordingToMax(parval)
25:        $i \leftarrow i + 1$
26:    end for
27: end for
28: for $l^v \in L^V$ do
29:    environmentVector[$i$] $\leftarrow$ fromNodeID($l^v$)
30:    $i \leftarrow i + 1$
31:    environmentVector[$i$] $\leftarrow$ toNodeID($l^v$)
32:    $i \leftarrow i + 1$
33: end for
34: currentNormalizedNodeID $\leftarrow$ self.currentRequestNodeID/lastRequestID
35: environmentVector[$i$] $\leftarrow$ currentNormalizedNodeID
36: return environmentVector
```

Algorithm 4.4 DistanceInformation

```
1: $i \leftarrow 0$
2: vectorLength ← $|SP_{Distance}|$
3: environmentVector ← np.zeros(shape=(vectorLength))
4: lastSNNodeID ← $|N^S| - 1$
5:
6: distances ← getNormalizedPathLengthsTo(prevouslyPlacedOnSNNodeID)
7: for $n^s \in N^S$ do
8:     environmentVector[$i$] ← ID($n^s$)/lastSNNodeID)
9:     $i \leftarrow i + 1$
10:    for $parval \in PAR(n^s)$ do
11:        environmentVector[$i$] ← normalizeParameterAccordingToMax(parval)
12:        $i \leftarrow i + 1$
13:        environmentVector[$i$] ← distances[$n^s$]
14:        $i \leftarrow i + 1$
15:    end for
16: end for
17:
18: for $parval \in PAR(currentRequestNode)$ do
19:     environmentVector[$i$] ← normalizeParameterAccordingToMax(parval)
20:     $i \leftarrow i + 1$
21: end for
22: return environmentVector
```

Step Function

In implementing our OpenAi Gym interface we need to provide the state space, but also the custom step and reset functions such that the agent and PPO can interface with our resource allocation simulator. We define a step as the placement of one node from the currently arrived request. The agent will receive a state space vector and produce an action that represents where the agent chooses to place the current VNF, and this action is an integer within the action space which is the size of a number of SN nodes $(0 <= a < |N^S|, a \in \mathbb{N}_0)$.

The step function will take the action that the agent produced and return a new state to the agent along with a reward. The step function also calls a function to increment the internal simulation time so that when there are no new arrivals of SRs time will be skipped. The step function calls different internal simulation functions such that the simulation proceeds and updates information used for statistics and logging.

Algorithm 4.5 StepFunction(action)

```

reward ← 0
episodeFinished ← $false$
isPlacementSuccess ← place(action)

if isPlacementSuccess then
    reward ← $reward$ + getPlacementReward()
    if isNodesPlaced() then
        isEdgesEmbedded ← embedEdges(action)

        if isEdgesEmbedded then
            reward ← $reward$ + getAcceptanceReward()
            episodeFinished ← $true$
        else
            edgeFailure()
            episodeFinished ← $true$
        end if

    else
        episodeFinished ← $true$
        nodeFailure()
    end if
end if

updateSimulation()
updateStatistics()
updateLogging()
return getNewState(), reward, episodeFinished

```

Reset Function

The reset function's task is to ready the environment for a new episode. In our case, we have found that in regards to learning the best is to only do a full release of all resources after all the arrivals of the full scenario have been processed which is in our case 1000. Therefore we can see in Algorithm 4.6 that the reset is deferred until all SRs are processed.

We also find that whether these 1000 SRs are generated randomly every time or a single set of 1000 SRs are re-used makes little to no difference. Therefore we will reuse the same set of 1000 SRs for training and generate a different set of 1000 SRs when the agent is evaluated after the completion of the training.

Algorithm 4.6 ResetFunction(action)

```
1:
2: reward ← 0
3: episodeFinished ← $false$
4: isPlacementSuccess ← place(action)
5:
6: if isAllRequestsDone() then
7:     releaseEmbeddings()
8:     if configuration[reuseRequests] then
9:         arrivals ← initialArrivalsCopy
10:        departures ← initialDeparturesCopy
11:    else
12:        (arrivals, departures, requests) ← GenerateRequests()
13:    end if
14: end if
15:
16: updateSimulation()
17: return getNewState()
18:
```

Rewards

The PPO algorithm should learn our agent to optimize our objective, including minimizing the LTAC and maximizing the LTAR and AR. To do so, we must create a reward system that encapsulates the objective so that the agent will be optimized toward a policy that best fits the objective.

We determine that there are mainly three ways the reward can be implemented. Firstly we can reward the agent for every accepted SR; we call this the acceptance reward. Secondly, it is also possible to reward the agent per VNF it places; we call this the placement reward. Thirdly a combination of the first two methods can be used.

It is reasonable to assume that giving a placement reward for choosing an appropriate SN node will optimize the maximum VNFs placed and the acceptance rate. This comes from the fact that the acceptance of an SR relies entirely on the placement of all its VNFs, and as such, increasing the likelihood of a successful placement will indeed increase the likelihood of the acceptance of a request. However, it is also necessary to optimize the placement regarding the link cost and, therefore, the proximity and consolidation of placed VNFs in an SR to reduce the required links. Therefore we will adopt a combined approach to the reward to provide the full optimization information needed easily.

The acceptance reward can be modeled in many ways, and the most obvious one is to model it as close to the objective as possible. The AR is inherent to getting an acceptance reward, so we only need to represent the two other objective metrics. In this regard, we will try using the LTRCR as an acceptance reward. Still, it's worth noting that the perceived importance of policy changes will diminish over time as the LTRCR is a running average that eventually spans the arrival of 1000 SRs.

$$\text{AccRew}_{\text{LTRCR}} = \text{LTRCR}$$

We believe that the LTRCR has a disadvantage in learning because it is a running average. This means that the efficiency of an accepted SR in terms of LTRCR matters less and less as the episode progresses, which comes in addition to the reward being discounted by the PPO. Therefore we also test a per SR revenue-to-cost ratio as an acceptance reward:

$$\text{AccRew}_{\text{RCR}} = \frac{\text{Revenue}(D^V)}{\text{Cost}(D^V)}$$

We also try simply the revenue of the SR minus the cost of allocation:

$$\text{AccRew}_{\text{RMC}} = \text{Revenue}(D^V) - \text{Cost}(D^V)$$

Another more complicated variant of the $\text{AccRew}_{\text{RMC}}$ which rewards the revenue minus costs of extra embedded edges in comparison to the SR links and adds the LTAR:

$$\text{AccRew}_{\text{RMEC}} = \text{Revenue}(D^V) - (\text{Cost}(L^V) - \text{Cost}(L_V^S)) + LTAR$$

In addition, we will try a simpler form of acceptance reward that is less similar to the objective to test whether simplifying the reward will also make it easier to learn or not and the performance of a simpler reward. Therefore we call this acceptance reward a simple reward. The simple reward will convey only the most basic and important information to the PPO algorithm, firstly the number of edges embedded in comparison to the edges present in the SR will be considered to convey to what degree the acceptance of the SR has optimized for the least amount of links used in the SN. Moreover, the difficulty of achieving this optimization will be considered by rewarding the length of the SR. The simple reward formula is given as follows:

$$\text{AccRew}_{\text{Simple}} = |N^V| + |L^V| - |L_V^S|$$

where:

$$\begin{aligned} |N^V| &: \text{Number of nodes in the currently accepted SR} \\ |L^V| &: \text{Number of logical links in the currently accepted SR} \\ |L_V^S| &: \text{Number of embedded links in the SN} \\ &\quad \text{that corresponds to the currently accepted SR} \end{aligned}$$

Finally, we try another simple variant, dividing the number of SR links by the number of embedded links. In this reward function, we get a reward based on the difficulty of placement, which comes from the number of SR links, and then we scale this number by how efficiently we managed to place the VNFs, which comes from the number of embedded links:

$$\text{AccRew}_{\text{SRLtoSNL}} = \frac{|L^V|}{|L_V^S|}$$

where:

$$\begin{aligned} |L^V| &: \text{Number of logical links in the currently accepted SR} \\ |L_V^S| &: \text{Number of embedded links in the SN} \\ &\quad \text{that corresponds to the currently accepted SR} \end{aligned}$$

Chapter 5

Evaluation

In this chapter, we compare our results to our implementation of GRC proposed by Gong et al. [21]. All the objective function metrics will be used for evaluation. We will show the PPO learning and evaluation process in Appendix A, and show the results from our trained model compared to the GRC in section 5.2.

5.1 Setup

5.1.1 GRC

We change a part of the GRC algorithm to support the additional resources available in our thesis to make a fair comparison (algorithm 5.1). Otherwise, the algorithm stays the same.

Algorithm 5.1 GetResources

```
1: resources ← $\{\text{None}\}^{|N^S|}$
2: resourceSum ← 0
3: i ← 0
4: for $n^s$ in $N^S$ do
5:     availableResources ← $\text{CPU}(n^s) + \text{RAM}(n^s) + \text{STO}(n^s)$
6:     resources[$i$] ← availableResources
7:     resourceSum ← resourceSum + availableResources
8:     $i \leftarrow i + 1$
9: end for
10: return $\frac{\text{resources}}{\text{resourceSum}}$
```

5.1.2 PPO

We use the version of PPO called PPO-clip, where the change in policy parameters is clipped to improve learning stability. PPO-clip updates the policy parameter θ to maximize the clipped surrogate objective function $L^{CLIP}(\theta)$. Similarly to Equation 2.13, the update process Equation 5.1 iterates until it converges.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta L^{CLIP}(\theta_k) \tag{5.1}$$

Here, the learning rate is also α, and $\nabla_\theta L^{CLIP}(\theta_k)$ is the gradient of the clipped surrogate objective function with respect to the policy parameter at iteration k.

