UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

INTENT-BASED RADIO RESOURCE SCHEDULING IN RAN SLICING SCENARIOS USING REINFORCEMENT LEARNING

Cleverson Veloso Nahum

TD: 19/2024

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2024

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Cleverson Veloso Nahum

INTENT-BASED RADIO RESOURCE SCHEDULING IN RAN SLICING SCENARIOS USING REINFORCEMENT LEARNING

TD: 19/2024

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2024

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

N153i Nahum, Cleverson Veloso.
Intent-based Radio Resource Scheduling in Ran Slicing Scenarios using Reinforcement Learning / Cleverson Veloso Nahum. — 2024.
133 f. : il. color.

Orientador(a): Prof. Dr. Aldebaro Klautau
Tese (Doutorado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2024.

1. Alocação de recurso de rádio. 2. Fatiamento da rede de acesso. 3. Redes baseadas em intenções. 4. Aprendizado por reforço. 5. Redes móveis. I. Título.

CDD 384

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Cleverson Veloso Nahum

INTENT-BASED RADIO RESOURCE SCHEDULING IN RAN SLICING SCENARIOS USING REINFORCEMENT LEARNING

Thesis submitted to the examination committee in the graduate department of Electrical Engineering at the Federal University of Pará in partial fulfillment of the requirements for the degree of Doctor of Science in Electrical Engineering with emphasis in Telecommunications.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2024

INTENT-BASED RADIO RESOURCE SCHEDULING IN RAN SLICING SCENARIOS USING REINFORCEMENT LEARNING

Tese submetida à avaliação da banca examinadora aprovada pelo colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará e julgada adequada para prosseguimento trabalho para obtenção do Grau de Doutor em Engenharia Elétrica na área de Telecomunicações.

Aprovada em 04 / 11 / 2024

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior

ORIENTADOR

Prof. Dr. Leonardo Lira Ramalho

MEMBRO INTERNO - PPGEE

Prof. Dr. Kleber Vieira Cardoso

MEMBRO EXTERNO - UFG

Prof. Dr. Tarcisio Ferreira Maciel

MEMBRO EXTERNO - UFC

Dr. Pedro dos Santos Batista

MEMBRO EXTERNO - ERICSSON

Prof. Dr. Diego Lisboa Cardoso

DIRETOR DA PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA/UFPA

Agradecimentos

Eu gostaria de deixar por escrito meus sinceros agradecimentos a todos os meus familiares, amigos e colegas de trabalho pelo suporte oferecido durante essa jornada que o doutorado representa. Ao longo desses quatro anos, experimentei os mais variados sentimentos de felicidade, tristeza, falha e sucesso que a pesquisa pode proporcionar, e agora um pouco mais experiente, percebo com clareza a importância de estar cercado por pessoas que estejam ao seu lado em todos esses momentos.

Gostaria de agradecer profundamente ao meu orientador, Aldebaro Klautau, por me acompanhar desde a graduação e ter sido não só uma grande referência acadêmica e profissional, mas um amigo cujos conselhos foram muito valiosos, apesar da sua clara inabilidade em escolher times de futebol para torcer. Estendo a gratidão a todos do laboratório LASSE pelo suporte e ajuda nas pesquisas, e principalmente pelas conversas sobre os mais variados temas na copa. Gostaria de agradecer em especial à Kelly Pereira pelo carinho e dedicação na administração do laboratório e pelas incontáveis vezes que me ajudou. O LASSE é uma família.

Quero agradecer aos meus amigos e familiares por todo amor e apoio que tornaram essa conquista possível. Em especial, a minha amada esposa, Amanda Guedelha, meus pais, Cleonildo e Ilza, minha irmã, Jéssica, e avós, Manoel e Beatriz. Incontáveis foram os sacrifícios de vocês para que o meu sonho se tornasse possível. Devo tudo a vocês.

Sou grato à banca examinadora pelo tempo investido na avaliação deste trabalho e pelas contribuições.

Por fim, agradeço ao apoio e financiamento da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Cleverson Veloso Nahum

Outubro, 2024

Do rio que tudo arrasta se diz que é violento. Mas ninguém diz violentas as margens que o comprimem.

Bertolt Brecht

List of Acronyms

A2C advantage actor critic

A3C asynchronous advantage actor-critic

BE best-effort

B5G beyond 5G

CDF cumulative distribution function

CQI channel quality information

CSI channel state information

DDPG deep deterministic policy gradient

DQN deep Q-network

DRL deep reinforcement learning

EDF earliest deadline first

eMBB enhanced mobile broadband

EURA enhanced utilization resource allocation

GBR guaranteed bit rate

IaaS infrastructure as a service

IBS intent-based system

IETF internet engineering task force

KPI key performance indicator

LOS line-of-sight

LSTM long short-term memory

LTE long term evolution

MAC medium access control

MCS modulation coding scheme

MARL multi-agent reinforcement learning

MARR multi-agent round-robin

MAPF multi-agent proportional fair

Mbps megabits per step

MDP Markov decision process

MIMO multiple-input and multiple-output

MTC machine-type communication

mMTC massive machine-type communication

MT maximum throughput

NLOS non-line-of-sight

NR new radio

OFDM orthogonal frequency division multiplexing

PF proportional fair

POMDP partially observable Markov decision process

PPO proximal policy optimization

QCI QoS class identifier

QoS quality of service

RAN radio access network

RB resource block

RBG resource block group

RFC request for comments

RL reinforcement learning

RR round-robin

RRM radio resource management

RRS radio resource scheduling

RSRP reference signal received power

SAC soft actor-critic

SE spectral efficiency

SISO single-input and single-output

SLA service-level agreement

SNR signal-to-noise ratio

SSR SLA satisfaction rate

TDD time division duplex

TRPO trust region policy optimization

TTI transmission time interval

UE user equipment

ULA uniform linear array

URLLC ultra-reliable low latency communication

VNF virtual network function

VoLTE voice over LTE

ZSM zero-touch network and service management

List of Figures

List of Tables

Contents

Abstract

Network slicing at the radio access network (RAN) domain, called RAN slicing, requires elasticity, efficient resource sharing, and customization to deal with scarce and limited frequency spectrum resources while fulfilling the slice intents in an intent-based system. In this scenario, radio resource scheduling is an essential function to provide the resource management needed to prevent intent violations, hence providing sufficient radio resources for RAN slices to accomplish their intents. The wide variety of scenarios supported in 5G and beyond 5G (B5G) networks makes the radio resource scheduling (RRS) problem in RAN slicing scenarios a significant challenge. This thesis proposes an intent-based RRS for RAN slicing using reinforcement learning (RL) to fulfill the slice intent. The proposed method aims to prevent intent violations by making the management of resource block groups (RBGs) available between slices and users' equipment (UEs) using inter-slice and intra-slice schedulers, respectively. This thesis also proposes investigating a slice prioritization structure to ensure the intent of more important slices when the available radio resources are insufficient to guarantee all slice's intents. This thesis proposal presents results obtained using an intent-based RRS with RL for a fixed number of slices and also for multiple network scenarios, aiming to demonstrate the importance of intent-based RRS design for scenarios with RAN slicing. The proposed method outperformed the baselines in fixed and multiple network scenarios, protecting high-priority slices and minimizing the total number of violations.

***Keywords* —** Radio resource scheduling, RAN slicing, intent-based scheduling, reinforcement learning, mobile networks.

Resumo

O fatiamento da rede móvel no domínio da rede de acesso requer elasticidade, compartilhamento de recursos de forma eficiente e customização para lidar com a escassez e limitação dos recursos de rádio enquanto cumpre as intenções das fatias de rede definidas em um contrato de nível de serviço. Nesse cenário, o alocador de recursos de rádio é essencial para prover a administração de recursos a fim de prevenir as violações de intenções de rede, e consequentemente oferecer recursos de rádio suficientes para as fatias de rede de acesso cumprirem seus objetivos. A grande variedade de cenários suportados nas redes 5G e pós-5G torna o problema da alocação de recursos de rádio em cenários de fatiamento da rede de acesso ainda mais desafiador. Essa tese propõe investigar um alocador de recursos de rádio baseado nas intenções das fatias de rede de acesso, utilizando aprendizado por reforço para cumprir as intenções de rede. O método proposto tem por objetivo prevenir as violações de intenções de rede através da administração de recursos de rádio disponíveis entre as fatias de rede de acesso e usuários usando um alocador de recursos de rádio entre as fatias de rede e outro para os usuários dentro da fatia de rede. Esta tese também descreve uma estrutura para priorização de fatias de rede para assegurar os requisitos definidos nas intenções de rede para as fatias mais importantes quando os recursos de rádio não são suficientes para garantir todas as intenções de rede requisitadas. Esta tese apresenta os resultados obtidos usando um alocador de recursos de rádio baseado nas intenções das fatias de rede de acesso, utilizando aprendizado por reforço para um número fixo de fatias de rede e também para múltiplos cenários de rede para evitar violações de intenções de rede, e demonstra a importância de um alocador de recursos de rádio baseado nas intenções das fatias de rede em cenário com fatiamento da rede de acesso. O método proposto apresentou melhor desempenho em comparação aos métodos da literatura avaliados tanto na proteção de slices prioritários quanto na minimização do número total de violações.

Palavras-chave — Alocação de recursos de rádio, fatiamento da rede de acesso, alocação baseada em intenções, aprendizado por reforço, redes móveis.

Chapter 1

Introduction

6G networks will support various applications thanks to new technologies and architectures designed to improve network capacity and provide higher throughput, lower latency, and increased reliability [1]. Some applications that will benefit from 6G are smart healthcare, smart cities, extended reality, virtual reality, holographic communication, and cloud gaming [1–3]. Technologies such as network slicing, artificial intelligence, and advanced resource allocation techniques are key enablers and are essential for providing network functionality to meet application requirements while improving resource utilization efficiency [1, 2]. The network has the complex task of distributing the available resources among various applications, each with different requirements while improving resource utilization efficiency and providing guarantees of service-level agreement (SLA) fulfillment.

Network slicing provides service customization, isolation, and multi-tenancy support on shared physical network infrastructure, enabling logical and physical separation of network resources. Therefore, 5G and beyond 5G (B5G) systems are capable of deploying different configurations and structures for each specific service by creating slices on demand and changing their functions online [4]. End-to-end network slicing involves the radio access network (RAN), transport network, and the core network domains, providing computational and network resource management. RAN slicing (network slicing at the RAN domain) requires elasticity, efficient resource sharing, and customization to manage scarce and limited frequency spectrum resources [4]. One of the multiple RAN slicing functions is to deploy a radio resource scheduling (RRS) to allocate radio resources among slice instances (called *inter-slice* scheduling). Another RRS deals with resources within slice instances (named *intra-slice* scheduling) [5]. The intra-slice RRS allocates the radio resources assigned by the inter-slice RRS to the users' equip-

ment (UEs). The inter- and intra-slice RRS schedulers must distribute the radio resources to fulfill the slices' intents.

RRS usually presents a high complexity in network slice scenarios in 5G and B5G networks due to the wide variety of network structures and application requirements [5]. RRS must work in various scenarios to enable radio resource management that meets the requirements of the slices. Therefore, new RRS methods should (i) attend to different types of slice with different requirements, (ii) provide sufficient radio resources for each slice to fulfill its network intents, and (iii) prioritize the most important slice intents when the amount of available radio resources is not sufficient to satisfy all slices' requirements.

The RAN is a data-rich environment where data is continuously gathered in the form of radio measurements or other system observations by user devices and network entities. In this context, machine learning is a key enabler in handling large data sets, which presents an opportunity to give data a central role in wireless networking [6]. Therefore, machine learning techniques such as reinforcement learning (RL) are up-and-coming to learn from network data and create flexible policies to deal with the wide variety of RRS scenarios that 5G and B5G hold [7].

The RRS needs to be aware of the diverse requirements of the different 6G applications. Then, it is the responsibility of the RRS to allocate the radio resources required to meet these application requirements. The slice consumer declares the communication service requirements to the operator in an SLA through network performance attributes, such as throughput, latency, and reliability requirements [8]. Another alternative is to declare these communication service requirements in network slice intents in an intent-based system. An intent-based system handles intents via a closed-loop process where intents formally specify requirements, goals, and constraints given to a technical system [9]. Using the RRS as an intent-based system, slice intents (requirements and goals) can be provided via a common intent model [10]. In this context, the intent-based RRS is responsible for allocating radio resources to fulfill the received intents. Instead of defining the RRS policy to deal with a specific group of slices, intent-based RRS receives the intents and implements a policy to fulfill the requested intents. It is possible to add more slices and applications to the system by specifying or updating intents, and the intent-based RRS is expected to automatically adapt its policy to meet the intents of the new slice.

When considering RRS methods for RAN slicing scenarios, data-driven approaches have

gained increasing attention due to their ability to directly build knowledge about the network from data without the need for statistical models of the system [11]. Machine learning techniques, particularly RL, can learn from network data and create flexible policies to deal with the wide variety of RRS scenarios in B5G networks [6, 7]. There are different RRS methods using RL for RAN slicing [12–19]. The methods presented in [12–14] focus on maximizing/minimizing specific network metrics, such as maximizing the slice throughput or minimizing the transmission delay. These works, however, do not consider the case of minimum performance guarantees and, therefore, are incompatible with an intent-based system as they cannot fulfill specific network requirements. Related works [15–19] have an SLA satisfaction rate (SSR) approach in which the network slice objectives are specified. However, they do not provide intent prioritization mechanisms or track intent drift to avoid future intent violations.

Additionally, the works in [12–19] do not generalize well to diverse network scenarios. Specifically, they define a group of slice types, usually enhanced mobile broadband (eMBB), ultra-reliable low latency communication (URLLC), and massive machine-type communication (mMTC), and design RRSs that are trained to handle specific network conditions (e.g., channel conditions, network load, traffic profiles). Moreover, the authors do not evaluate the performance of their methods when dealing with previously unseen and different network scenarios and do not provide clear guidelines on how to tackle scenarios that go beyond those considered in these works or how to consider varying numbers of active slices, UEs, and channel characteristics. However, when proposing an RRS method for RAN slicing to B5G networks, it is essential to assess the method's capacity to generalize or be utilized for different network scenarios. This thesis focuses on developing a single RRS method that is able to generalize well across different network scenarios to provide a solution for production cellular networks.

This thesis investigates the usage of intent-based RRS methods for RAN slicing using RL with slice prioritization, using different slice types and quality of service (QoS) intents based on throughput, latency, and packet loss rate. This thesis is organized into two investigations written in self-contained Chapters 3 and 4. The former investigates the use of an intent-aware RRS method for RAN slicing scenarios with eMBB, URLLC, and best-effort (BE) slice types with variations in channel trajectories experienced by UEs in each RL episode. The proposed RL agent implements an inter-slice scheduler with optimized predefined weights to prioritize slice intents. The intra-slice scheduler utilizes a round-robin algorithm. It defines different traffic models and varies the magnitude of traffic for each UE according to its slice type, assessing

the RL agent capacity to generalize its decision for different network conditions. This chapter investigates the proposed method's capacity to generalize for different channel and traffic conditions when considering regular slice types eMBB, URLLC, and BE.

In Chapter 4, this thesis presents an enhanced intent-based RRS using multi-agent reinforcement learning (MARL) where the generalizability is evaluated in different network scenarios with multiple slice types. The intent-based RRS uses an RL agent to inter-slice scheduling and MARL with shared parameters to the intra-slice scheduler. The proposed method is evaluated in various network scenarios with different numbers of active slices, slice types, channel trajectories, number of UEs, and UE characteristics. It approaches the capacity of the proposed method and baselines of dealing with various network scenarios when trained and tested in the same network scenario, when training and testing in different network scenarios, and finally when using transfer learning to improve performance in specific network scenarios.

1.1 Objectives

1.1.1 General objectives

Formulate an intent-based RRS using RL focusing on inter-and intra-slice allocation to fulfill slice intents in an intent-based network, providing support to distinct slice types with their requirements and priorities. The proposed method should avoid intent violations and prioritize critical slices when available resources are insufficient to meet all the slice's intents.

1.1.2 Specific objectives

The specific objectives to be approached in this thesis are as follows.

1. Discuss the RRS challenges in future networks in scenarios with RAN slicing.
2. Study of RRS proposals for RAN slicing scenarios emphasizing reinforcement learning solutions, with their challenges and opportunities.
3. Create an RL environment for RRS validation in scenarios with RAN slicing to facilitate the deployment of different RL techniques for comparison.
4. Design the RRS method with support for different intents for each slice, with the flexibility to change its requirements in real-time, promoting the necessary network adaptations

to meet intents still.

5. Investigate important network data features for RRS and alternatives to decrease the number of inputs in the RL agent since there is a massive amount of information available in the RAN.

6. Investigate the generalizability of the proposed RRS methods and baselines methods for different network scenarios with various numbers of active slices, types of slices, channel trajectories, number of UEs, and UE characteristics.

1.2 Outline

This thesis proposal is organized as follows: Chapter 2 presents the fundamental concepts for understanding the proposed method to RRS with RAN slicing, such as the RRS process in 5G networks, network slicing, intent-based networks, and RL algorithms. Chapter 3 presents the intent-aware RRS using RL in the inter-slice scheduler in a scenario with a fixed number of slices and UEs, providing results and discussions about the slice's intents fulfillment. It considers only eMBB, URLLC, and BE slice types. Chapter 4 presents a proposal for improved intent-based RRS method using MARL for both inter-and intra-slice scheduling. It investigates the generalizability of different network scenarios. Finally, Chapter 5 concludes the thesis proposal, summarizing the results presented and future work.

1.3 Outcomes

The following list summarizes the papers produced as part of this thesis effort.

Journal Papers:

1. **Nahum, C.V.**, Lopes, V.H.L., Dreifuerst, R.M., Batista, P., Correa, I., Cardoso, K.V., Klautau, A. and Heath, R.W., 2023. Intent-aware radio resource scheduling in a RAN slicing scenario using reinforcement learning. IEEE Transactions on Wireless Communications, 23(3), pp.2253-2267.

2. **Nahum, C.**, Ramalho, L., Klautau, A., Medeiros, E., Almeida, I. and Trojer, E., 2021. Functional Split and Frequency-Domain Processing for Fronthaul Traffic Reduction. IEEE Communications Letters, 25(8), pp.2758-2762.

3. Lopes, V.H.L., **Nahum, C.V.**, Dreifuerst, R.M., Batista, P., Klautau, A., Cardoso, K.V. and Heath, R.W., 2022. Deep reinforcement learning-based scheduling for multiband massive MIMO. IEEE Access, 10, pp.125509-125525.

4. Both, C.B., Borges, J., Gonçalves, L., **Nahum, C.**, Macedo, C., Klautau, A. and Cardoso, K., 2022. System intelligence for UAV-based mission critical with challenging 5G/b5G connectivity. ITU Journal on Future and Evolving Technologies, v.3, pp.359-373.

5. Correa, I., Oliveira, A., Du, B., **Nahum, C.**, Kobuchi, D., Bastos, F., Ohzeki, H., Borges, J., Mehta, M., Batista, P. and Kondo, R., 2022. Simultaneous beam selection and users scheduling evaluation in a virtual world with reinforcement learning. ITU Journal on Future and Evolving Technologies, 3(2), pp.202-213.

6. Blessed, G., Aliyu, I., Agajo J., Sarmento, T., **Nahum, C.**, et al., 2022. Network resource allocation for emergency management based on closed-loop analysis. ITU Journal on Future and Evolving Technologies, 3(2).

Conference Papers:

1. **Nahum, C.**, Mauricio, W., Silva, M., Takeda, M. and Klautau, A., 2023, November. Safeguard URLLC Services in Industry 4.0 Through Reinforcement Learning Scheduling. In 2023 Workshop on Communication Networks and Power Systems (WCNPS) (pp. 1-7). IEEE.

2. **Nahum, C.**, Mauricio, W.V., Silva, M., Facina, M.S., Takeda, M.Y. and Klautau, A., Reinforcement Learning Scheduling for URLLC Service Protection in Industry 4.0 Scenario. In 2023 Brazilian Symposium on Telecommunications and Signal Processing (SBrT).

3. Borges, J.P.T., De Oliveira, A.P., Bastos, F.H.B.E., Suzuki, D., Junior, E.S.D.O., Bezerra, L.M., **Nahum, C.V.**, dos Santos Batista, P. and Júnior, A.B.D.R.K., 2021, December. Reinforcement learning for scheduling and MIMO beam selection using Caviar simulations. In 2021 ITU Kaleidoscope: Connecting Physical and Virtual Worlds (ITU K) (pp. 1-7). IEEE.

4. Campos, D., de Almeida, G.M., Junior, W.T., **Nahum, C.V.**, Klautau, A., Abdel-Rahman, M.J. and Cardoso, K.V., Stepwise Optimal Inter-Slices Radio Resource

Scheduling for Service-Level Agreement Assurance. In 2024 Brazilian Symposium on Computer Networks and Distributed Systems (SBRC).

5. Aben-Athar, R., **Nahum, C.**, da Silva Brilhante, D., de Rezende, J.F., Mendes, L. and Klautau, A., User Scheduling and Beam-Selection with Tabular and Deep Reinforcement Learning. In 2022 Brazilian Symposium on Telecommunications and Signal Processing (SBrT).
6. Nóvoa, L., Tavares, V., **Nahum, C.**, Lins, S. and Klautau, A. Middleware implementation for RYU SDN Controller to manage switches in a C-RAN scenario. In 2021 Integrated Software and Hardware Seminar. SBC.

Extended Abstracts:

1. Costa, L., Aben-Athar, R., **Nahum, C.**, Gonçalves, G., Correa, I. and Klautau, A., Near-RT RIC and Emulated Basestation for Initial xApps Development. In 2023 Brazilian Symposium on Telecommunications and Signal Processing (SBrT).
2. Matni, L., Aben-Athar, R., **Nahum, C.**, Gonçalves, G., Correa, I., Lins, S. and Klautau, A., Implementing Service Management and Orchestration for ORAN Software Community. In 2023 Brazilian Symposium on Telecommunications and Signal Processing (SBrT).

Chapter 2

Theoretical foundations

This chapter presents the main fundamental concepts of the thesis to facilitate the understanding process through the following chapters. It explains the RRS process in 5G networks and some existing scheduling mechanisms available to be used in scheduler algorithms. It defines the concept of network slicing, focusing on RRS to clarify the differences between RRS with and without RAN slicing. It explains the intent-based networks and establishes the motivation for an intent-based RRS method. Finally, it presents the RL technique focusing on proximal policy optimization (PPO) and soft actor-critic (SAC) methods.

2.1 Radio resource scheduling

A key characteristic of mobile radio communication is the significant and usually rapid variations in instantaneous channel conditions caused by frequency-selective fading, distance-dependent path loss, and random interference variations due to transmissions in other cells and by different devices. Channel-dependent scheduling exploits these variations by dynamically sharing time-frequency resources between users. Dynamic scheduling is used in long term evolution (LTE), and on a high level, the new radio (NR) scheduling framework is similar to the one in LTE. Therefore, the base station scheduler takes allocation decisions based on channel quality reports obtained from network devices, also taking into account different traffic priorities and quality of service requirements when forming the scheduling decisions to be sent to the scheduled devices [20, Subsection 5.1].

The scheduler is part of the medium access control (MAC) layer and controls the assignment of uplink and downlink resources in terms of so-called resource blocks (RBs) in the

frequency domain and orthogonal frequency division multiplexing (OFDM) symbols and slots in the time domain. The essential operation of the scheduler is dynamic scheduling, with the base station taking the scheduling decisions, usually once per slot, and sending scheduling information to the selected devices. The uplink and downlink scheduling are separated in NR. Hence, their decisions are independent of each other [20, Subsection 6.4].

Fig. 2.1 illustrates the downlink and uplink scheduling process. The downlink schedule controls the device transmission and the set of RBs upon which the device's information should be transmitted. The transport format selection comprises the transport block size, modulation scheme, and antenna mapping. The base station controls the logical channel multiplexing for downlink transmissions. The uplink scheduler works similarly but controls which devices should transmit on which uplink RBs. In the uplink, the device is responsible for selecting from which radio bearer(s) the data are taken according to a set of rules the base station can configure. The scheduling strategies are implementation-specific and not specified by 3GPP. However, the overall goal of most schedulers is to use the channel variation information, obtained through channel state information (CSI), between devices as an advantage by scheduling transmissions to devices on resources with advantageous channel conditions in both the time and frequency domains [20, Subsection 6.4].

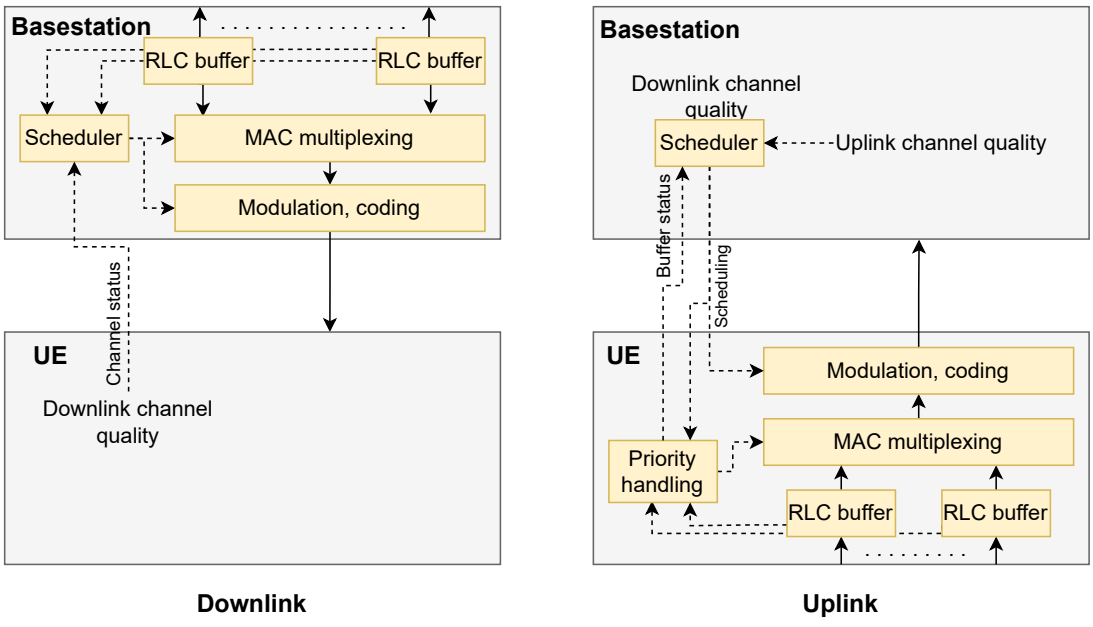

Figure 2.1: Downlink and uplink scheduling process in NR communications.

Only a set of supporting mechanisms are standardized, on top of which a vendor-specific

scheduling strategy is implemented since the NR does not standardize scheduling behavior. Therefore, the information that the scheduler needs depends on the specific scheduling strategy implemented. However, most schedulers need information on at least the channel conditions on the device (CSI) and buffer status [20, Subsection 14.1].

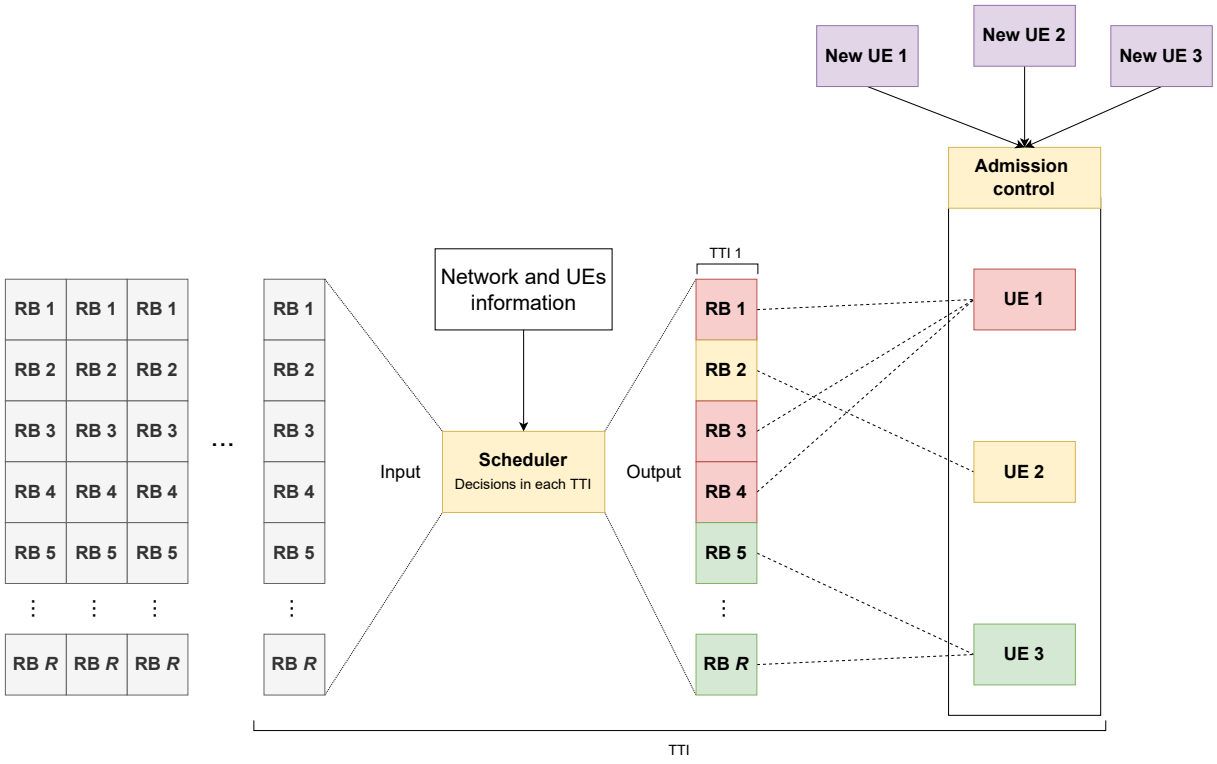

Figure 2.2: General view of scheduling and admission control process emphasizing the scheduler decision at each TTI.