$$L^{CLIP}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\Big)\Big] \tag{5.2}$$

Here, θ is the policy parameter, $\hat{A}_t$ is the estimated advantage function at time t, and $r_t(\theta)$ is the probability ratio of the new policy to the old policy at time t given the current state and action, which is expressed as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{5.3}$$

5.1.3 Model Simulation

The simulation of the model differs from the training process regarding how actions from the model are used. In the training process, actions are picked non-deterministically according to the PPO entropy parameter. When simulating, however, the best action according to the model is always chosen first, and should this action not be possible, the simulation will pick the next action until possible, or there are no more actions (Algorithm 5.2). In addition, the seed is updated after the SN generation and before the SR generation such that the SN is the same as when training, but SRs are unique to the simulation.

Algorithm 5.2 Model Simulation

```
1:
2: incrementSeed(1000000)
3: generateRequests()
4: model ← PPO.load
5: timesToEmbedd ← []
6:
7: while simulationTime ≠ maximumTime do
8:     action ← model.predict(obs, deterministic = True)
9:     obs_tensor ← torch.tensor(np.array([obs]))
10:    Probs ← model.policy.get_distribution(obs_tensor).distribution.probs
11:    sortedDistribution ← torch.sort(probs)
12:    indices ← list(np.array(sortedDistribution.indices)[0])
13:
14:    for SNNodeID in indices do
15:        if sufficcientNodeResources(SNNodeID, RequestID, RequestNodeID) then
16:            action ← SNNodeID
17:            break
18:        end if
19:    end for
20:
21:    obs, done ← SNEnv.step(action)
22: end while
23:
```

5.1.4 SN

Parameter	Value
Seed	123
Nodes	100
Alpha	0.5
Beta	0.5
MEC	30

Table 5.1: SN

We use the same SN with seed 123 for DRL training and evaluation. We choose an SN size of 100 as this is a reasonable number of nodes that can still be computed and trained on and chosen by previous papers Table 3.3.

We choose 30% of the SN as MECs as this number will allow us to test the impact of constraining the ingress and egress VNFs to a smaller percentage of the SN. The nodes of the SN are given parameters according to Table 4.10.

5.1.5 SRs

Parameter	Training	Evaluation
Seed	Derived SN Generator Seed	Derived SN Generator Seed
Number of SRs	1000	1000
Nodes	Uniform(4-8)	Uniform(4-8)
Average Lifetime	250	250, 500, 1000, 2000, 4000
Arrival Rate (λ)	0.04	0.04
Directed	1	1
Branching Probability	0	0

Table 5.2: SRs

Each SR generated will consist of 4 to 8 VNFs, given by a uniform distribution. In our case, the SR will be a linear directed acyclic graph (LDAG) as we consider the ordering of VNFs, and thus we set the "directed" value to 1.

We generate a set of 1000 SRs for DRL training purposes and a different set of 1000 SRs for evaluation results to ensure that the DRL model is not unfairly optimized towards the evaluation SR set.

The lifetime of each SR is given randomly by the Poisson distribution. We set the intensity such that the average is 250 for the training of our model, but we will evaluate lifetimes of 250, 500, 1000, 2000, and 4000 time units. We set the arrival rate(λ) to 0.04 for four arrivals per 100 time units.

5.1.6 DRL Training

We begin PPO training using the default hyperparameters. Then, we tune the parameters through extensive testing and choose the best reward functions; the full training process can be seen in Appendix A.

Exp.	**Learning rate**	**Epochs**	**Steps**	**Batch Size**	**Ent Coef.**	**Clip Range**	γ	**Gae** λ	**Clip Range VF**	**VF Coef.**
1	0.0003	10	2048	64	0.01	0.1	0.99	0.95	0.2	0.5
50	0.00005	10	8192	256	0.3	0.1	0.99	0.99	0.2	0.5

Table 5.3: Default PPO hyper parameters (1), Vs. tuned parameters (50)

Experiment	**Acceptance Reward**	**Placement Reward**
1	RMEC	0.001
50	SRLtoSNL	0.01

Table 5.4: Worst (1) Vs. best reward functions (50).

In Figure 5.1 and Figure 5.2 we show the training of the PPO agent according to the two configurations listed in Table 5.3 and Table 5.4. The placement fail rate (PFR) is graphed in the two figures, and this metric tells us how often the agent fails at placing nodes as a percentage. Note, however, that this metric only counts the fails that happen when the agent tries placing a VNF, so when a placement fails, if there are more VNFs left, these cannot fail or be placed as the whole SR is discarded after the first fail. However, it is still a good measure of the general performance of the agent, where of course, lower values are better.

The same goes for the placement rate (PR), which also does not count VNFs that were never considered for placement, and a positive gradient in this metric indicates a general improvement.

The mean reward shows how the agent was rewarded, and when comparing different reward functions, this graph will be of a different scale. However, a positive trend tells us that the agent's policy is improving regarding the acceptance and placement reward. Still, importantly, the agent's real performance relies on reward functions that describe the objective.

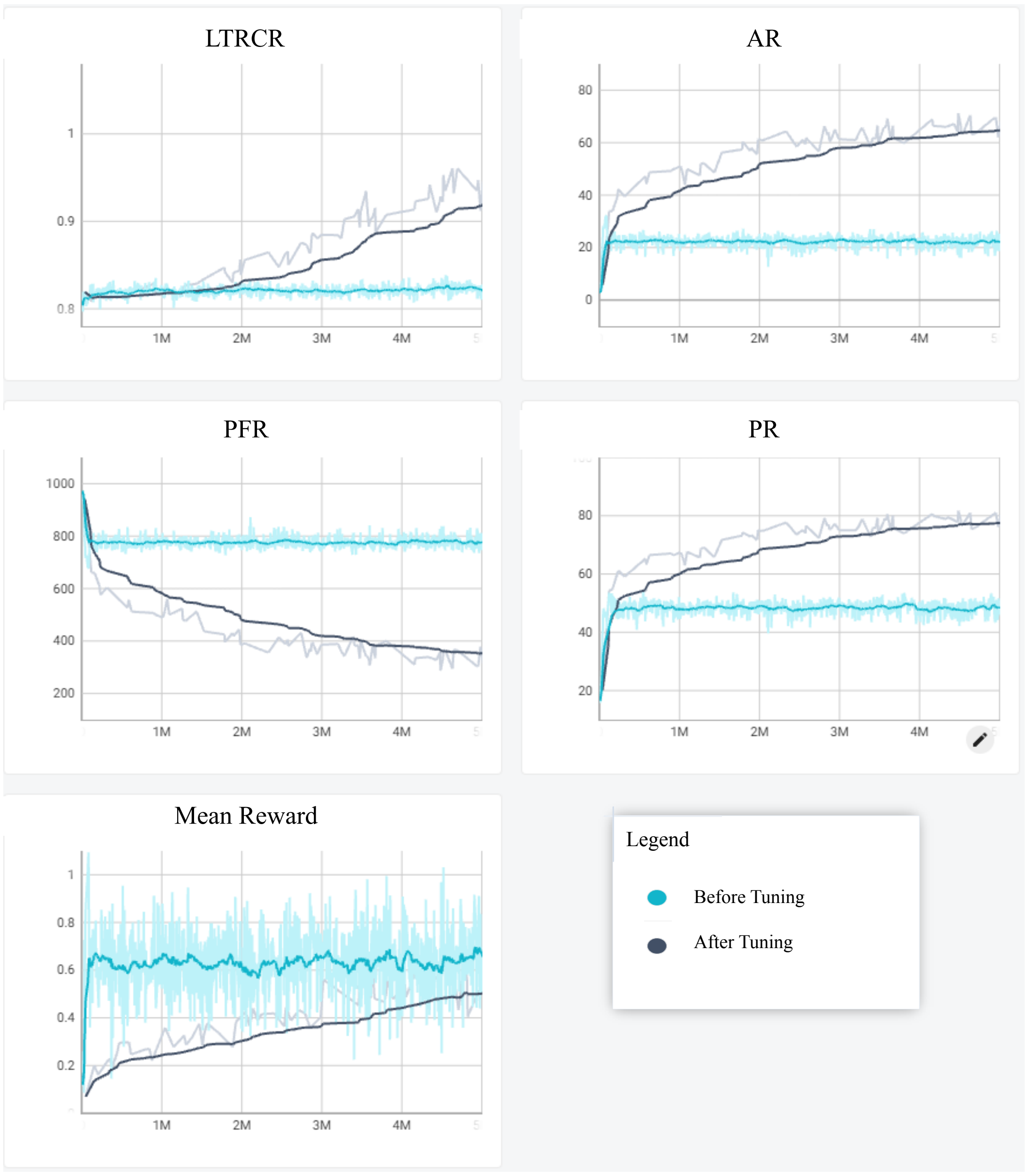

FIGURE 5.1: Training performance using default Vs. tuned hyperparameters and reward functions. Here 5 million steps are shown.