Fig. 2.2 shows a general view of a base station scheduling process valid for the downlink and uplink. Since radio resources are limited, an admission control module handles new device requests. It evaluates if the network can provide a minimum service level to the new device without harming other UEs connections [21]. If the admission control decides that the network cannot provide radio resources to new devices, it should deny the network entry of UEs. In each transmission time interval (TTI), the scheduler is responsible for distributing the available RBs between connected UEs. Considering an OFDM system, there is only one UE allocated per RB. The scheduler distributes the RBs following the scheduler's design and the network status. For example, a scheduler that aims to maximize total system throughput should prioritize allocating UEs with higher spectral efficiency (SE) and incoming traffic. Therefore, the base station scheduler functions depend on the network system goals defined by the network operators.

There are several categories of algorithms for radio resource scheduling in the literature, each containing a set of algorithms with common characteristics, such as QoS-unaware and QoS-aware schedulers.

2.1.1 QoS-unaware methods

The QoS-unaware schedulers aim to maximize/minimize general system metrics, such as maximizing the total system throughput, minimizing the total latency, or increasing the network fairness in the allocation process without focusing on fulfilling UE's QoS requirements. Some typical examples of QoS-unaware algorithms are the round-robin (RR), maximum throughput (MT), and proportional fair (PF) methods.

RR is largely used in radio resource allocation because of its simplicity and low complexity, focusing mainly on increasing fairness between users. This algorithm allocates the same amount of radio resources per user. When the number of RBs is smaller than the number of users, it alternates the number of users served per time to provide all users with the same amount of RBs along with time. This strategy lacks spectral efficiency and throughput performance since it does not consider the CSI information when performing the allocation process [22].

MT algorithm aims to maximize the system achieved throughput by allocating resources to the UEs with higher SE values and high incoming traffic. The maximum throughput algorithm does not take into account fairness between users, always prioritizing the UEs with higher SE values [22]. PF algorithm aims to increase the system achieved throughput while providing a minimum level of service for all users, increasing the system's fairness. It considers a maximum throughput calculation divided by the average throughput of each UE, so when the average throughput of a given UE is small, the proportional fair should increase the chances of allocating this UE. When the average throughput discrepancy is not high, the method should prioritize UEs with higher throughput capacities [22].

Due to the QoS-unaware methods simplicity, they do not meet the increased requirement that arose at 5G to provide individual QoS requirements fulfillment to the maximum number of UEs in the mobile network.

2.1.2 QoS-based methods

The QoS-based algorithms aim to fulfill individual UEs requirements defined by QoS requirements, enabling different scheduling treatments for various applications. For example, a video streaming application should be differentiated from a best-effort application (web browser traffic and email services) providing higher throughputs for the former and more relaxed performance for the best-effort applications since it has elastic requirements [5]. The fulfillment of the UE's QoS requirements gained attention since LTE with QoS-aware RRS designed to maximize specific utility functions, which attempts to guarantee QoS for the maximum possible number of users [23]. LTE standardization introduced several QoS class identifiers (QCIs) to support different requirements, such as guaranteed bit rate and non-guaranteed bit rate services [5]. QoS-unaware RRS allocates the RBs based on different buffer status, channel quality information (CQI), transmission queues, allocation history, etc. Otherwise, a QoS-aware also considers QoS priorities in resource allocation decision-making, provisioning the required services as throughput, delay, and latency [5]. The work [24] presents an example of a method that tries to balance the total system throughput maximization with the fulfillment of the QoS requirements using a called enhanced utilization resource allocation (EURA).

In 5G mobile networks, the problem of QoS requirements fulfillment becomes more complex since 5G users are diverse in demand for resources due to the diversity of applications such as ultra-high-definition videos, online games, and time-critical applications. It results in requirements such as high reliability, very low latency, and high data rate, increasing the need for efficient RRS approaches to satisfy network resource demands [25]. With more stringent requirements, 5G and B5G mobile networks provide a high volume of RAN data with information that could be obtained through network controllers [26] to be used in the RRS design.

It is challenging to propose methods that can extract all the benefits of the data available. This fact motivates the flourishing of machine learning for communications, gaining considerable momentum in computational vision and communication [27]. When contrasted to other artificial intelligence techniques, the main characteristic of machine learning is that the designed models are based on experience learned automatically from data [28], which is abundant in mobile communications, mainly in the RAN that has a large amount of information about the mobile network, UEs, and applications running, being extremely useful to learn complex patterns and improve network operations.

2.2 Network slicing

Network slicing provides service customization, isolation, and multi-tenancy support on shared physical network infrastructure through logical separation of the network resources [4]. It has a basis on infrastructure as a service (IaaS) from cloud computing models where an infrastructure provider can share their computer, network, and storage resources with more than one client using virtualization to create isolated virtual networks over a shared infrastructure. In the 5G and B5G context, a network slice is an implementation of virtual network functions (VNFs) interconnected via a virtual network that enables the creation of cost-efficient end-to-end network slices and dedicates them to dynamic provisioning of main features of 5G.

The network slicing concept has gained attention from different standardization bodies that define network slicing architectures, such as 3GPP [29], NGMN [30] and ITU-T [31]. Using network slicing, VNFs can adapt their operations depending on the service intents in the SLA specified for each slice [4]. The network defines service by creating different slices with their virtual structure and resources. Therefore, each slice can configure a virtual network with specific characteristics to provide requested services and guarantee the fulfillment of their requirements, e.g., a real-time communication application could be assigned to a slice that has VNFs near the application location to reduce communication latency and provide a radio resource reservation to the slice devices. The seven main principles of network slicing concept and operation are automation, isolation, customization, elasticity, programmability, end-to-end, and hierarchical abstraction [4].

Automation using signaling-based mechanisms allows clients to define their slice requirements, such as jitter, latency, and duration of a network slicing, enabling an on-demand configuration without needing fixed contractual agreements and manual intervention. Isolation ensures the performance and security of each slice without performance interference among them. Customization enables different treatments of slices with distinct requirements through resource allocations, network configuration, and policies. This customization can be improved using big data and context awareness features in the slices to perceive how to perform changes in the network, improving attended slices. Elasticity guarantees that slice's resources can be increased/decreased to attend the desired SLA over different conditions. Programmability enables clients (third parties) to control their slices' resources via APIs, which generate much flexibility in resource management and service customization [4].

End-to-end abstraction is a characteristic of network slicing due to the provisioning of a

complete application working from the service provider to the end user, combining resources of different infrastructure providers and heterogeneous technologies such as RAN, core network, transport, and cloud to implement a network slice. Finally, hierarchical abstraction enables the tenants of the slices to create partial or complete slices inside their contracted slice [4].

Fig. 2.3 shows a network slice diagram based on [4, 32]. Vertical segments, application providers, or mobile network operators own one or more slices based on their service intents. The network slices comprehend customized resources to fulfill personalized service performance requirements. One slice for mobile devices (slice A) and another for Internet of Things devices (slice B) are depicted in Fig. 2.3. Each network slice contains an end-to-end communication with virtualized resources allocated in the physical infrastructure. Both slices can share or isolate resources from the access, transport, and core network.

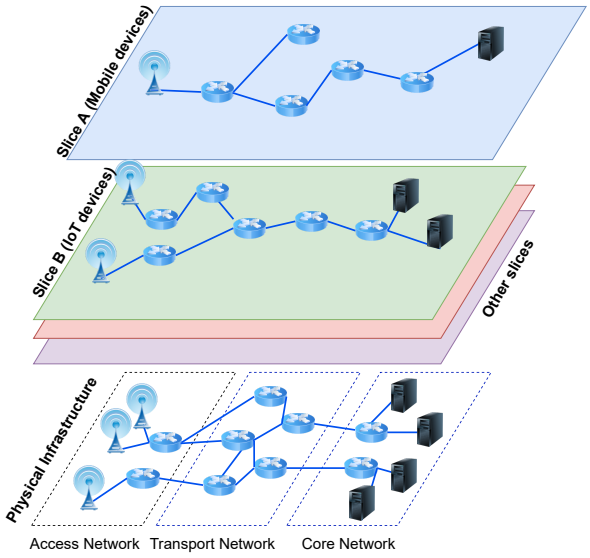

Figure 2.3: Network slicing example representing one slice for mobile devices and another to Internet of Things devices over a physical infrastructure.

It is important to emphasize that all slices consume limited physical resources, even if these resources are virtually isolated. Thus, network slicing management should avoid overprovisioning slice resources to fulfill its requirements since it may cause performance degradation of other slices in the network, mainly under high network resource utilization conditions.

Each network slice instance has a life-cycle management process that verifies if the service intents are fulfilled and improves the allocation of the network slicing. This life cycle includes four phases [4] illustrated in Fig. 2.4. The preparation phase is responsible for prepar-

ing and supporting the network slice. The instantiation, configuration, and activation phases allocate and initiate the resources needed for the slice instance. After the network slice instance is activated, it handles network traffic. The run-time phase involves supervising the network slicing process and monitoring network metrics to attend to intents defined in the SLA. If a fault is detected or there is an opportunity to improve the slicing process, the slice instance can be upgraded, reconfigured, or scaled. Each slice instance continuously reports its status to a controller. The decommissioning phase starts when a slice instance is finalized by request or execution time defined in the SLA, so this phase starts the deactivation and termination of the slice instance, reclaiming the allocated resources.

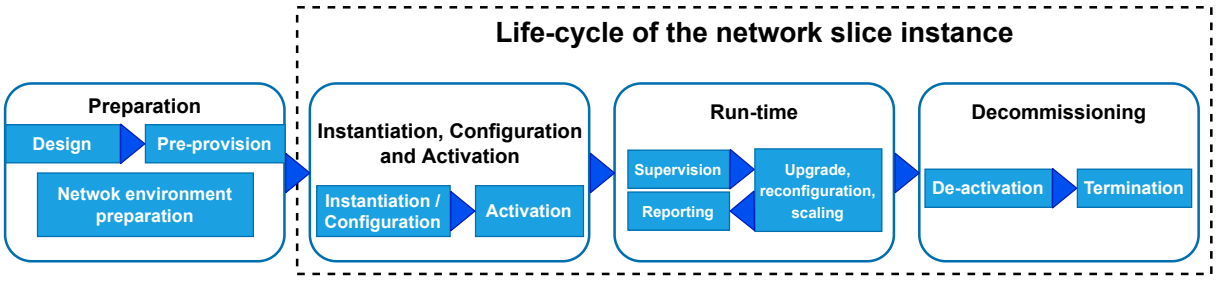

Figure 2.4: Network slicing instance life-cycle.

As explained before, a network slice has to deliver an end-to-end solution to a service, making continuous management and monitoring of resources to attend to slice requirements. This management and monitoring can be applied on RAN, transport network, and the core network domains of a mobile network. Therefore, it is essential to emphasize that the network slice needs to be applied to these three domains to enable end-to-end network slicing.

2.2.1 RAN slicing and radio resource scheduling

This work focuses on network slicing in the RAN domain, called RAN slicing, and discusses the RAN slicing optimizations in the RRS to meet slice intents. RAN slice implementation demands dynamic resource management, resource isolation, and sharing [4]. Dynamic resource management enables efficient resource sharing through sophisticated MAC functions, considering different intents for each slice. For example, a URLLC slice has a stringent latency requirement, whereas an eMBB slice requires high data rates. Therefore, the network should provide different treatments to these RAN slices since they present distinct network function behaviors and RRS needs.

The isolation and sharing of resources are essential aspects of RAN slicing due to the required spectrum isolation for some critical applications focused on latency and security. However, complete slice isolation reduces multiplexing gains in the processing and spectrum resource. Therefore, different levels of isolation of slice resources are supported in RAN slicing architectures, allowing isolated slices or complete sharing of RAN functions and radio resources. As an example, it is possible to deploy slices that contain isolated/dedicated RRC/PDCP/RLC/MAC/PHY[1] layers or RRC/PDCP/RLC layers isolated with shared MAC and PHY layers among slices [33].

This work aims to maximize the multiplexing gains of radio resources. At the same time, it maximizes the slice SLAs fulfillment. Thus, it considers a RAN slicing structure with isolated RRC/PDCP/RLC layers and shared MAC and PHY layers [33]. Using this configuration, radio resources are shared among slices, with an inter-slice scheduler distributing the radio resources between slices and an intra-slice scheduler distributing the assigned radio resources made by the inter-slice scheduler for each slice among the UEs. Therefore, both the inter- and intra-slice scheduling are responsible for allocating the available radio resources, aiming at attending to the slice's intents defined in the SLA requirements and providing radio resource guarantees for more important slices. At the same time, this scenario offers a higher radio resource multiplexing gain; it increases the complexity of RRS design since the resources of the slices are not isolated, and a change in the allocation of resources for a specific slice may impact others.

Fig. 2.5 illustrates a simplified inter- and intra-slice RRS process with three slices and five UEs connected to the network. The inter-slice scheduler allocates the RBs to the RAN slices following the obtained network information. It specifies the number of RBs for each slice to use. After the inter-slice scheduler defines the RB allocation for each slice, the intra-slice scheduler distributes the assigned RBs between slice's UEs following network information to fulfill the slice's intents defined in the SLA. There is an admission control module responsible for accepting new incoming slices in case the network can provide sufficient resources to fulfill the new slice's requirements without harming the performance of other slices in the network. An admission control module could also be used for the UEs of each slice, similarly to the one presented previously in the RRS without RAN slicing.