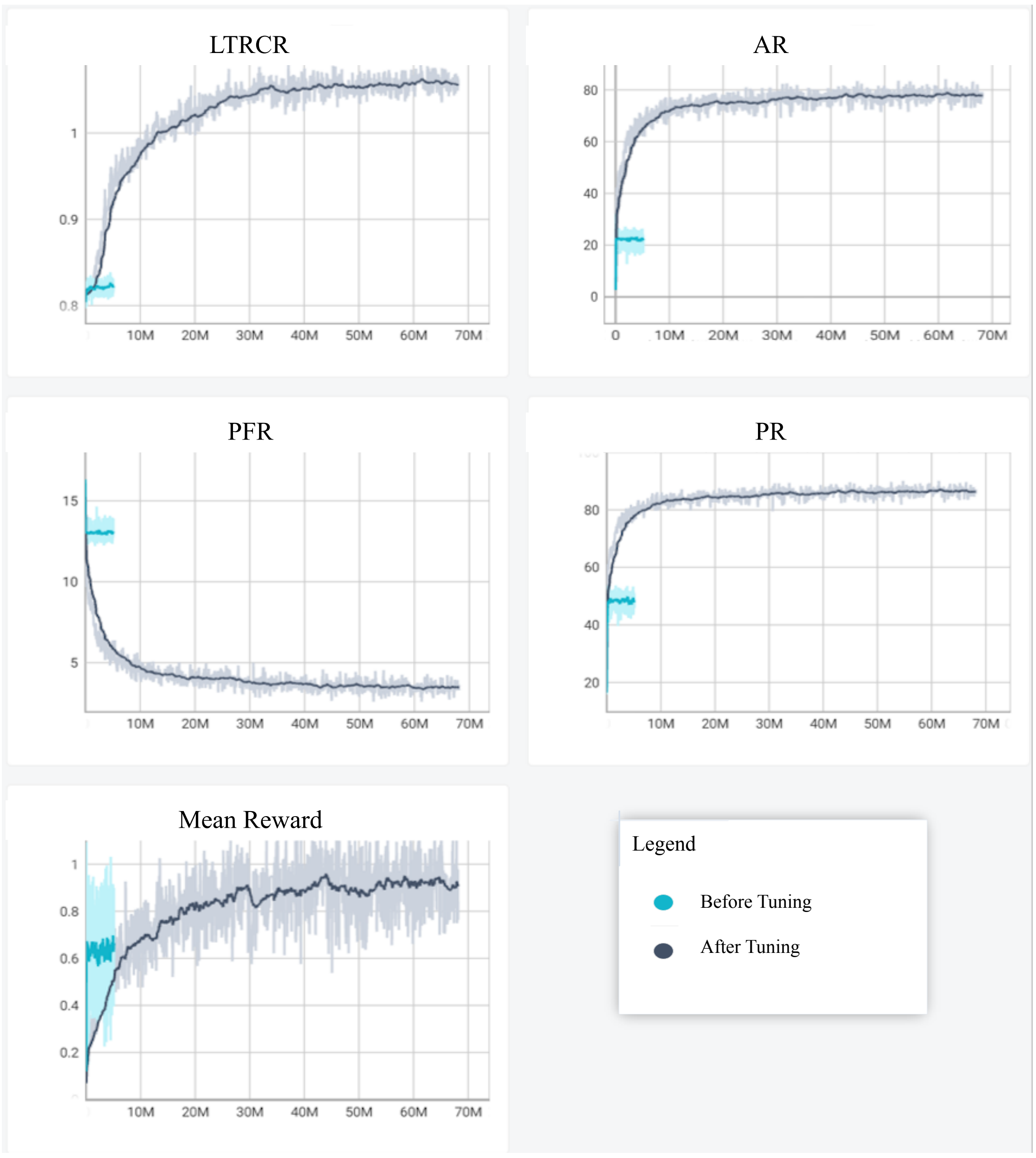

FIGURE 5.2: Training performance using default Vs. tuned hyperparameters and reward functions. Here 70 million steps are shown. We stopped after 5M steps for the untuned training since there were no significant developments.

5.2 Results and Analysis

We present our simulation results comparing our PPO model to GRC with an average SR lifetime of 250 (Figure 5.3), 500 (Figure 5.4), 1000 (Figure 5.5), 2000 (Figure 5.6), and 4000 (Figure 5.7) time units. In addition we show the allocation time per SR when a lifetime of 250 is selected in Figure 5.8.

5.2.1 AR

The AR shows expected results when comparing the lifetimes for higher lifetimes where fewer resources are available over time; the AR is lower. From these graphs, we can see that the AR for lifetimes 250 (Figure 5.3) and 500 (Figure 5.4) GRC performs better than our model. However, our model performs better in higher lifetimes (1000 (Figure 5.5), 2000 (Figure 5.6), and 4000 (Figure 5.7)). The AR results indicate that our model performs worse in regards to AR when there are many resources available compared to GRC. Conversely, the results also indicate that when there is more resource contention, our model manages to make better placements that, over time, allow for the acceptance of more SRs.

5.2.2 LTAR

The LTAR of our model shows close to equal performance to GRC in 250 (Figure 5.3) and 500 lifetime (Figure 5.4) and slightly better in the rest of the plots (1000 (Figure 5.5), 2000 (Figure 5.6), and 4000 (Figure 5.7)), which means that our model can extract as a little more revenue than GRC in the long term.

5.2.3 LTAC

LTAC for our model shows an improvement over GRC in all the plots. However, the difference is greatest for 250 lifetime (Figure 5.3) where there is a 19% improvement, and this difference seems to dwindle as the lifetime is increased as there is only a 5% improvement for 4000 lifetime.