[1]Radio Resource Control, Packet Data Convergence Protocol, Radio Link Control, Medium Access Control, and Physical layer

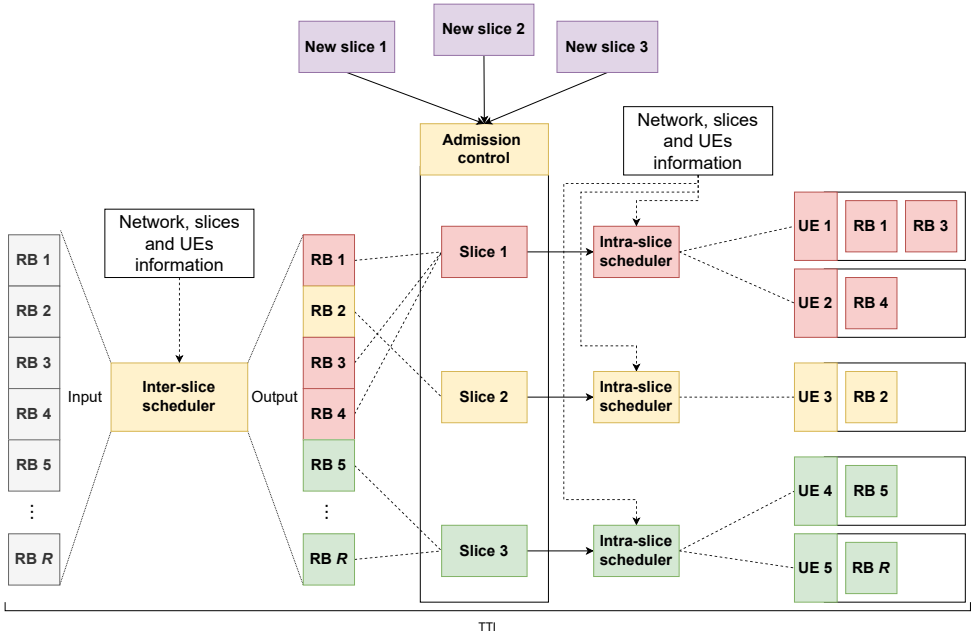

Figure 2.5: RRM process considering RAN slices.

2.3 Intent-based system

The mobile communications in B5G and 6G networks are leading to increased network automation in which human interventions are changed to more efficient management using artificial intelligence. The introduction of automation refers to a system that can automatically execute a process without human participation in every step and, ideally, without human involvement at all [9]. When considering network automation, the policies are crucial since they work as a rule or decision tree initiated when a defined precondition or event is triggered. It delivers an action or action plan to be executed. The input to a policy-based decision is usually the technical state of the system, including the specifications of what customers have ordered, all data and system information available such as inventories, analytic insights, and network key performance indicators (KPIs) [9].

Policy-based automation is similar to a pre-determined recipe in which developers write policies to generate all the decisions at design time by analyzing the system's situation and developing an action plan for each situation. In this case, policy-driven systems automate the execution at run-time, but the intelligent decisions are still primarily human-driven. Policy-based automation has limitations regarding the capability to adapt to changes and situations that were not explicitly defined at design time. New experienced situations can come from external environment factors such as changing user behaviors leading to new system states or

even new system functions typically introduced as new products or customization of existing ones. Therefore, keeping the system capable of dealing with new situations would imply a constant redesign of policies with partial human support [9].

Increasing the degrees of autonomy in a system would mean more automated tasks. The system would gradually take over intelligent decision-making and determine and adapt its operational solutions without human involvement. It is not limited by pre-designed recipes but makes new recipes itself when needed. This requires the automated system to have access to all relevant goals, requirements, and constraints. Furthermore, this knowledge must be presented to the system in a way that enables automated reasoning processes to translate them into adapted system behavior, where the knowledge about expectations needs to be formally expressed, communicated, and managed. In this context, [9] defines intents to specify and communicate knowledge about expectations to a system, allowing automated processes to reason about it and derive appropriate decisions and actions. Intent enables communication of what is preferable and what needs to be avoided. An intent-driven system cannot just blindly follow human-determined solution recipes but modify them and make their own.

The recent concept of intent was first introduced in this context by internet engineering task force (IETF) as goals that a network should meet and the outcomes that a network is supposed to deliver, defined in a declarative manner without specifying how to achieve or implement them [34]. Policies are considered part of the system because they determine the actions taken by the system. Specifically, it excludes any specification of how to achieve or implement an operation goal and mandating policies. It also excludes the requirement of hard-coded logic or artifacts, such as rules and workflows that define decision trees and decision-making processes [9]. The intent is purely the specification of requirements and goals separated from all implementation artifacts.

The formal definition of intents proposed by the Autonomous Networks Project in TM Forum [9] and zero-touch network and service management (ZSM) [35] is: Intent is the formal specification of all expectations, including requirements, goals, and constraints given to a technical system. This definition is inspired and compatible with the IETF definition [34]. Furthermore, this definition excludes all imperative implementation and solution aspects from the intent. In this context, the intent is purely an expression of what needs to be achieved or avoided or what outcome is more or less preferable, rather than indicating how and by which strategies and actions this can be realized [9]. Artifacts such as policies, rules, and decision trees

are still very needed to realize an intent-driven autonomous system, but they are separated from the intent expression. This thesis assumes the TM Forum [9] and ZSM [35] as the fundamental intent definition due to the large number of available resources and the leading role in intent standardization in the industry.

Intent is considered declarative because it specifies the desired outcomes and the outcomes that need to be avoided. Still, it can include quantitative specifications, such as a goal that can be set by defining target values or value ranges using KPI and metrics [9]. Depending on the scope of the intent, the definition can be high-level and abstract or technical and detailed. For example, a business-level intent can specify the need to make a financial gain from autonomously managed SLAs. An example of a lower-level technical intent would be to guarantee a $10\,\mathrm{ms}$ latency on a particular network link. Even with these different intent contents, the intent is only useful if all aspects it specifies can be observed since it can only control what it can measure. This means a system based on intent must be connected to data and knowledge sources. This includes metrics and KPIs being measured, aggregated, and calculated or analytic functions that provide information to the system [9].

Source: TM Forum [9].

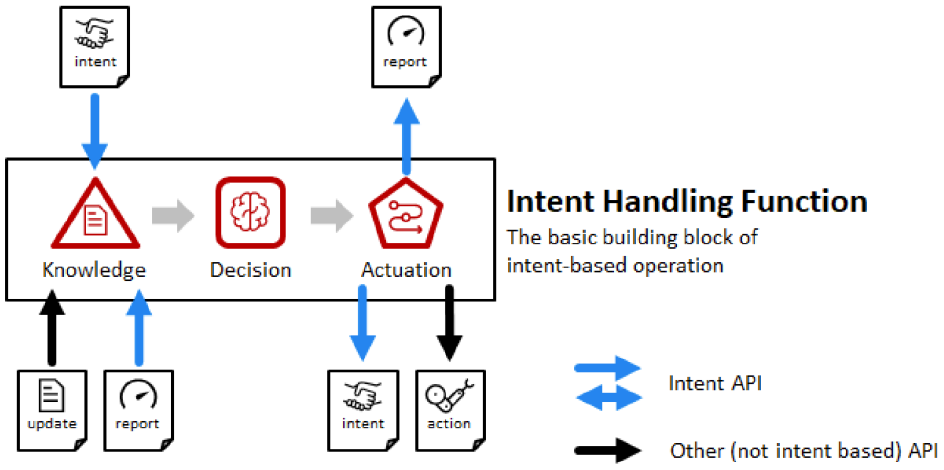

Figure 2.6: Intent handling function description.

The TM Forum defines the intent handling function (also called the intent manager function) as an entity that operates an autonomous system using intent, as illustrated in Fig. 2.6. The intent manager receives an intent generated by an external system (which could be another

intent manager). It operates based on knowledge, which refers to the operational goals and requirements specified in the intent and information about the system state or domain in which the intent handling function operates. The intent specifies the desired state, while the measured results determine the current state. The decision of the intent management function is mainly about reducing the distance between the current and wanted states.

The intent manager decides which actions are needed to fulfill the requested intent. It is important to note that an action plan can also include intent requests for other intent managers. In this case, the intent handling function becomes an intent owner since it generates an intent to be handled by another function. The intent function can act through conventional interfaces to control external modules to invoke processes or change system configurations. It also reports the intent fulfillment state to the intent handling function, which has requested the received intent and other important information related to the intent. The implementation of the intent handling function needs to implement all interfaces required to interact with the external functions of the domain in which it is operating. Therefore, the intent manager is highly contextual and depends on the domain.

Intent managers communicate with each other through intent reports to report on the status and success of intent handling. The intent manager function can be an intent owner or an intent handler. Intent owners generate intents for other intent managers and receive reports on the generated intent status. Intent handlers are responsible for receiving intents and acting in the system to fulfill the received intents. An intent manager can act as both an intent owner and a handler simultaneously.

Fig. 2.7 shows four distinct intent managers with scope specifications and the related models they support. The intent manager responsible for service management receives and handles intent from an intent manager assigned to business operations and is responsible for order management. Intents can contain requirements about service KPI and customer experience metrics. Intent managers need to understand the same intent models to communicate with each other. The intent manager in the service operations uses the subsequent intent to put requirements on RAN and core network operation. This figure emphasizes the relationship between different intent managers located in various domains. The intent manager in the servicer operation is both an intent handler since it acts to fulfill the intents received from the business operation, and an intent owner since it generates intents to the intent manager in the RAN and core network domains. The intent manager in autonomous domain RAN can interact with different external

Source: TM Forum [9].

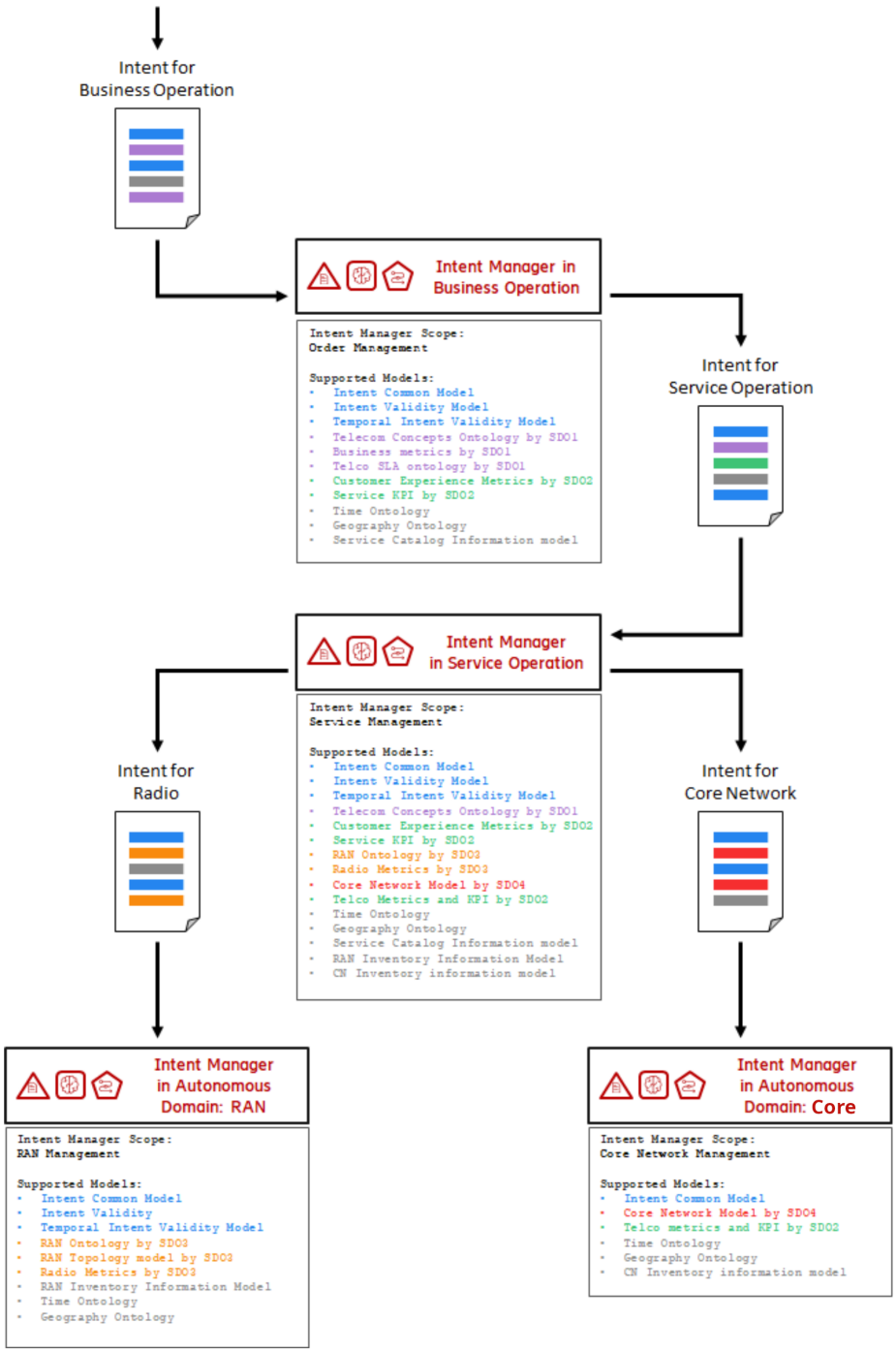

Figure 2.7: Example of different domain-specific intent manager interactions.

functions (outside the intent-based system) such as the RRS, modulation coding scheme (MCS) selector, and other RAN functions. It is essential to build intent-based RAN functions for the RRS in order to be able to ensure intents in the RAN and communicate with the intent manager in the RAN domain.

Source: TM Forum [10].

```
@prefix cat: <http://www.operator.com/Catalog/> .
@prefix met: <http://www.sdo2.org/TelecomMetrics/Version_1.0/> .

ex:ExampleIntentXYZ
  a icm:Intent ;
  icm:hasExpectation ex:Exp1_delivery ,
                     ex:Exp2_property .

ex:Exp1_delivery
  a icm:DeliveryExpectation ;
  icm:target _:service ;
  icm:params ex:Par1_description .
ex:Par1_description
  a DeliveryParam ;
  icm:targetDescription cat:ExampleService .

ex:Exp2_property
  a icm:PropertyExpectation ;
  icm:target _:service ;
  icm:params ex:Par2_latency ,
             ex:Par3_throughput ,
             ex:Par4_availability .
ex:Par2_latency
  a icm:PropertyParam ;
  met:latency [ icm:atMost "10 ms" ] .
ex:Par3_throughput  a icm:PropertyParam ;  met:throughput [ icm:atLeast [
met:value 5 ;
                                  met:unit met:unitMBPS  ]
.ex:Par4_availability  a icm:PropertyParam ;  met:availability [
icm:greater [ met:value 99.9 ;
                                    met:unit met:percentage ] .
```

Figure 2.8: Example of an intent requiring a certain latency, throughput, and availability for a service.

TM Forum's definition of intents stands for the use of formally defined models, meaning that intents would be expressed with formally defined and complete semantics and vocabulary, avoiding ambiguities in the meaning of an intent (the sender and receiver of intent must be in perfect agreement about its interpretation). TM Forum defines a common intent model [10] that could be used to enable communication between different intent managers. Fig. 2.8 shows an example of intent description for a service that specifies numerical requirements for latency, throughput, and availability of a service.