5.2.4 LTRCR

Our model's LTRCR is substantially higher than GRC in all the lifetimes tested. For 250 lifetime (Figure 5.3), there is a 20% increase over GRC, while this difference also decreases with the increase of the lifetime, and for 4000 lifetime we see an improvement of 11%. The LTRCR results show that our model is up to 20% more efficient.

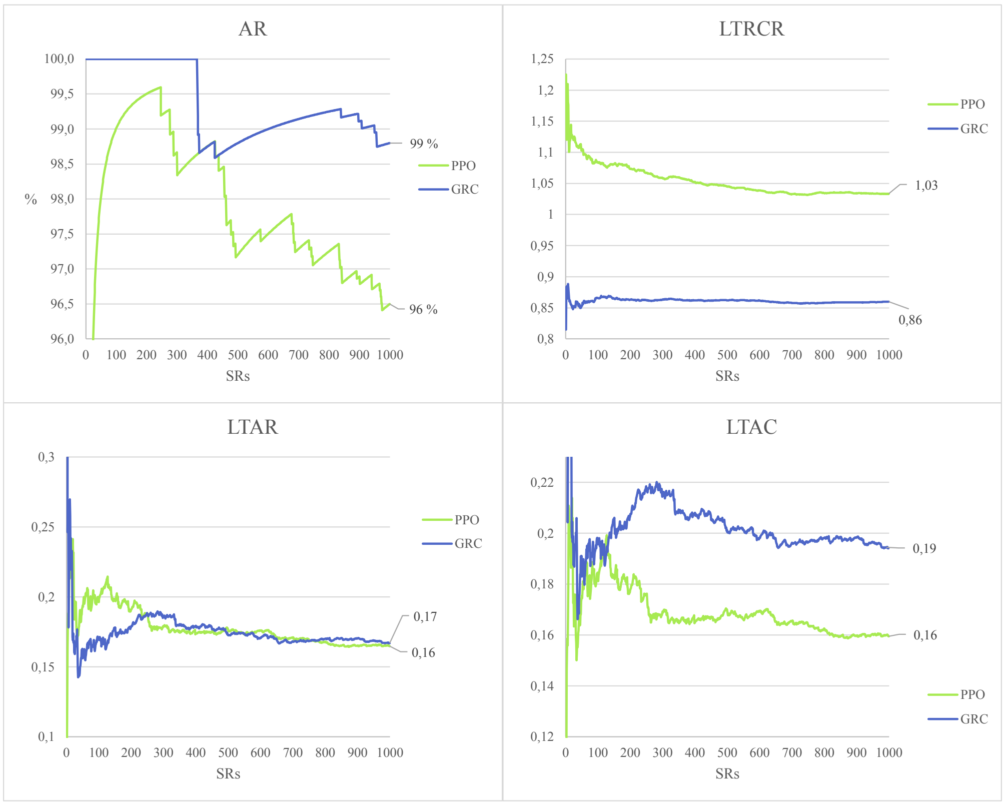

FIGURE 5.3: Results with an average lifetime of 250

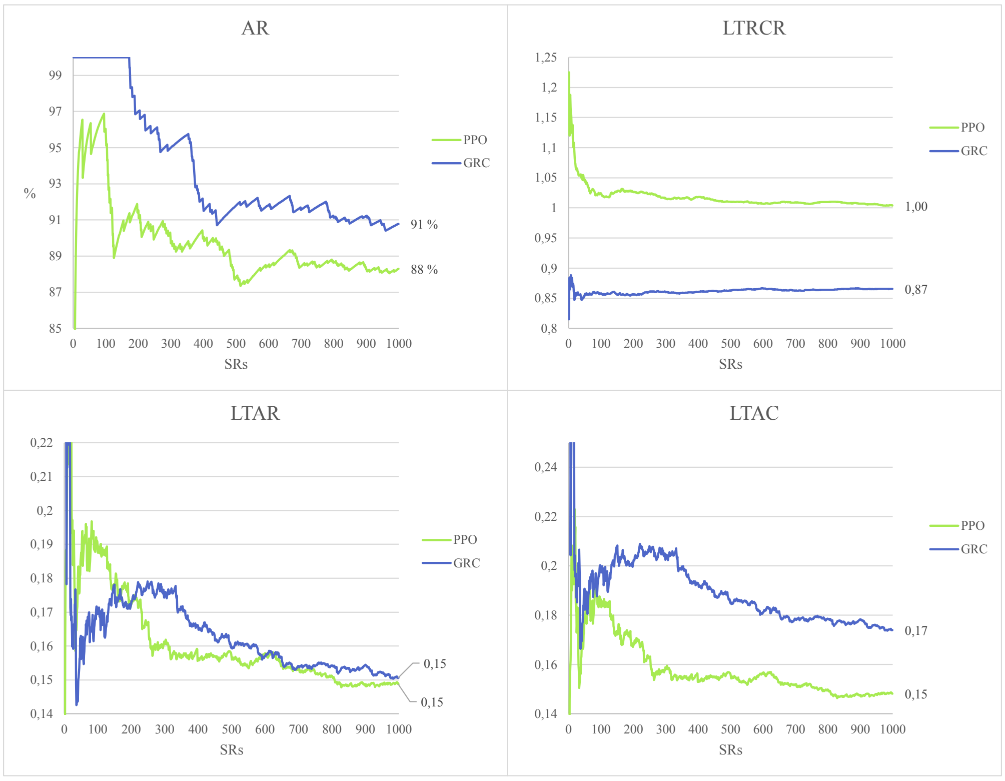

FIGURE 5.4: Results with an average lifetime of 500

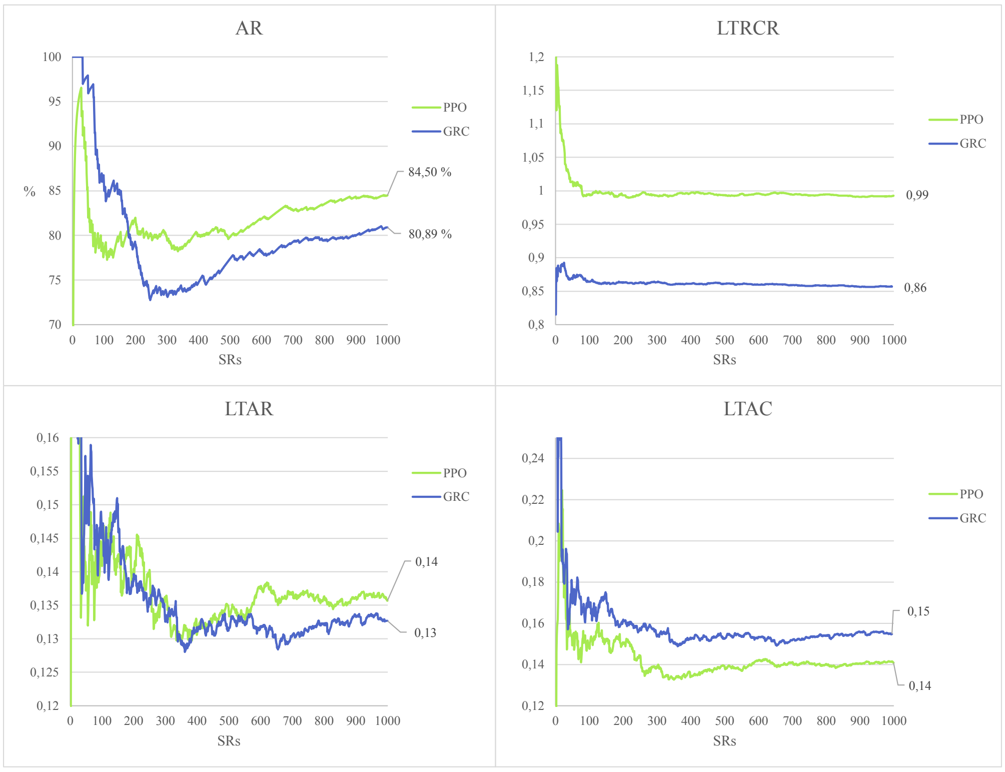

FIGURE 5.5: Results with an average lifetime of 1000

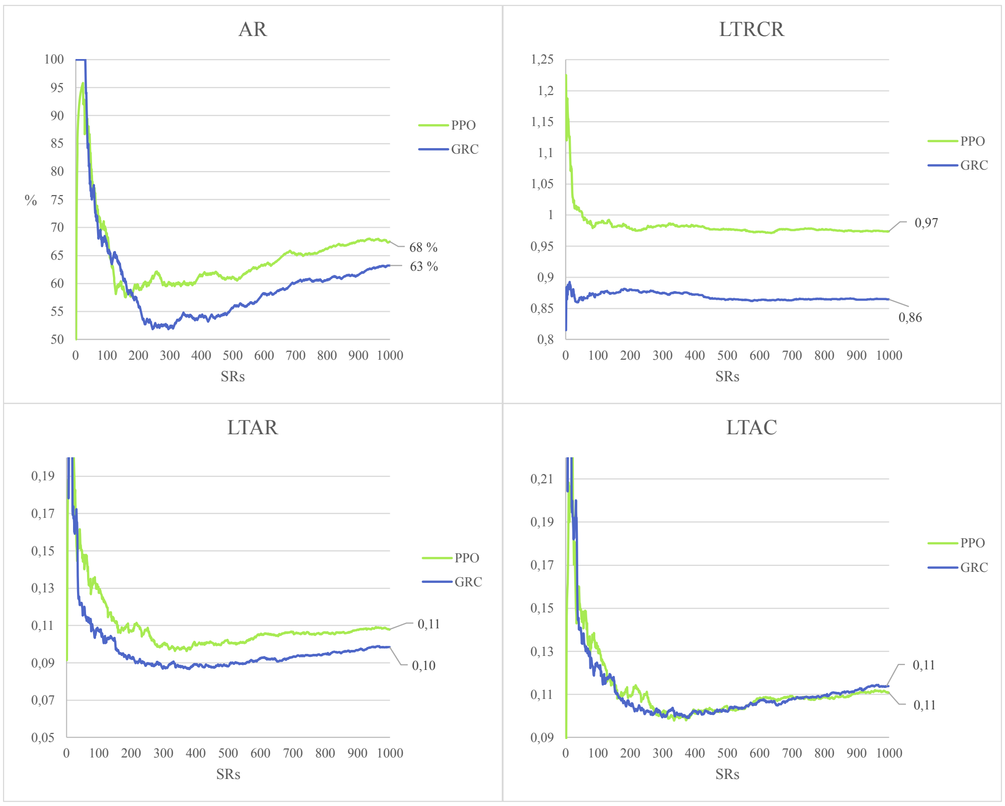

FIGURE 5.6: Results with an average lifetime of 2000

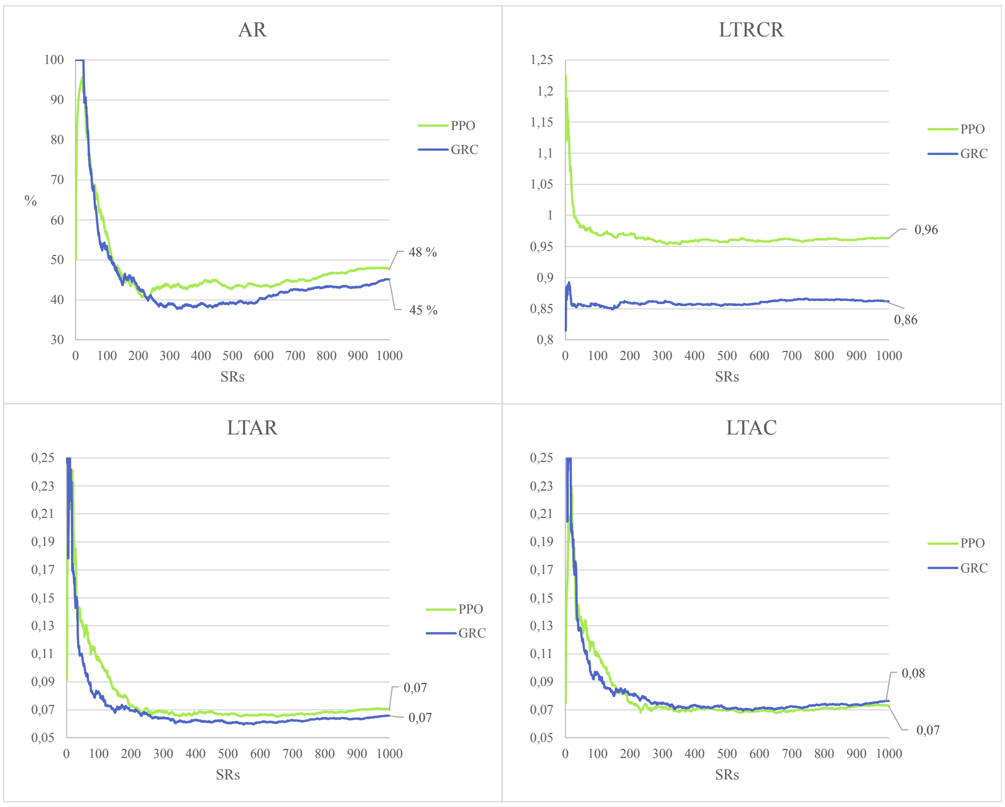

FIGURE 5.7: Results with an average lifetime of 4000

5.2.5 Allocation Time

The allocation time graph shows how much time was spent per SR allocation. Here we see that our model is 25x faster in terms of its allocation time. While the allocation time does not impact our problem model it is reasonable to assume that in real-world scenarios this difference could be quite important, i.e., in situations where a SR is not relevant after a certain time period. Or where the arrival rate is so high that not falling behind on allocations becomes important. It is imaginable that allocation time will important in URLLC given its latency requirements.

GRC has not only a higher average time consumption but also more unstable results with a peak going as high as 153 ms which is a 3x deviation from the average. Our model consistently performs around 2 ms with only slight deviations.