2.4 Reinforcement learning

Reinforcement learning is a machine learning technique to learn directly from interacting with an environment. RL is learning how to map situations to actions and how to maximize a numerical reward signal. The learner is not reached about which actions to take but must discover which actions yield the most reward by trying them. Actions taken by an RL agent may affect not only the immediate reward but also the following situations and, hence, all subsequent rewards. These two characteristics (trial-and-error search and delayed reward) are the two most important features of reinforcement learning [36]. Fig. 2.9 illustrates a usual Markov decision process (MDP) process that represents the interaction between the RL agent and the environment. The environment comprises everything outside the agent. The agent decides on an action and applies it in the environment. Then, the environment generates a new state due to the applied action and returns a reward signal to the agent. The reward signal defines the goal of the RL problem since the agent's objective is to maximize the total reward it receives over the long run [36].

Source: Sutton and Barto [36].

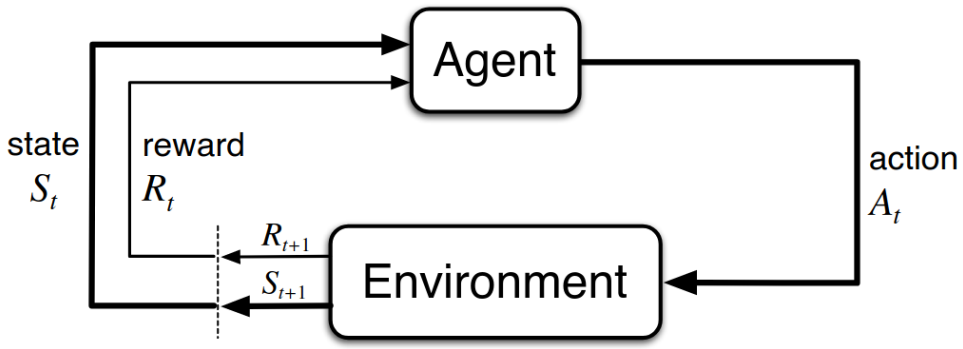

Figure 2.9: The agent-environment interaction in a Markov decision process.

The RL scenario can be described with an MDP $(S, A, \mathsf{RW}, P, \rho_0)$, where S is the set of all valid states, A is the set of all valid actions, RW is the reward function, P is the transition probability function, and ρ_0 is the initial state distribution of the system [36]. In a time step t, the agent in a state s_t takes action a_t and reaches the next state s_{t+1} receiving the reward $\mathsf{RW}(s_t, a_t)$. The RL agent has policy $\pi(\cdot|s_t)$ representing a function that maps states to actions. Rewards are numerical values given to the agent's actions to represent if the chosen action was

good, and the agent aims to maximize the long-term cumulative reward [36]. This thesis uses a subset of RL methods that utilize deep neural networks as approximators, usually called deep reinforcement learning (DRL). This thesis will utilize these terms interchangeably.

This thesis utilizes actor-critic algorithms, which are a type of RL algorithm that combines aspects of both policy-based methods in the actor and value-based methods in the critic [36]. The actor's role is to make decisions (select actions) based on the current policy and explore the action space to maximize the expected cumulative reward. When continually refining the policy, the actor learns the dynamic nature of the environment. The critic's role is to evaluate the actors' actions and estimate their value, providing feedback on their performance. The critic is vital to driving the actor's learning for actions that lead to higher expected returns. Usually, in actor-critic RL methods, both the actor and the critic are implemented using deep neural networks. Fig 2.10 illustrates the actor-critic method with both the actor and critic utilizing deep neural networks as approximators. The actor has a policy $\pi(a|s_t)$ representing the probability of taking action a in the state s where this policy is implemented by a deep neural network with parameters denoted as θ. The actor maximizes the expected return by optimizing its policy. The critic network estimates the expected cumulative reward $V(s_t)$ starting from a state s and optimizes its neural network parameters W to improve its ability to predict the correct value. This thesis focuses on PPO and SAC RL algorithms, which are actor-critic-based methods, and explains these methods specifically in the following subsections.

2.4.1 Proximal Policy Optimization (PPO)

The PPO is an on-policy actor-critic algorithm that alternates between sampling data through interactions with the environment and optimizing a surrogate objective function using stochastic gradient ascent [37]. The PPO method has the benefits of data efficiency and reliable performance of trust region policy optimization (TRPO) but with a much simpler implementation, more general, and has a better sample complexity [37]. It proposes a novel objective with clipped probability ratios and a pessimistic estimation of the policy's performance. This thesis utilizes the PPO variation with the clipped probability ratios due to its best performance [37]. The PPO improves the update of the policy parameters in the actor network while using a critic implementation similar to other standard actor-critic methods.

First, using their current network parameters, the PPO agent uses the actor and critic networks to generate a batch of experiences (actions, state, and rewards). Then, the actor and

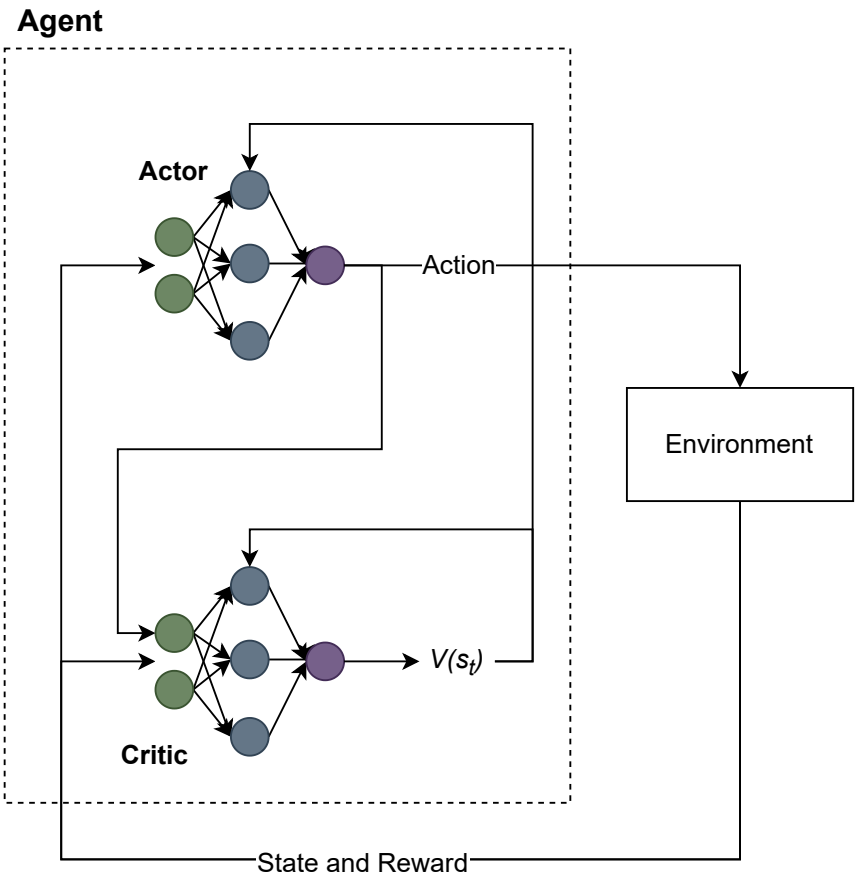

Figure 2.10: Actor-critic general architecture with actor-network generating actions to the environment while the critic network estimates the value function.

critic networks are updated using the obtained batches of experience. The critic (value function) network with its parameters W minimizes the loss function

$$L_t^{\mathsf{VF}}(W) = \mathbb{E}\left[(V_W(s_t) - R_t)^2\right], \tag{2.1}$$

where $V_W(s_t)$ represents the critic network output to the value function that represents the expected return value when starting in the state s_t. At the same time, R_t is the actual return obtained in the batch of experiences, which is the discounted sum of rewards over time from time step t until the T samples in the batch

$$R_t = \mathsf{RW}(s_t) + \gamma \mathsf{RW}(s_{t+1}) + \gamma^2 \mathsf{RW}(s_{t+2}) + \cdots + \gamma^T \mathsf{RW}(s_T), \tag{2.2}$$

where γ is the discount factor, which balances the weight between immediate and future rewards.

The actor network concentrates the main contributions of the PPO method. The actor-network maximizes the clipped objective function

$$L_t^{\mathsf{clip}}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big], \tag{2.3}$$

where the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\mathsf{old}}(a_t|s_t)} \tag{2.4}$$

represents the current policy π_θ changes in relation to the old policy $\pi_{\theta\mathsf{old}}$. And, A_t is the generalized advantage estimation which represents how much better a particular action was compared to the average action at a given state s_t [38]

$$A_t = \sum_{l=0}^{T-1} (\gamma\lambda)^l \delta_{t+l}, \tag{2.5}$$

where λ represents the generalized advantage estimation parameter with values between 0 and 1, with 0 representing a one-step return being highly biased by the immediate returns and with a low variance, while 1 represents the usage of long-term returns with a low bias but a high variance. Finally, the temporal difference error δ_t measures the difference between the predicted value of a state and the actual value. The temporal difference error is

$$\delta_t = \mathsf{RW}(s_t) + \gamma V(s_{t+1}) - V(s_t), \tag{2.6}$$

where a positive δ_t indicates that the critic predicted that the value function was too pessimistic and that the value function should be updated for a higher value. Otherwise, the value function was too optimistic and should be updated for a smaller value.

The first term $r_t(\theta)\hat{A}_t$ within the minimum function of Equation 2.3 represents the conservative policy iteration where the maximization of only this term would lead to an excessively large policy update. Equation 2.3 takes the minimum between the clipped and unclipped probability ratio so that the final objective is a lower bound on the unclipped objective. Clipping the objective between the defined interval improves stability and reliability when updating the policy values [37].

Finally, the PPO total loss is

$$L_{\mathsf{total}}(\theta) = \mathbb{E}_t\left[-L_t^{\mathsf{clip}}(\theta) + c_1 L_t^{\mathsf{VF}}(\theta)\right], \tag{2.7}$$

which includes an coefficient c_1 to balance the importance between the actor and critic loss functions [37]. The total loss represents the overall objective the training process seeks to minimize.

2.4.2 Soft Actor-Critic (SAC)

The SAC RL method is an actor-critic approach that combines elements of value-based and policy-based methods exploring the off-policy strategy and encouraging exploration through entropy maximization [39]. SAC is an off-policy algorithm since it learns from experience stored in a replay buffer instead of learning from the latest actions, which leads to better data efficiency when compared to on-policy algorithms such as PPO [39]. Unlike the previously explained actor-critic architecture, SAC uses three different networks, one being the actor and two critics. The actor, similarly to the actor-critic approach, maps states to actions. The critics estimate the action-value function (Q-values) instead of the value function since it expresses a relation between each action option in a given state s_t and expected return instead of calculating the expected return independent of the action taken as done in the value function calculation [36]. Moreover, it utilizes two critic networks to reduce the bias that might occur if only one critic were used.

SAC's core idea is adding an entropy term to the reward function to maintain exploration over training. The agent is incentivized to choose effective and diverse actions to prevent premature convergence to suboptimal policies [39]. The SAC objective is represented as

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}[\mathsf{RW}(s_t, a_t) + \varrho \mathcal{H}(\cdot|s_t)], \tag{2.8}$$

where $\mathsf{RW}(s_t, a_t)$ represents the reward for state s_t when choosing action a_t, $\mathcal{H}(\cdot|s_t)$ is the entropy term, and the temperature parameter ϱ controls the trade-off between exploration and exploitation [39].

The critic networks are updated by minimizing the loss function

$$J_Q(\phi_i) = \mathbb{E}\left[\left(Q_{\phi_i}(s_t, a_t) - y\right)^2\right] \tag{2.9}$$

where

$$y = \mathsf{RW}(s_t, a_t) + \gamma \min(Q_{\phi_1}(s_{t+1}, a_{t+1}), Q_{\phi_2}(s_{t+1}, a_{t+1})) - \varrho \log \pi(s_{t+1}|a_{t+1}). \tag{2.10}$$

$Q_{\phi_i}(s, a)$ is the Q-value predicted by the i-th critic network with parameters ϕ_i for the current state-action pair (s, a). It considers the minimum between the Q-values predicted by the two critics to avoid overly optimistic predictions. The entropy regularization term $\varrho \log \pi(s_{t+1}|a_{t+1})$ encourages the method to explore when the entropy value is high and leads to deterministic actions otherwise.

The actor parameters are updated to maximize the expected return with an entropy bonus by minimizing

$$J_\pi(\theta) = \mathbb{E}\left[\varrho \log \pi(s_t|a_t) - \min(Q_{\phi_1}(s_t, a_t), Q_{\phi_2}(s_t, a_t))\right], \tag{2.11}$$

where a higher entropy term increases the actor loss, encouraging the policy to be more exploratory and select actions stochastically.

An important characteristic of SAC is its off-policy nature in which the agent utilizes a replay buffer to store past experiences and then samples mini-batches of experiences from this buffer to update its network, enabling a more efficient reuse of data compared to on-policy algorithms such as PPO.

When comparing PPO and SAC methods, SAC has a more complex implementation, usually requiring more fine-tuning in the hyperparameters while providing a better sampling efficiency due to its off-policy nature using a replay buffer. The PPO method has a simpler implementation, providing a more stable and higher processing throughput while requiring less fine-tuning and being easier to use for general applications. Both PPO and SAC algorithms have been used in different related works for RRS [12, 19, 40, 41] due to their excellent performance in control problems.

Chapter 3

Intent-aware RRS using RL for fixed network scenarios

This chapter presents the proposed intent-aware RRS for RAN slicing using RL with slice prioritization for fixed network scenarios, using three different slice types and QoS intents based on throughput, latency, and packet loss rate. This approach considers an RL agent for inter-slice allocation, where the RRS provides the distribution of radio resources among slices and utilizes a round-robin scheduler to perform the intra-slice allocation. It does not consider the slice admission problem [5]. The RL agent learns to take RRS actions to fulfill the slices' intents and give higher priorities to slice intents following pre-defined weights. An intent drift reward calculation is proposed to deal with different QoS intents, enabling different slice specifications. To test the effectiveness of the proposed data-driven RRS algorithm, a simulator with a number of realistic features was developed to avoid oversimplifying the data. For example, the proposed simulator uses the QuaDRiGa channel simulation framework to generate spatially consistent millimeter wave multiple-input and multiple-output (MIMO) channels. It also defines different traffic models and varies the traffic magnitude for each UE depending on its slice type, assessing the RL agent capacity to generalize its decision for different network conditions. The proposed software is publicly available to facilitate future work and comparisons. The results to be presented in this chapter were published in [40].

The main contributions of this chapter are summarized as follows:

- An intent-aware RRS using RL with an intent drift reward calculation to prevent violations on the QoS intents and prioritize the most critical slices' intents when the radio resources available are not sufficient to fulfill all intents.