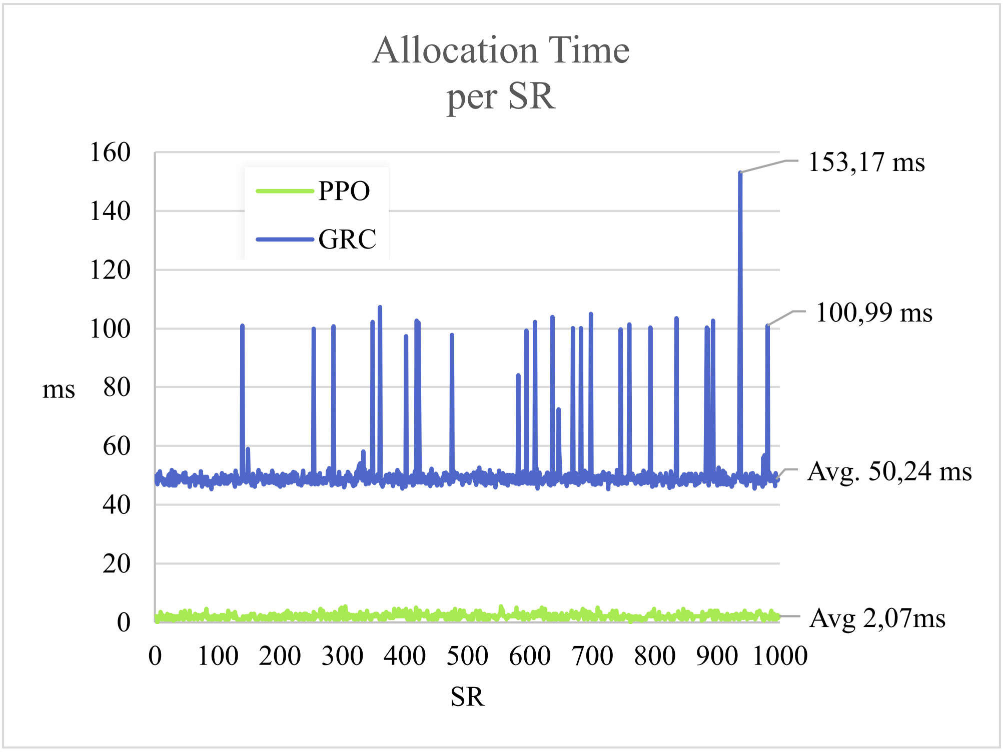

FIGURE 5.8: Average computational time per accepted SR

5.2.6 Overall Performance

Viewing the AR in combination with LTRCR gives a good understanding of the overall performance of our solution in comparison to GRC. Since the LTRCR is a ratio between the LTAR and LTAC, it gives information about both, though we can look at LTAR and LTAC for more details. Due to the metrics being the respective long-term averages, the graphs look spikier in the beginning because of the smaller average. Therefore we can disregard the first few spikes between 0-100 SRs. When understanding the results, the trends of the graphs tell us how the performance changes as SRs arrive. At the same time, the final value gives us the total average and is a representative measure of the total performance in the simulation.

In the results from 250 (Figure 5.3) and 500 lifetime (Figure 5.4) we see that GRC is about 3% better in terms of AR, however since the LTRCR results show a 20% difference in favor of our solution, we can argue that our solution is better overall. At the same time, it trades some AR for a higher LTRCR. This is what we see from training, too: if the agent receives a relatively bigger reward from placements only, the agent will learn to prioritize maximizing only the AR at the cost of LTRCR, and we assume that we could produce an agent that would perform similarly to GRC in if we had chosen placements rewards only. In addition, we assume that using the same reward functions, a performance gain that would equalize our solution to GRC or exceed it in the AR metric for lower lifetimes would be possible given more time; either from more fine-tuning of the PPO hyperparameters, from a longer training time or a combination of the two. This 3% difference might also be due to an overtraining on the training SR set, which of course GRC is not susceptible to. In addition, the performance will vary for each SR test set, and the results might not be the same choosing a different seed for the SN.

In the higher lifetimes (1000 (Figure 5.5), 2000 (Figure 5.6), and 4000 (Figure 5.7)), we easily see that our model performs better in all metrics. Again we see that the LTRCR is substantially higher, but in this case, the AR is also higher. The improved LTRCR mostly comes from the reduced LTAC, but in these higher lifetimes, also the LTAR exceeds GRC's.

Considering the around 25x decreased allocation time per SR compared to GRC, the advantage of our solution is increased. Overall the results show that our solution performs better, especially regarding LTRCR and allocation time.

Chapter 6

Conclusions

Our thesis introduced the basic concepts of resource allocation for SCVI in smart cities and RL. Furthermore, our thesis discussed the related work in this field of research, especially regarding DRL solutions to the VNF-FGE problem. We defined isolation levels in NFV to ensure network security and reliability. Our isolation levels definition aligns with the principles of network slicing and is compatible with NFV. We also gave a comprehensive problem definition and solution that allows us to understand the specifics of the problem and implementation. In our DRL training, we extensively experimented to arrive at much-improved hyperparameters and reward settings and documented our steps in tables and graphs with relevant metrics. Finally, we discussed our evaluation settings and results and gave insight into the metrics.

In training the model, we quickly found that the complete state space variant we proposed was many times slower than the partial version, which caused us to discard it and continue only with the partial state space due to the time constraints of the thesis. Moreover, we found simpler reward functions to yield the best training results.

Our objective was to minimize the LTAC and maximize the LTAR and AR. We developed a resource allocation model trained using the state-of-the-art PPO algorithm to achieve this objective. We tested our model using different average SR lifetimes and demonstrated improved performance compared to GRC. The results show that our model can help to increase the LTRCR of resource allocation for SCVIs in a smart city by 20% compared to GRC. Our results also reveal our DRL model's 25x allocation time improvement versus GRC, which greatly improves the SR allocation time of the InP.

6.1 Future Directions

Future work may better separate the different SCVIs by introducing different classes of SRs with different ranges of resources and constraints based on real-world data instead of the mix we have in our thesis, which is a weakness of our work. Furthermore, future research may go more into simulating SLAs where, i.e., maximum network latency is constrained and where the maximum response time to an SR is considered, as it is conceivable that these constraints can improve the realism of the simulation and the real-world performance of the resource allocation in terms of URLLC.

Future research might go into isolation levels in NFV and their impact and if the simulation of these levels can be more realistic by simulating how the containers and VMs will work or improving our definitions.

In our DRL training, we were not able to get a higher explained variance value than around 0.2-0.3, which seems to indicate that we do not fully realize the potential of PPO, as this explained variance could go up to one when the value function network is very good at predicting rewards. If this value could be increased by future work, this could reduce the training time needed and perhaps increase the solution performance.

Appendix A

PPO Training

A.1 Default Parameters

We begin testing using the default PPO parameters from the Stable Baselines 3 library. We will test parameters that greatly impact the training time towards the end to save time; this includes the epochs, steps, and batch size parameters.

- **learningTimesteps**: 10000
- **learning_rate**: 0.0003
- **n_epochs**: 10
- **n_steps**: 2048
- **batch_size**: 64
- **ent_coef**: 0.01
- **clip_range**: 0.1
- **gamma**: 0.99
- **gae_lambda**: 0.95
- **clip_range_vf**: 0.2
- **normalize_advantage**: true
- **vf_coef**: 0.5
- **max_grad_norm**: 0.5

A.2 Acceptance Reward

Firstly we will try two of our acceptance reward functions and state space variants to assess their viability because we assume that the reward function will significantly impact the final results and the training speed. The tables below show the parameters tested in blue and those we choose to continue testing in green. Note that in Figure A.1, Figure A.1 and Figure A.3, the scale of the LTRCR is not correct due to a mistake, but the relative difference between the graphs is still correct.

Experiment	**Acceptance Reward**	**Placement Reward**	**State Space**
1	RMEC	0.001	Distance
2	LTARCR	0.001	Distance
3	RMEC	0.001	Edge
4	Simple	0.001	Distance

Table A.1: 1-4

From Figure A.1, we compare the performance of the state space, comparing the reduced state space distance information (1 in purple) and the complete state space edge information (3 in orange). We see that the performance of the edge information is relatively lacking in every regard except for the LTRCR, where it is only slightly better. Using PPO on the larger edge information state space means learning is comparatively slower. Therefore, we will continue using the distance information state space.

Among the other experiments with the distance information state space, we see that it is the simple acceptance reward (4 in blue) and LTRCR (2 in green) that perform the best, however in terms of the LTRCR metric, the simple acceptance reward provides the best results by far, so we will use this reward function in the following experiments in combination with the distance information state space.

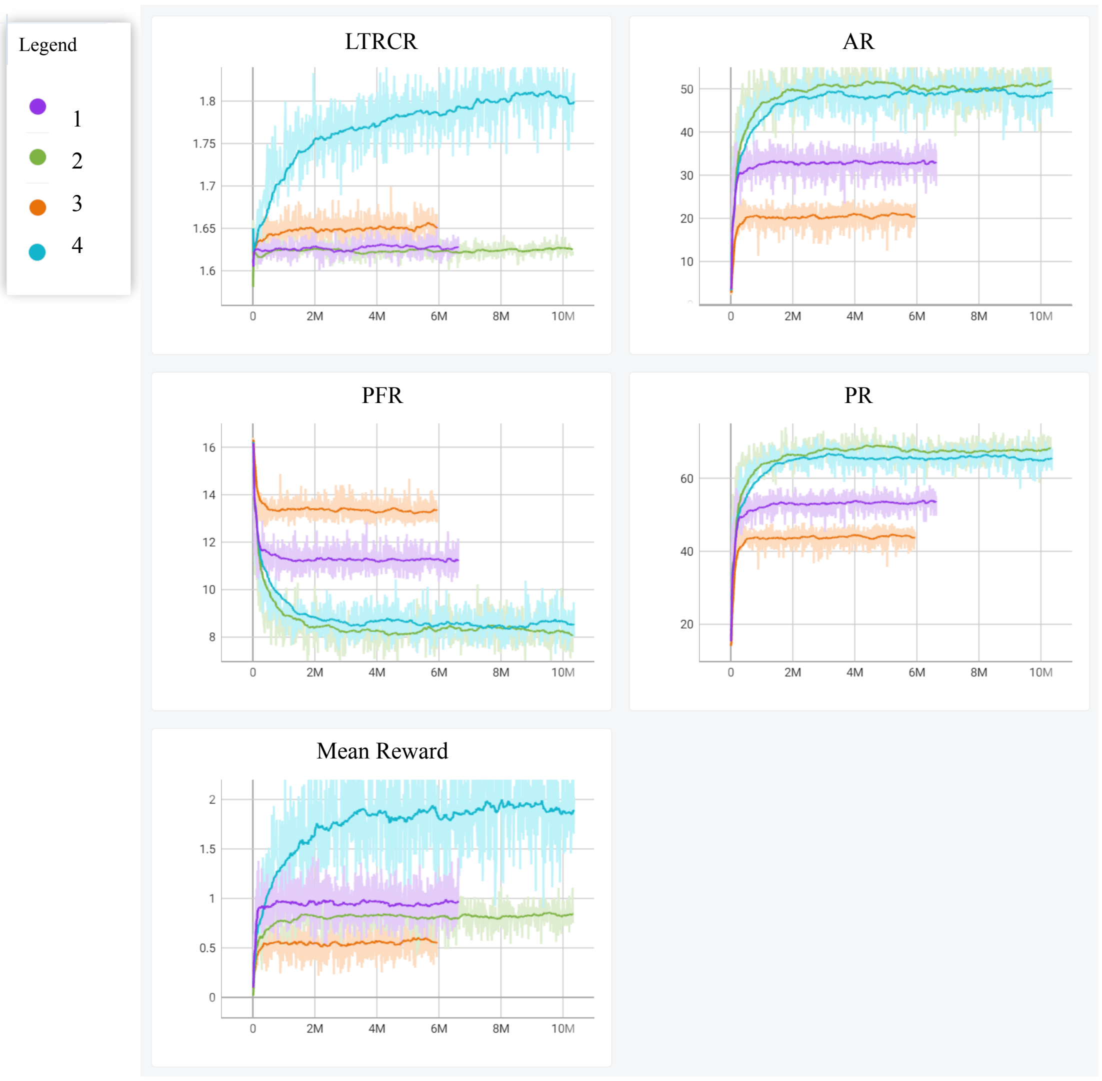

FIGURE A.1: 1-4

A.3 Placement Reward

We now try different placement rewards to increase our AR as the placement reward will incentivize placements and, therefore, acceptance of ARs.

Experiment	**Acceptance Reward**	**Placement Reward**	**State Space**
5	Simple	0.01	Distance
6	Simple	0.1	Distance
7	Simple	1.0	Distance
8	Simple	0.5	Distance

Table A.2: 6-9

In Figure A.2, we find that increasing the placement reward does increase the PR and AR, but this comes at the cost of the LTRCR. Using 1 for the placement reward (7 in purple) seems too high, and 0.5 (8 in green) is the sweet spot for the placement reward when combined with the simple reward.

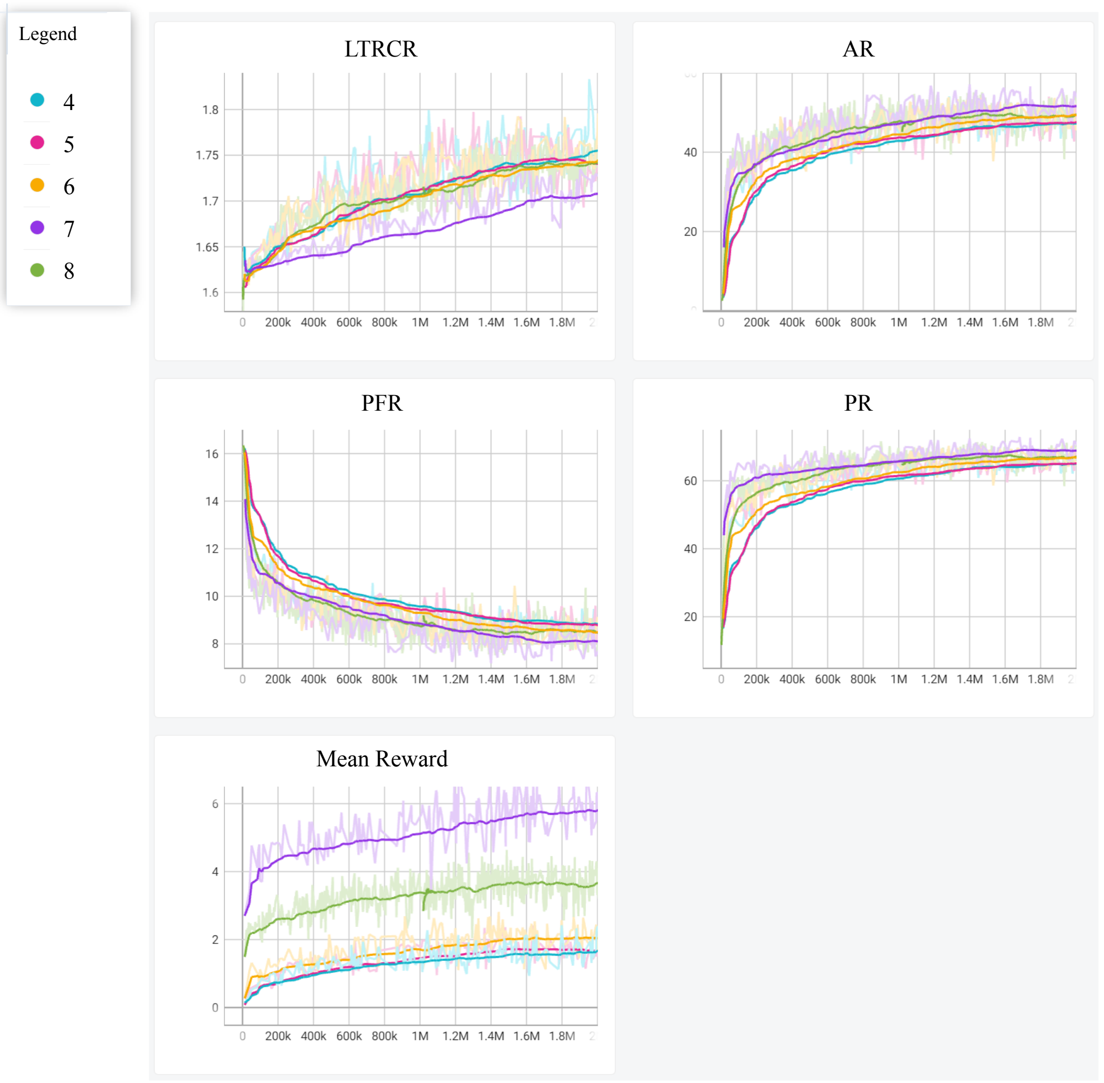
Legend
4
5
6
7
8
LTRCR
AR
PFR
PR
Mean Reward
200k
400k
600k
800k
1M
1.2M
1.4M
1.6M
1.8M

FIGURE A.2: 4-8

A.4 Hyperparameter Tuning

In the case of our hyperparameter tuning, we show the parameters we tried in Table A.3; however, we will only show the best training experiments marked in green compared to each other.

Exp.	**Learning Rate**	**Epochs**	**Steps**	**Batch Size**	**Ent Coef.**	**Clip Range**	γ	**Gae** λ	**Clip Range VF**	**VF Coef.**
9	0.0003	10	2048	64	0.01	0.1	0.99	0.93	0.2	0.5
10	0.0003	10	2048	64	0.01	0.1	0.99	0.97	0.2	0.5
11	0.0003	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
12	0.0003	10	2048	64	0.01	0.1	0.99	0.85	0.2	0.5
13	0.0003	10	2048	64	0.01	0.2	0.99	0.99	0.2	0.5
14	0.0003	10	2048	64	0.01	0.3	0.99	0.99	0.2	0.5
15	0.0003	10	2048	64	0.01	0.05	0.99	0.99	0.2	0.5
16	0.0003	10	2048	64	0.01	0.2	0.99	0.99	0.2	0.3
17	0.0003	10	2048	64	0.01	0.2	0.99	0.99	0.2	0.7
18	0.0001	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
19	0.0006	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
20	0.001	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
21	0.0002	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
22	0.00005	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
23	0.00001	10	2048	64	0.01	0.1	0.99	0.99	0.2	0.5
24	0.00005	10	512	64	0.01	0.1	0.99	0.99	0.2	0.5
25	0.00005	10	1024	64	0.01	0.1	0.99	0.99	0.2	0.5
26	0.00005	10	4096	64	0.01	0.1	0.99	0.99	0.2	0.5
27	0.00005	10	6144	64	0.01	0.1	0.99	0.99	0.2	0.5
28	0.00005	10	8192	64	0.01	0.1	0.99	0.99	0.2	0.5
29	0.00005	10	8192	32	0.01	0.1	0.99	0.99	0.2	0.5
30	0.00005	10	8192	128	0.01	0.1	0.99	0.99	0.2	0.5
31	0.00005	10	8192	256	0.01	0.1	0.99	0.99	0.2	0.5
32	0.00005	10	8192	512	0.01	0.1	0.99	0.99	0.2	0.5
33	0.00005	5	8192	256	0.01	0.1	0.99	0.99	0.2	0.5
34	0.00005	20	8192	256	0.01	0.1	0.99	0.99	0.2	0.5
35	0.00005	50	8192	256	0.01	0.1	0.99	0.99	0.2	0.5
36	0.00005	10	8192	256	0.001	0.1	0.99	0.99	0.2	0.5
37	0.00005	10	8192	256	0.005	0.1	0.99	0.99	0.2	0.5
38	0.00005	10	8192	256	0.2	0.1	0.99	0.99	0.2	0.5
39	0.00005	10	8192	256	0.3	0.1	0.99	0.99	0.2	0.5

TABLE A.3: Hyperparameter Testing

In Figure A.3, the overall performance is more promising after hyperparameter training (37 in purple), even if the training starts off slower.