- A closed loop to deal with intent drift caused by changes in the network scenario (without human intervention) and intent updates provided by network operators.
- A method that supports different QoS requirements for each slice type, with the flexibility to change its requirements in real-time, promoting the needed network adaptations to fulfill the slices' intents still.
- Evaluation of the proposed method using realistic millimeter wave MIMO channels generated using QuaDRiGa and variations in the UE's traffic magnitude according to each slice characteristic.
- The simulation code is publicly available to facilitate reproducibility and comparison with other methods.[1]

This chapter is organized as follows: Section 3.1 describes the related works, emphasizing the data-driven methods using RL for RRS systems with RAN slicing. It also relates the main contributions of this proposed method to previous work. Section 3.2 presents the adopted communication system model and formulates the RRS problem in a scenario with RAN slicing. Section 3.3 presents the proposed intent-aware RRS using a SAC RL agent to perform inter-slice allocation and a round-robin scheduler for intra-slice scheduling. Section 3.4 presents the results obtained using the proposed method and a comparison with baselines, focusing on slice intent fulfillment.

3.1 Related work

Considering the need to design RRS solutions with support to RAN slicing, previous work has proposed optimization techniques and machine learning methods using model-based paradigms such as the Lyapunov optimization method [42], earliest deadline first (EDF) scheduling adaptation [42], MDPs [43], and supervised machine learning methods [41]. The success of the model-based paradigm depends on its accuracy in representing the behavior of mobile networks. With the evolution of the cellular network, the underlying mathematical models have become even more complex to exploit sophisticated technologies [11]. On the other hand, the data-driven network's approach provides network functions directly built on the data produced, presenting advantages such as better network information usage and the reduced need

[1] https://github.com/lasseufpa/rrm-slice-rl

for accurate models [11]. The data-driven RRS approach usually utilizes RL methods due to its unsupervised learning characteristic and high efficiency in learning from large amounts of data [36].

The data-driven paradigm using RL for RRS in scenarios with RAN slicing was tackled in [12–16], which is the main scope of this chapter. The prior work [12–14] focuses on the maximization/minimization of specific network metrics, where [12] proposed an RL PPO method that is O-RAN compliant and considers both inter-slice and intra-slice allocation. It considers eMBB, machine-type communication (MTC), and URLLC slice types. The RL agent is responsible for distributing resource block groups (RBGs) among slices (inter-slice allocation) and chooses an intra-slice scheduling between RR, PF, and water filling techniques. This work [12] uses two utility/reward functions: the first focuses on maximizing the rate for eMBB and transport blocks for MTC and minimizing the buffer size for URLLC. The second focuses on maximizing the rate for eMBB and MTC and the RB ratio for URLLC. The RL agent uses a reward function to maximize system metrics without providing guarantees to meet specific slice requirements previously described in an SLA. Therefore, this method could not be considered in an intent-based system where the network goals are provided through specific QoS values/objectives. For example, the system cannot serve a slice request to keep the throughput rate above a specified value in the intent definition of TM Forum [10], even considering QoS metrics in the reward calculation. Still, since the QoS goals are not defined, the agent is unable to deal with intents.

The previous work [13] proposes a combined method utilizing long short-term memory (LSTM) deep learning technique and asynchronous advantage actor-critic (A3C) RL method. It considers that each slice contains an RL agent that defines the number of resources required to meet its requirements. With this network environment, actions between slices are unobservable, such that each slice makes decisions independently. An indicator vector designates if there are sufficient resources to be shared among slices and guides the slices' strategies towards decreasing the required resources to make feasible resource allocation decisions. Using the approach of decentralized decisions considered in [13], each slice has an RL agent that gives more flexibility to learn different behaviors for each slice. However, the indicator vector for insufficient resources would be a problem in a scenario where the available resources are insufficient to provide all slice requirements, generating a great negative reward for all slices that would not converge to an ideal solution.

The authors in [14] propose a two-level radio resource and power allocation scheduler using RL for RAN slicing scenarios. The upper-level controller uses a deep deterministic policy gradient (DDPG) RL method to set a guaranteed bit rate for all slices to ensure the QoS requirement of services and a maximum bit rate value to avoid too many radio resources occupied by a unique slice. The lower-level controller, using a double deep Q-network (DQN), utilizes the upper-level controller's outputs as a constraint to allocate the RBs and transmit power for each active UE in each slice. Each slice has a utility function related to the QoS performance, represented by the aim to minimize the average packet latency and packet drop rate and maximize the SE. There are weights for each of the requirements of the utility function to create requirement priorities that differentiate each slice, but without establishing priorities among slices. Therefore, similarly to [12], [13, 14] cannot be implemented or adapted to work in an intent-based system because their systems did not consider clear QoS values to fulfill the slice intents.

Otherwise, the prior work [15,16] focuses on the SLA satisfaction rate for each slice, with work [15] proposing an LSTM network to improve the advantage actor critic (A2C) RL method to set the bandwidth for each slice considering the SLA satisfaction. This work does not specify the SLA satisfaction ratio functions for each slice type nor discuss slices or network metrics prioritization as required for an intent-based system as defined by TM Forum [9]. In [16], a DRL method using Ape-X for distributed learning is implemented to provide RB allocation for slices. It considers a binary representation of the slice fulfillment, where 1 represents that the slice requirements are fulfilled and 0 otherwise, which is insufficient to represent the intent drift described by the IETF definition for intent-based systems [34]. It does not enable the RL agent to realize a distance to fulfill the slice intents or even to perceive when the provided actions are increasing the distance between the monitored metrics and network objectives. Moreover, there is no discussion about slice and network metric prioritization to cope with more sensitive slices in scenarios where the experienced channel capacity is insufficient to fulfill all served slice intents.

The main problems related to the slice's intent fulfillment in [12–16] are summarized below:

- None of the related works [12–14] could be utilized in an intent-based system since in the TM Forum specification, the intent should provide wanted outcomes and outcomes to avoid, including quantitative specifications such as defining target values or value ranges using KPIs. All these related works consider that the goal of each slice is to maximize or

minimize specific network metrics. Still, it is insufficient for an intent description since it is impossible to define when the intent is being fulfilled. For example, in work [12], the eMBB slice goal is defined as maximizing the rate for UEs. Still, considering an intent-based system, it is impossible to determine when the system is not maximizing the throughput rate (fulfilling the intent).

- The designed reward/utility functions that drive the learning process do not enable the RL agent to realize the intent drifts. Instead, they aim to maximize/minimize specific slice metrics [12–14] that may cause over-provision of radio resources for slices in relation to the requested intents or consider the intent-fulfillment as a binary variable in the reward calculation [16].

- Despite the method [14] providing weights to define the importance of each of the three requirements in the slice reward contribution, the prioritization among slices is not tackled. For instance, slices with greater priorities may compete for the same radio resources with less important ones in scenarios with scarce radio resources due to high network demands.

- The different slices in the network have distinct intents that are not always described by the same metrics as considered in [12–14, 16]. Therefore, the network should support slice types with different metrics to enable intent-based slices. For example, the prior work [44, 45] considers the existence of minimum bit rate, constant bit rate, and BE slice types, each of them considering different metrics for their requirements.

- In an intent-based sliced network, a change in the slice intents (promoted by the network operator) or an intent drift should cause changes in the RRS operations to fulfill the intents. In the prior work [12–16], there is no discussion of the effects of changes in requirements in different network conditions to evaluate the capacity of the RL agent in various scenarios.

Taking into account the above-highlighted problems, this chapter presents an intent-aware RRS for RAN slicing using RL, with support for different types of slice and metrics focusing on fulfilling the intent requirements. Unlike [12–14], it designs a reward function that incentivizes the RRS agent to fulfill QoS intents described in a common intent model [10] and adapt to the changes in intents during the simulation. Unlike [14], which defines the importance weights for

each slice metric, it defines weights that prioritize metrics into specific slices and in relation to other slice metrics. It also considers different metrics per slice, considering eMBB, URLLC, and BE slice types.

3.2 Communication system model and problem formulation

The model assumes a massive millimeter wave MIMO system with $c = 1, 2, \ldots, C$ cells with a target cell c_{target} representing the cell being evaluated. The target cell c_{target} has N_{t} transmit antennas for a specific carrier frequency f_{c} and contains $\upsilon = 1, 2, \ldots, \Upsilon$ slices, where Υ represents the total number of active RAN slices in the cell. The target cell has $u = 1, 2, \ldots, U$ UEs, where each UE u is assigned to a specific slice υ and contains N_{u} antennas. It implements a simulation of a MIMO system because of its high transmission capacity and vast advantages in 5G and B5G networks. Still, other RF transmission schemes (such as SISO) could be adopted without incurring any change in the proposed system.

The target cell has a bandwidth of B MHz, divided into RBs, and RBs are grouped into R RBGs, which are the minimum radio resource allocation unit considered in the proposed scheduling system. The minimum time unit considered in the scheduling process is a TTI t, representing the time to process and allocate all RBGs in a specific step n. Therefore, each step takes t ms with $t_n = t \cdot n$ representing the time from step 1 to n. The inter-slice RRS allocates each RBG for a specific slice υ, and in each slice, there is an intra-slice RRS responsible for distributing the inter-slice RRS assigned RBGs to the slice's UEs. The model considers the uplink RRS problem where the base station defines the RBGs each UE uses to send its information, but the method could also be applied for downlink.

Fig. 3.1 represents a scheduling process in a RAN slice scenario with 3 slices and 2 UEs, composed of inter- and intra-slice scheduling. In action 1, the inter-slice scheduling allocates the R available RBGs among the slices, generating an association between RBGs and slices depicted in action 2. It considers a contiguous allocation, so when inter-slice RRS allocates 3 RBGs to a specific slice, it should allocate 3 sequentially located RBGs. After the RBGs' distribution into slices, intra-slice scheduling executes for each slice, assigning their RBGs (action 3) to their UEs (action 4). This chapter focuses on inter-slice RRS and uses an RR method for intra-slice scheduling.

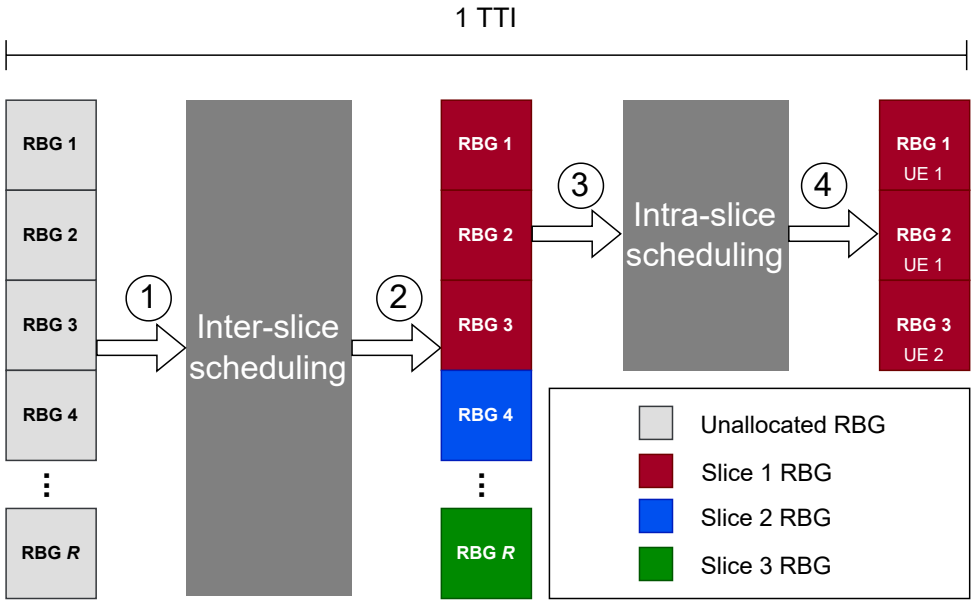

Figure 3.1: RRS scenario adopted in this work contains the inter- and intra-slice scheduling process. The inter-slice scheduling distributes the available RBGs between the slices, and the intra-slice scheduling distributes the assigned RBGs between the UEs

3.2.1 Massive MIMO system model

The modeled system uses a time division duplex (TDD) transmission protocol that receives pilots from UEs sent through the uplink to obtain the base station's CSI. Each base station obtains a perfect channel estimation for each UE. UEs have their spectral efficiency values varying with time, but the same UE's spectral efficiency value for all RBGs is used, similar to [12, 46, 47].

The modeled system uses a scenario with a hexagonal tessellation of cells with $C = 7$ cells, as illustrated in Fig. 3.2. The analysis occurs in the target cell and considers interference from the six neighboring cells. Each cell has a base station that serves three independent sector antennas formed by a controlled vertical uniform linear array (ULA), having a total of $3C = 21$ serving cells. There is no cooperation among the base stations or sectors, with all interferences (inter-cell/inter-sector) being treated as noise, allowing interference from antenna back lobes as well as from other base stations. The hexagonal network layout is chosen for simplicity and reproducibility. The simulation Subsection 3.2.2 uses a rich and expressive stochastic channel generator that produces realistic channel variability even with a hexagonal base station layout. The proposed algorithm works in the target cell and analyzes the performance of the target cell as is typical in prior work, such as [48].

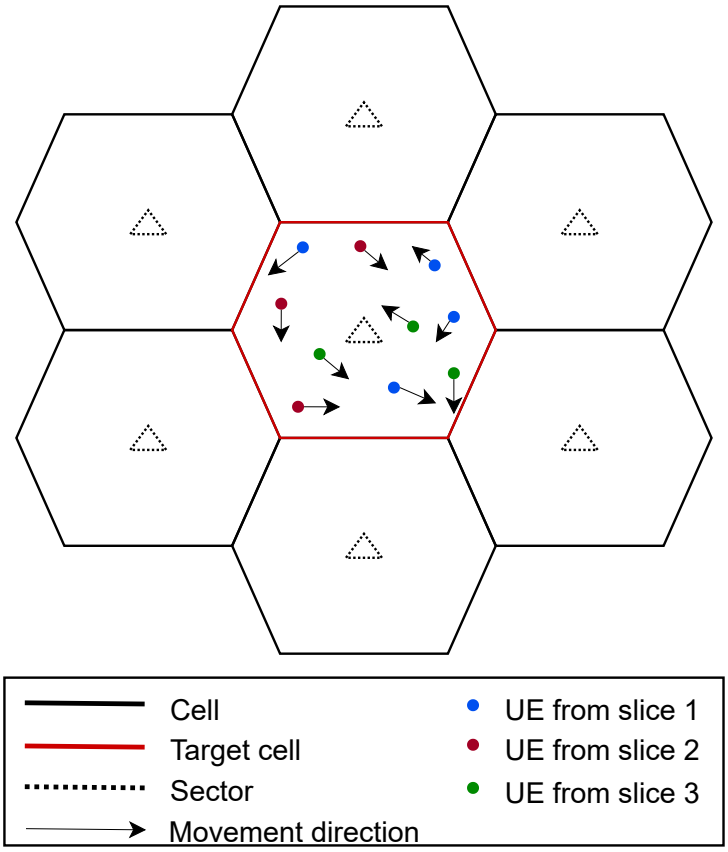

Figure 3.2: There is a mobile network scenario with 7 base stations (each with 3 antenna sectors), and 10 UEs distributed in the target cell. The UEs are randomly placed within $150\,\mathrm{m}$ in relation to the target base station.