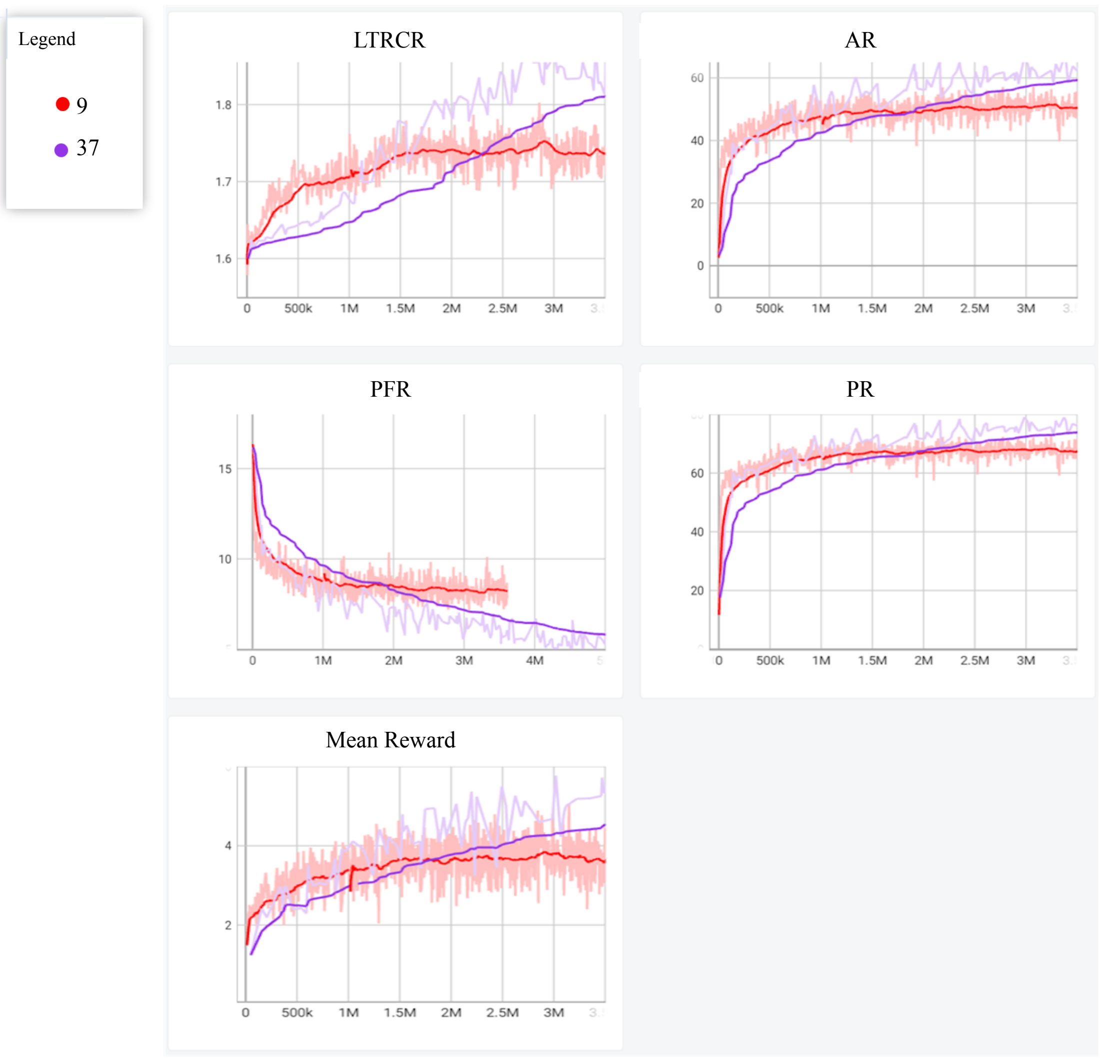

FIGURE A.3: 9 VS. 37

A.5 Revised Acceptance and Placement Reward Functions

Due to our experiences with the training thus far, we aim to try different acceptance reward functions that are similar but perhaps better or worse than our simple acceptance reward. The experiments are given in Table A.4.

Experiment	**Acceptance Reward**	**Placement Reward**
43	RMC	0.5
44	RCR	0.5
45	SRLtoSNL	0.5
46	LTRCR	0.5
47	RCR	0.1
48	SRLtoSNL	0.1
49	SRLtoSNL	0.05
50	SRLtoSNL	0.01

Table A.4: 43-50

From experiment 43-46 (Figure A.4) we find that $AccRew_{SRLtoSNL}$ and $AccRew_{RCR}$ give the best results, though it seems that all functions are viable except for $AccRew_{RMC}$, which only does well in the PR metric.

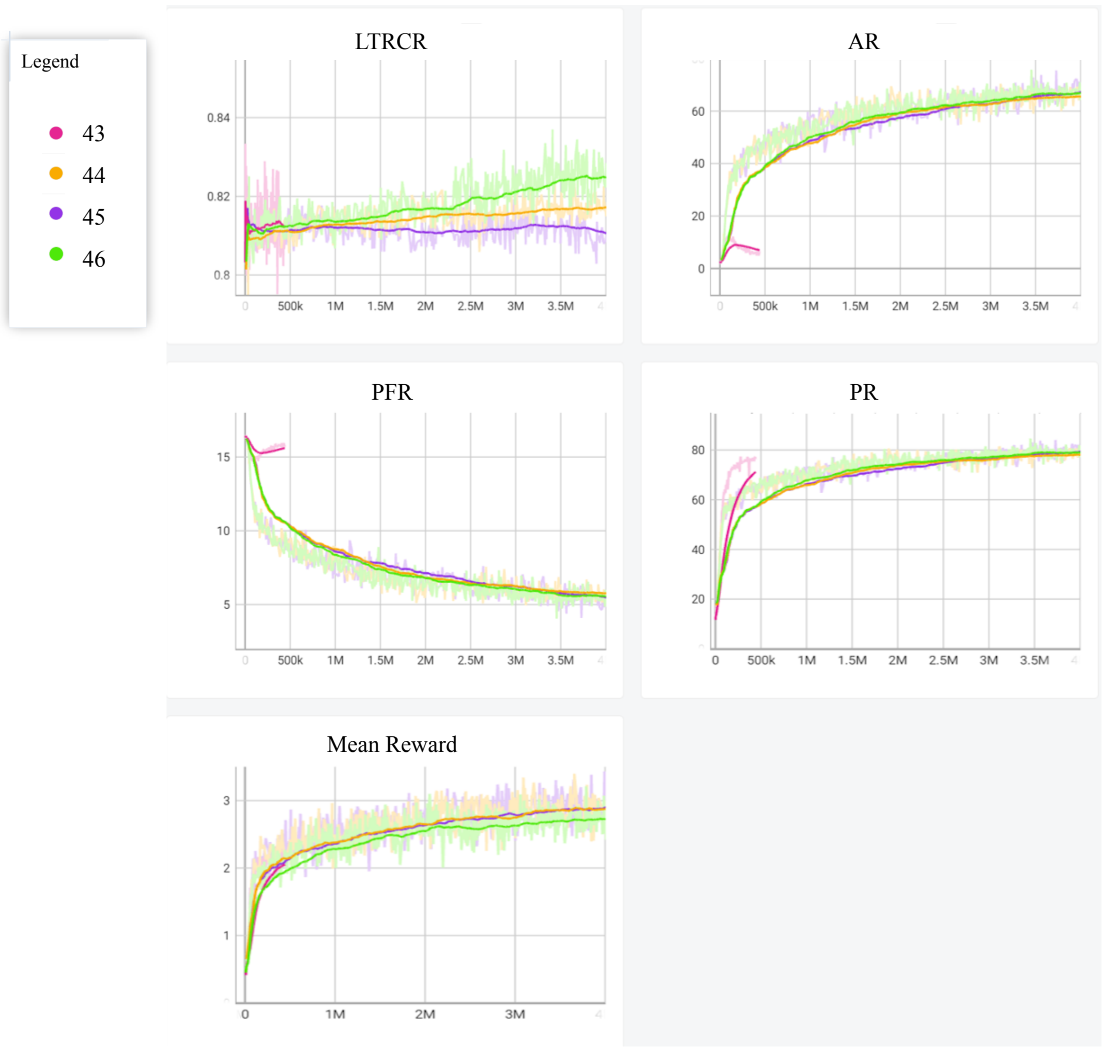

FIGURE A.4: 43-46

Trying different placement reward values for the best acceptance reward functions in experiment 47-50 (Figure A.5) gives quite a jump in learning performance and increases the values PPO converges to both in terms of AR and LTRCR in comparison to our best simple acceptance reward training (38 in purple).