It is assumed that each sector has a set of independently controlled ULAs in horizontal orientation at the base station. Using this configuration, the elevation beamforming is controlled by the sector's tilt control, while the combination of ULAs allows for azimuth beamforming. This array structure balances the flexibility and full-dimensional beamforming gain with the simplicity and efficiency of more traditional arrays. Operators have traditionally preferred sectorization and tilt control as a means of beamforming [49, 50].

The neighboring cells' interference in the target cell is primarily defined by simulating the interference cells' channels and assuming they beamform in the same azimuth direction as the UE. This represents the worst-case setting where the interfering base station is beamforming toward a user in the same direction as the target UE, thereby assuming the UE receives the upper bound on the interference. It is, therefore, a worst-case assumption on the amount of imposed interference by a base station without changing the physical arrays. RBs and time

slots in different cells are assumed to be time-aligned but without pilot contamination, since it is largely inconsequential for the array structure due to limited pilots needed without elevation beamforming.

3.2.2 Channel modeling and spectral efficiency

It uses the QuaDRiGa [51] software to consistently generate UE channels in space and frequency. This widely used statistical channel simulator generates spatially and correlated MIMO channels from statistical models, including experimentally validated channel models using the 3GPP 38.901 UMi [52, 53] statistical models based on dual-slope path loss with significant inter-parameter correlations. The process of generating channel coefficients is meticulously presented in [54]. However, it is essentially the combination of generating random values, corresponding to the per-ray paths with specified distributions, correlating the values, and applying path loss and shadowing effects described in the simulation code[2]. According to the 3GPP UMi scenario [52, 53], it generates channel coefficients and correlates them across UEs in space and time.

Cellular systems use the reference signal received power (RSRP) information for UE assignment and resource allocation, calculating for a set of A clusters and Z rays with a path loss PL and shadow fading SF for a base station b and UE u pair [54, Section 8.1]. So, the RSRP is defined as

$$\mathsf{RSRP}_{b,u} = p_{u,b}\mathsf{PL}\ \mathsf{SF}\left(\left|\alpha_0\right|^2 + \sum_{a=1}^{A}\sum_{z=1}^{Z}\left|\alpha_{a,z}\right|^2\right), \tag{3.1}$$

where $p_{u,b}$ is the transmission power for UE u at base station b, α_0 is the line-of-sight (LOS) path contribution, and $\alpha_{a,z}$ is non-line-of-sight (NLOS) paths for each ray z and cluster a [54, Section 8.1].

Finally, the RSRP is calculated for each sector-UE pair with the UE reporting only the strongest cell (the nominal serving cell) and the top-6 strongest interfering cells during measurement reports to the base station [55]. Because of the sectorization, the top-6 interfering cells cover the majority of the interference since all other cells are not aligned in the UE's direction. Despite the existence of back lobes on the sector antennas, the interference is limited due to the large front-to-back ratio and the tilt mechanism, which causes back lobes to be projected upwards.

[2] https://github.com/Ryandry1st/QuaDRiGA-Simulation-Extensions

The SE for massive MIMO systems have different well-established capacity-like formulas that allow us to estimate the SE in $\mathrm{bit/s/Hz}$ [56]. The capacity bound SE is defined as

$$\mathsf{SE}_{b,u}^{\mathrm{UP}}(n) = \log_2\left(1 + \frac{\mathsf{RSRP}_{b,u}}{I_{b,u}^{\mathsf{inter}} + \sigma^2}\right), \tag{3.2}$$

where σ^2 is the noise power and $I_{b,u}^{\mathsf{inter}}$ is the inter-cell interference defined by

$$I_{b,u}^{\mathsf{inter}} = \sum_{i \neq b} \max{}^{(6)}(\mathsf{RSRP}_{i,u}), \tag{3.3}$$

where $\max^{(6)}$ is the set of the 6 largest elements. These equations assume all base stations are beamforming in the direction of each UE, including out-of-cell users, with uniform power control. The vast majority of the interference is accounted for by the top 6 interfering cells due to the significant NLOS path loss seen in millimeter wave channels.

3.2.3 RAN slicing and radio resource scheduling

In a scenario with RAN slicing, the RRS should distribute the RBGs among the slices in such a manner that the described slice intents are fulfilled, reserving radio resources to more important slices (with higher priority) and sharing resources with less important ones when possible. Each slice υ contains a set of U_υ UEs with similar traffic behavior and the same QoS intents. A vector

$$\boldsymbol{R}_n(n) = [R_1(n), R_2(n), \ldots, R_\Upsilon(n)] \tag{3.4}$$

defines the number of RBGs allocated for each slice in the step n, where $R_\upsilon(n)$ represents the number of RBGs allocated to slice υ at step n. The RRS process obeys to

$$\sum_{\upsilon=1}^{\Upsilon} R_\upsilon(n) = R, \tag{3.5}$$

where the sum of all RBGs distributed along with the slices is always equal to the total amount of RBGs available R. Therefore, the main function of the RRS in a RAN slicing scenario is to define $R_\upsilon(n)$ for each slice υ in a step n according to the network conditions to fulfill the slice intents.

Since there is a specified number of RBGs R available to distribute among the slices, the number of possible combinations of RBGs is limited and represented by

$$\boldsymbol{R}_{\mathsf{comb}}(n) = [\boldsymbol{R}_{\mathsf{comb}}^1(n), \boldsymbol{R}_{\mathsf{comb}}^2(n), \ldots, \boldsymbol{R}_{\mathsf{comb}}^{\left|\boldsymbol{R}_{\mathsf{comb}}(n)\right|}(n)], \tag{3.6}$$

where $\boldsymbol{R}_{\mathsf{comb}}^{\text{option}}(n)$ represents one of the possible combinations of RBGs and $\left|\boldsymbol{R}_{\mathsf{comb}}(n)\right| = \frac{(R+\Upsilon-1)!}{R!(\Upsilon-1)!}$ represents the number of possible RBGs combinations for $\boldsymbol{R}_n(n)$ in each step n.

It is important to define network metrics to evaluate the RRS performance and propose slice intents. Therefore, the served throughput $r_u(n)$ is the maximum throughput given in megabits per step (Mbps) that a UE u can obtain, considering the number of RBGs allocated and the spectral efficiency available

$$r_u(n) = \left\lfloor \frac{(R_\upsilon^u(n)/R)B\mathsf{SE}_u(n)}{10^6\ \mathsf{PS}} \right\rfloor \mathsf{PS}, \tag{3.7}$$

where $R_\upsilon^u(n)$ represents the number of RBGs from slice υ allocated to the user u by intra-slice scheduling at step n, $\mathsf{SE}_u(n)$ the spectral efficiency to the user u at step n, and PS the packet size that is the minimum network's transfer unit in bits.

The effective throughput $e_u(n)$ represents the data throughput sent over the network containing data that were available in the buffer. It is defined as

$$e_u(n) = \min(r_u(n), b_u(n)), \tag{3.8}$$

where $b_u(n)$ represents the data available in the buffer of UE u at step n. Hence the effective throughput is always $e_u(n) \leq r_u(n)$ with $e_u(n) = r_u(n)$ when $b_u(n) \geq r_u(n)$.

The buffer occupancy rate $b_u^{\mathsf{occ}}(n)$ is defined as

$$b_u^{\mathsf{occ}}(n) = \frac{b_u(n)}{b^{\mathsf{max}}}, \tag{3.9}$$

where b^{max} is the maximum UE buffer capacity. Packets are discarded every time the buffer is full or the packet latency exceeds the maximum latency allowed l^{max}, so these packets are included in the dropped data $d_u(n)$ that represents the summation of the size of the dropped packets in step n. The packet loss rate $p_u(n)$ is calculated over a window interval of w steps, being defined as

$$p_u(n) = \begin{cases} \frac{\sum_{i=(n-w)}^{n} d_u(i)}{b_u(n-w)+\sum_{i=(n-w)}^{n} \iota_u(i)}, & \text{if } n \geq w \\ \frac{\sum_{i=1}^{n} d_u(i)}{b_u(1)+\sum_{i=1}^{n} \iota_u(i)}, & \text{if } n < w \end{cases}, \tag{3.10}$$

where $\iota_u(n)$ is the requested throughput by UE u at step n. The requested throughput depends on the slice υ that the UE is associated with since the traffic behavior of UEs of the same slice is similar. The requested throughput will be later defined for each slice.

The average buffer latency $\ell_u(n)$ represents the average time that a packet waits in the buffer of the UE u, and it is defined as

$$\ell_u(n) = \frac{\sum_{i=0}^{l^{\mathsf{max}}} i\boldsymbol{l}_n^u(i)}{\sum_{i=0}^{l^{\mathsf{max}}} \boldsymbol{l}_n^u(i)}, \tag{3.11}$$

with $\boldsymbol{l}_n^u = [l_0, l_1, \ldots, l_{l^{\max}}]$ where $\boldsymbol{l}_n^u$ is a vector with size $l^{\max} + 1$ representing the packets' latency (the total number of TTIs the packets waited) on buffer for user u at step n, and $l_{l^{\max}}$ represents the number of packets that have waited for $l^{\max}$ TTIs in the buffer.

The long-term served throughput $g_u(n)$ represents the average served throughput obtained over w steps, and it is defined as

$$g_u(n) = \begin{cases} \frac{\sum_{i=(n-w)}^{n} r_u(i)}{w}, & \text{if } n \geq w \\ \frac{\sum_{i=1}^{n} r_u(i)}{n}, & \text{if } n < w \end{cases}, \tag{3.12}$$

and the fifth-percentile served throughput is also calculated over a window with w steps:

$$f_u(n) = \begin{cases} P_{5\%}(r_u(n-m), \ldots, r_u(n)), & \text{if } n \geq w \\ P_{5\%}(r_u(1), \ldots, r_u(n)), & \text{if } n < w \end{cases}, \tag{3.13}$$

where $P_{5\%}(\cdot)$ represents the fifth-percentile calculation for all values in the function argument.

3.2.4 Slice types and intents

The network slice metrics are defined as an average of the UEs' metrics associated with a specific slice υ. So, a specific metric SM_υ for slice υ is represented as

$$\mathsf{SM}_\upsilon = \frac{\sum_{u=1}^{U_\upsilon} \mathsf{SM}_u}{U_\upsilon}, \tag{3.14}$$

where SM_u can represent the spectral efficiency, served throughput, effective throughput, buffer occupancy, packet loss rate, requested throughput, average buffer latency, long-term served throughput, or the fifth-percentile served throughput of a specific UE u. And U_υ represents the total number of UEs associated with the slice υ. Each slice type has different intents, so different metrics are used as requirements. This work assumes three different slice types: eMBB, URLLC, and BE. The mMTC applications are out of the scope of this work due to the high connection density (about one million devices per km^2) that would be unfeasible to the channel generation process used in this work. Therefore, the RRS should be optimized to fulfill the slice's intents as described by each slice type below.

3.2.4.1 eMBB slice

The UEs assigned to the eMBB slice require high throughput, regardless of the channel conditions, and they do not have stringent latency and packet loss requirements. Ultra-high-quality video streaming is an example of an eMBB application. It defines three main QoS

intents to the eMBB slice: the average served throughput $r_{\mathsf{embb}}(n)$ should be equal to or above a specified minimum served throughput $r^{\mathsf{req}}_{\mathsf{embb}}$. The average latency $\ell_{\mathsf{embb}}(n)$ and the packet loss rate $p_{\mathsf{embb}}(n)$ should be kept below the required average latency $\ell^{\mathsf{req}}_{\mathsf{embb}}$ and packet loss rate $p^{\mathsf{req}}_{\mathsf{embb}}$. The requested throughput from UEs u associated with eMBB slice $\iota^{\mathsf{embb}}_u(n)$ is a Poisson distribution with mean μ_{embb} [45].

3.2.4.2 URLLC slice

The UEs assigned to a URLLC slice require low latency and ultra-reliable communication, usually characterized by a low packet loss rate. The throughput requested in a URLLC is usually less than that required by eMBB. Some examples of URLLC applications are to monitor an automation process and its remote control, which does not require great throughput but instead requires high communication reliability and low latency to work well. It defines the same network intents from eMBB to URLLC, with $r_{\mathsf{urllc}}(n)$, $\ell_{\mathsf{urllc}}(n)$, and $p_{\mathsf{urllc}}(n)$ representing the URLLC served throughput, average latency, and packet loss rate, respectively. With $r^{\mathsf{req}}_{\mathsf{urllc}}$, $\ell^{\mathsf{req}}_{\mathsf{urllc}}$, and $p^{\mathsf{req}}_{\mathsf{urllc}}$ representing the URLLC intents for served throughput, average latency, and packet loss rate, respectively. The requested throughput $\iota^{\mathsf{urllc}}_u(n)$ from URLLC UEs u is defined as a Poisson distribution with mean μ_{urllc}.

3.2.4.3 BE slice

The BE slice represents a slice with less priority than eMBB and URLLC [5], with no stringent requirements for throughput and latency. Social network multimedia with intermittent traffic are examples of BE applications. BE considers two main intents: the long-term served throughput $g_{\mathsf{be}}(n)$ should be equal to or above the specified minimum throughput $g^{\mathsf{req}}_{\mathsf{be}}$. Furthermore, the fifth-percentile served throughput $f_{\mathsf{be}}(n)$ should be equal to or above a specified minimum throughput $f^{\mathsf{req}}_{\mathsf{be}}$. The BE UEs are activated or deactivated in each n_{be} steps with half probability for each of them. The requested throughput for UE u from BE slice $\iota^{\mathsf{be}}_u(n)$ is defined as a Poisson distribution with mean μ_{be} if the UE is activated or zero otherwise.

The proposed method differs from related works [12–14] by defining intents with specific QoS requirements/goals as described in the TM Forum intent common model for parameters [10]. Besides, it also differs from [15,16] by supporting other slice types with different QoS intents associated with them (using a different combination of network metrics, for example) due to the usage of the intent drift reward calculation (explained after in the Subsection 3.3.4)

that defines the reward design to accomplish QoS intent goals to the RL agent.