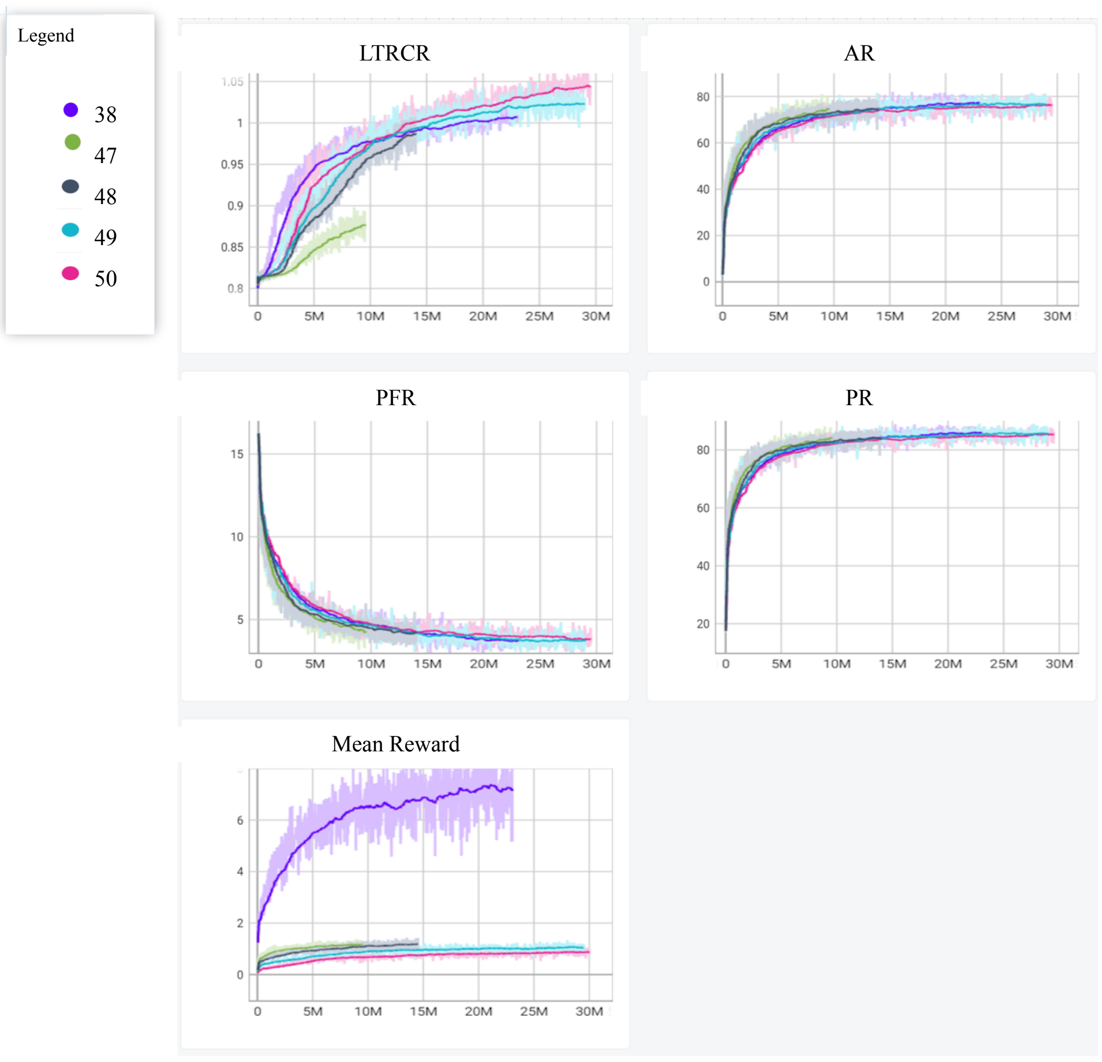

Figure A.5: 47-50

Bibliography

[1] Ali Gohar and Gianfranco Nencioni. The role of 5g technologies in a smart city: The case for intelligent transportation system. *Sustainability*, 13(9), 2021. ISSN 2071-1050. doi: 10.3390/su13095188. URL `https://doi.org/10.3390/su13095188`. Accessed: 2023-01-10.

[2] ETSI 3rd Generation Partnership Project (3GPP). System architecture for the 5g system (5gs). Technical Report 3GPP TS 123.501 version 16.6.0 Release 16, ETSI, November 2020. URL `https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf`. Accessed: 2023-01-10.

[3] Ali Gohar. Isolation Scheme for Virtual Network Embedding Based on Reinforcement Learning for Smart City Vertical Industries, November 2022. URL `http://arxiv.org/abs/2211.14158`. Issue: arXiv:2211.14158 arXiv:2211.14158 [cs].

[4] Andres J. Gonzalez, Jose Ordonez-Lucena, Bjarne E. Helvik, Gianfranco Nencioni, Min Xie, Diego R. Lopez, and Pal Gronsund. The Isolation Concept in the 5G Network Slicing. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 12–16, Dubrovnik, Croatia, June 2020. IEEE. ISBN 978-1-72814-355-2. doi: 10.1109/EuCNC48522.2020.9200939. URL `https://ieeexplore.ieee.org/document/9200939/`.

[5] Stan Wong, Bin Han, and Hans D. Schotten. 5G Network Slice Isolation. *Network*, 2(1):153–167, March 2022. ISSN 2673-8732. doi: 10.3390/network2010011. URL `https://www.mdpi.com/2673-8732/2/1/11`. Number: 1.

[6] Konstantinos Samdanis, Xavier Costa-Perez, and Vincenzo Sciancalepore. From network sharing to multi-tenancy: The 5G network slice broker. *IEEE Communications Magazine*, 54(7):32–39, July 2016. ISSN 0163-6804. doi: 10.1109/MCOM.2016.7514161. URL `http://ieeexplore.ieee.org/document/7514161/`.

[7] Frederico Schardong, Ingrid Nunes, and Alberto Schaeffer-Filho. NFV Resource Allocation: a Systematic Review and Taxonomy of VNF Forwarding Graph Embedding. *Computer Networks*, 185:107726, February 2021. ISSN 13891286.

doi: 10.1016/j.comnet.2020.107726. URL `https://linkinghub.elsevier.com/retrieve/pii/S1389128620313189`.

[8] Ultan Mulligan. Network Functions Virtualisation (NFV), 2022. URL `https://www.etsi.org/technologies/nfv%7D`.

[9] Ghasem Mirjalily and Zhiquan Luo. Optimal Network Function Virtualization and Service Function Chaining: A Survey. *Chinese Journal of Electronics*, 27(4):704–717, July 2018. ISSN 1022-4653, 2075-5597. doi: 10.1049/cje.2018.05.008. URL `https://onlinelibrary.wiley.com/doi/10.1049/cje.2018.05.008`. Number: 4.

[10] Yuxi Li. Deep Reinforcement Learning, October 2018. URL `http://arxiv.org/abs/1810.06339`. Issue: arXiv:1810.06339 Issue: arXiv:1810.06339 arXiv:1810.06339 [cs, stat].

[11] Nan He, Song Yang, Fan Li, Stojan Trajanovski, Fernando A. Kuipers, and Xiaoming Fu. A-DDPG: Attention Mechanism-based Deep Reinforcement Learning for NFV. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10, 2021. doi: 10.1109/iwqos52092.2021.9521285.

[12] Jie Sun, Yi Zhang, Feng Liu, Huandong Wang, Xiaojian Xu, and Yong Li. A survey on the placement of virtual network functions. *Journal of Network and Computer Applications*, 202:103361, June 2022. ISSN 10848045. doi: 10.1016/j.jnca.2022.103361. URL `https://linkinghub.elsevier.com/retrieve/pii/S1084804522000285`. 2 citations (Crossref) [2022-12-30].

[13] Lilian Weng. Policy Gradient Algorithms, April 2018. URL `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`. Section: posts.

[14] Sara Ayoubi, Samir Sebbah, and Chadi Assi. A Logic-Based Benders Decomposition Approach for the VNF Assignment Problem. *IEEE Transactions on Cloud Computing*, 7(4):894–906, October 2019. ISSN 2168-7161. doi: 10.1109/TCC.2017.2711622. Conference Name: IEEE Transactions on Cloud Computing.

[15] Zichuan Xu, Wanli Gong, Qiufen Xia, Weifa Liang, Omer F. Rana, and Guowei Wu. NFV-Enabled IoT Service Provisioning in Mobile Edge Clouds. *IEEE Transactions on Mobile Computing*, 20(5):1892–1906, May 2021. ISSN 1558-0660. doi: 10.1109/TMC.2020.2972530. Conference Name: IEEE Transactions on Mobile Computing.

[16] Weilin Zhou, Yuan Yang, Mingwei Xu, and Hao Chen. Accommodating Dynamic Traffic Immediately: A VNF Placement Approach. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2019. doi: 10.1109/ICC.2019.8761554. ISSN: 1938-1883.

[17] Shaoyang Wang, Chau Yuen, Wei Ni, Yong Liang Guan, and Tiejun Lv. Multiagent Deep Reinforcement Learning for Cost- and Delay-Sensitive Virtual Network Function Placement and Routing. *IEEE Transactions on Communications*, 70(8): 5208–5224, August 2022. ISSN 1558-0857. doi: 10.1109/TCOMM.2022.3187146. Conference Name: IEEE Transactions on Communications.

[18] Jun Cai, Zhongwei Huang, Liping Liao, Jianzhen Luo, and Wai-Xi Liu. APPM: Adaptive Parallel Processing Mechanism for Service Function Chains. *IEEE Transactions on Network and Service Management*, 18(2):1540–1555, June 2021. ISSN 1932-4537. doi: 10.1109/TNSM.2021.3052223. Conference Name: IEEE Transactions on Network and Service Management.

[19] Junzo Watada, Arunava Roy, Ruturaj Kadikar, Hoang Pham, and Bing Xu. Emerging Trends, Techniques and Open Issues of Containerization: A Review. *IEEE Access*, 7:152443–152472, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2945930. URL `https://ieeexplore.ieee.org/document/8861307/`.

[20] Jason Gerend. Containers vs. virtual machines, Oct 2021. URL `https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm`.

[21] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1–9, April 2014. doi: 10.1109/INFOCOM.2014.6847918. ISSN: 0743-166X.