3.3 Proposed intent-aware RRS using reinforcement learning

This work proposes an intent-aware RRS for a RAN slicing scenario using RL based on the TM Forum IG1253 [9] and IETF request for comments (RFC) 9315 [34]. IETF and TM Forum have compatible intent definitions. The former defines intent as a set of operational goals and outcomes provided in a declarative manner without specifying how to achieve or implement them. TM Forum defines intent as the formal specification of all expectations, including requirements, goals, and constraints given to a technical system. Moreover, TM Forum also stands for the use of formally defined models, meaning that intents would be expressed with formally defined and complete semantics and vocabulary, avoiding ambiguities in the meaning of an intent (the sender and receiver of intent must be in perfect agreement about its interpretation). Therefore, even using declarative statements provided by human operators, these statements should be translated for a common intent model as specified by the TM Forum in [10]. Both IETF and TM Forum do not specify the translation mechanisms from declarative statements to QoS intents or a common intent model. This proposal considers that intentions follow a common intent model defined by the TM Forum [10], and property parameters are extracted, generating the QoS intents described in Subsection 3.2.4.

Fig. 3.3 shows the proposed intent-aware system life cycle based on the IETF RFC [34] applied to the RRS problem. The two main horizontal planes represent the difference between functions related to fulfillment and assurance of intents. Intent fulfillment provides functions and interfaces that allow network operators to communicate network intents and perform the necessary action to ensure that intent is achieved. These include algorithms to determine proper courses of action and optimize outcomes over time [34]. The intent assurance provides functions and interfaces to validate and monitor if the network complies with intent, assessing the effectiveness of actions taken as part of fulfillment.

Functions are also divided into three spaces: user, translation, and network operations spaces. The user space involves functions that interface the network and the intent system with the human user (network operator). The translation or intent-based system (IBS) space converts the intent into a course of actions to be applied in the network operations infrastructure and the

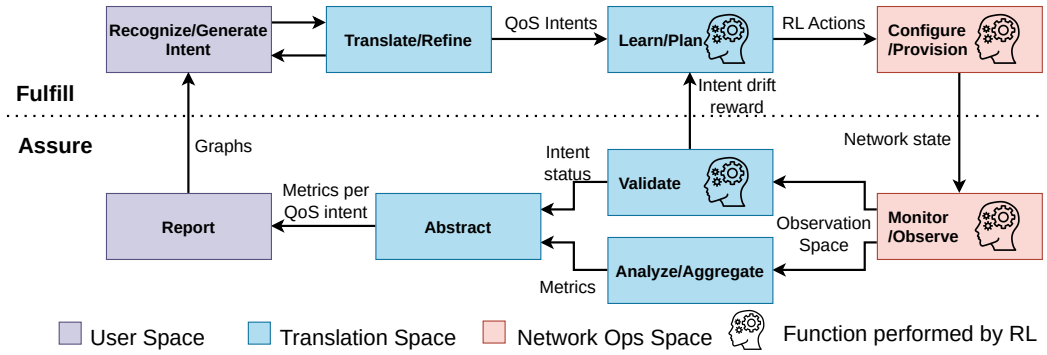

Figure 3.3: Proposed intent-aware RRS life cycle based on IETF intent-based system definition using an RL agent. The system is composed of two closed loops: the inner intent control loop is responsible for fulfilling the QoS intents for RAN slices obtained from the translate/refine function (without human intervention). The outer intent control loop is responsible for updating the QoS intents based on new interactions provided by the network operator.

translation from network metrics to the operator. Finally, the network operations space involves the orchestration, configuration, and monitoring functions to effectuate actions in the network and observe their effects. The whole process starts with the recognize/generate intent function, which is responsible for obtaining intent from network operators. An example of an intent expressed in a natural language could be “a 4K video streaming service for a group of 10 users.” The translate/refine function transforms the received declarative intent into a common intent model [10] wherefrom slice type and QoS intents could be extracted, as specified in Subsection 3.2.4. This work does not implement the recognize/generate intent and the translate/refine functions. Instead, it is assumed that intents are provided in a common intent model, wherefrom QoS intents are extracted.

An RL agent performs the learn/plan, configure/provision, monitor/observe, and validate functions in the proposed intent-aware RRS system. The RRS agent receives the QoS intents and the network status in the learn/plan function, generating an inter- and intra-slice scheduling action with the RBGs distribution among slices and UEs. In the configure/provision function, the network applies the scheduler decision, updating the network status and metrics from slices and UEs in the system following Section 3.2. In the assurance functions, the monitor/observe function monitors the network, calculating the metrics described in Subsection 3.2.3. It generates an observation space used as a basis for the next set of functions to assess whether the observed behavior complies with the expected behavior based on the intent.

The validate function evaluates the effectiveness of intent fulfillment actions. It calculates the intent drift over time. Intent drift occurs when a system originally meets the intent but, over time, gradually allows its behavior to change or be affected until it no longer does or does so in a less effective manner [34]. Therefore, based on QoS requirements, an intent drift reward calculates how distant the system is from fulfilling its goals. Thus, every time an intent drift occurs due to a network change (such as different traffic behaviors, channel variations, or even ineffective actions made by the agent), the RL agent can automatically learn from its designed intent drift reward to perform actions that will improve the intent fulfillment status. The learn/plan, configure/provision, monitor/observe, and validate functions compose the inner intent control loop that does not involve any human in the loop and is responsible for ensuring that the RRS actions will guarantee the defined intents.

The outer intent control loop includes the analyze/aggregate function that aggregates the network metrics, organizes the information by time, and enables analysis such as the QoS intent fulfillment along time. The abstract function filters the most important metrics for each slice and its QoS intents, e.g., the BE slice prefers long-term served throughput and the fifth-percentile served throughput. Finally, the report function organizes the metrics per QoS intent and generates informative graphs for the network operator.

3.3.1 Reinforcement learning agent

This work proposes an RL agent to perform inter-slice RRS operations jointly with a round-robin scheduler performing intra-slice RRS operations illustrated in Fig. 3.1. It implements the learn/plan, configure/provision, monitor/observe and validate functions from Fig. 3.3. The RL agent obtains an observation space with information about the network state (including QoS intents), and interacts with the network environment using actions that define the number of RBGs per slice and UEs. In an RL environment, the scenario can be described with an MDP $(S, A, \mathsf{RW}, P, \rho_0)$, where S is the set of all valid states, A is the set of all valid actions, RW is the reward function, P is the transition probability function and ρ_0 is the initial state distribution of the system [36]. In a time step t, the agent in a state s_t takes action a_t and reaches the next state s_{t+1} receiving the reward $\mathsf{RW}(s_t, a_t)$. Rewards are numerical values given to the agent's actions to represent if the chosen action was good, and the agent aims to maximize the long-term cumulative reward [36]. In this case, the reward represents the fulfillment of slices' intents described in the common intent model.

The agent follows a policy $\pi(\cdot|s_t)$ defined as a distribution of actions given the current state s_t. When dealing with a great number of states and actions, the use of classical tabular methods becomes prohibitively inefficient, with function approximators and model-free RL being implemented to deal with these problems [36]. Two major problems that emerge from these techniques are the high sample complexity and sensitivity to hyperparameters, with off-policy learning being used to improve sample efficiency. Off-policy methods can experience stability and convergence issues when using continuous states and action spaces.

This work adopts the SAC RL method defined in [39], which optimizes a stochastic policy using an off-policy technique for continuous actions, forming a bridge between stochastic policy optimization and DDPG approaches [57]. The SAC method improves exploration and solves the stability issues presented by off-policy methods. Standard RL methods maximize the expected sum of rewards. However, the SAC technique considers a more general maximum entropy objective which favors stochastic policies by augmenting the objective with the expected entropy:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}[\mathsf{RW}(s_t, a_t) + \varrho \mathcal{H}(\cdot|s_t)], \tag{3.15}$$

where T is the horizon and ϱ indicates the relative importance of the entropy term to the reward. The addition of the entropy term encourages exploration by assigning equal probabilities to actions that have near the same values, avoiding repeatedly selecting a particular action (causing high increases in their probabilities), allowing an improved exploration phase and better stability for the SAC method in relation to other RL methods such as DDPG and A3C [39].

In addition to the aforementioned reasons for using the SAC RL method, the RRS problem was defined with a continuous action space to deal with the different numbers of RBG available. Therefore, the RL agent output range is the same, even changing the number of RBGs available in the system. The SAC method is known for its stability and convergence when dealing with continuous action spaces [39].

3.3.2 Observation space

Due to the limited information of the system at the base station, the observation space $\boldsymbol{O}_n$ in a given step n is defined as a representation of the state s_t containing knowledgeable information to be used in (3.15). The observation space is defined as

$$\boldsymbol{O}_n = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\Upsilon, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\Upsilon, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_U], \tag{3.16}$$

being a vector composed of RAN slice intents vectors $\boldsymbol{r}_\upsilon$ defined in the SLA for each slice υ describing the QoS requirements that each slice needs to fulfill, the slice metrics vector $\boldsymbol{s}_\upsilon$, and the UEs metrics vector $\boldsymbol{u}_u$. The slice intents vector $\boldsymbol{r}_\upsilon$ value depends on the slice type, with $\boldsymbol{r}_{\mathsf{embb}} = [r^{\mathsf{req}}_{\mathsf{embb}}, \ell^{\mathsf{req}}_{\mathsf{embb}}, p^{\mathsf{req}}_{\mathsf{embb}}]$, $\boldsymbol{r}_{\mathsf{urllc}} = [r^{\mathsf{req}}_{\mathsf{urllc}}, \ell^{\mathsf{req}}_{\mathsf{urllc}}, p^{\mathsf{req}}_{\mathsf{urllc}}]$ and $\boldsymbol{r}_{\mathsf{be}} = [g^{\mathsf{req}}_{\mathsf{be}}, f^{\mathsf{req}}_{\mathsf{be}}]$. Slice metrics vectors $\boldsymbol{s}_\upsilon$ and UEs metrics vectors $\boldsymbol{u}_u$ are composed of the nine metrics defined for each slice and UE: spectral efficiency, served throughput, effective throughput, buffer occupancy, packet loss rate, requested throughput, average buffer latency, long-term served throughput, and the fifth-percentile served throughput. As an example, the observation space size $|\boldsymbol{O}_n|$ for a scenario with three slices, one of each slice type, and ten UEs is $|\boldsymbol{O}_n| = |\boldsymbol{r}_{\mathsf{embb}}| + |\boldsymbol{r}_{\mathsf{urllc}}| + |\boldsymbol{r}_{\mathsf{be}}| + 3|\boldsymbol{s}_\upsilon| + 10|\boldsymbol{u}_u| = 3 + 3 + 2 + 3 \cdot 9 + 10 \cdot 9 = 125$.

An observation space with a high dimension size increases the time and complexity of training the RL model, affecting the agents' performance. Besides the observation space $\boldsymbol{O}_n$, a limited observation space $\boldsymbol{O}^{\mathsf{lim}}_n = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\Upsilon, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\Upsilon]$ is defined, without UEs metrics since slices metrics are calculated as an average of their UEs metrics. Considering a limited observation space for the same example used before, the observation space size is $\left|\boldsymbol{O}^{\mathsf{lim}}_n\right| = 35$. Using a limited observation space, the state does not scale with the number of UEs connected to the network at the cost of decreasing the network description used by the RL agent.

Both observation space and reward calculation (presented later in Subsection 3.3.4) use batch normalization to reduce the dependence of gradients on the parameters' scale or of their initial values, facilitating the choice of hyperparameters, such as the learning rate, without risk of divergence [58]. Therefore, handling inputs with different range values is more effortless and speeds up the learning process.

3.3.3 Action space

The proposed method defines an action as a vector $\boldsymbol{A}_n$ in a given step n, which is defined as $\boldsymbol{A}_n = [a_1, a_2, \ldots, a_\Upsilon]$, where a_υ represents an action factor for slice υ with value in a range $[-1, 1]$ to match the output of the Gaussian distribution for continuous actions used, improving the learning process [39]. After that, the agent's chosen action $\boldsymbol{A}_n$ is mapped to one of the $\boldsymbol{R}_n(n)$ options using the following

$$\mathsf{index}(n) = \underset{\text{option}}{\arg\min} \left(d \left(\boldsymbol{R}^{\text{option}}_{\mathsf{comb}}(n), \left(R \frac{\boldsymbol{A}_n + 1}{\sum_{i=1}^{\Upsilon}(a_i + 1)} \right) \right) \right) \tag{3.17}$$

where d calculates the Euclidean distance between the options available at $\boldsymbol{R}_{\mathsf{comb}}(n)$ and RL agent output $\boldsymbol{A}_n$. Finally, the scheduling decision applied in the RL environment (network system) is $\boldsymbol{R}^{\mathsf{index}}_{\mathsf{comb}}(n)$.

3.3.4 Intent drift reward calculation and slice prioritization

The reward function $\mathsf{RW}(n)$ considers the slice QoS intents as a basis to define how close the slice metrics are to fulfilling their intents, avoiding intent drifts in the validate function from Fig. 3.3. So, the reward has one component for each slice type as defined in

$$\mathsf{RW}(n) = \sum_{i \in \Upsilon_{\mathsf{embb}}} \mathsf{RW}_{\mathsf{embb},i}(n) + \sum_{i \in \Upsilon_{\mathsf{urllc}}} \mathsf{RW}_{\mathsf{urllc},i}(n) + \sum_{i \in \Upsilon_{\mathsf{be}}} \mathsf{RW}_{\mathsf{be},i}(n), \tag{3.18}$$

where the $\mathsf{RW}_{\mathsf{embb},i}(n)$, $\mathsf{RW}_{\mathsf{urllc},i}(n)$ and $\mathsf{RW}_{\mathsf{be},i}(n)$ represent the reward for eMBB, URLLC, and BE slices with index i at step n. The objective of the SAC RL agent is to maximize the reward function values, maximizing the fulfillment of slice intents. Therefore, the slices' rewards are defined with zero values every time the network fulfills the intents. And negative values when the network perceives intent drifts that, in this case, are characterized as a non-zero distance between the obtained network metric and its QoS intent.

3.3.4.1 eMBB reward contribution

The served throughput $\mathsf{RW}^{r}_{\mathsf{embb}}(n)$, average buffer latency $\mathsf{RW}^{\ell}_{\mathsf{embb}}(n)$, and the packet loss rate $\mathsf{RW}^{p}_{\mathsf{embb}}(n)$ rewards compose the eMBB slice reward contribution

$$\mathsf{RW}_{\mathsf{embb}}(n) = -(\mathsf{RW}^{r}_{\mathsf{embb}}(n) + \mathsf{RW}^{\ell}_{\mathsf{embb}}(n) + \mathsf{RW}^{p}_{\mathsf{embb}}(n)), \tag{3.19}$$

where the served throughput reward contribution is defined as

$$\mathsf{RW}^{r}_{\mathsf{embb}}(n) = \begin{cases} w^{r}_{\mathsf{embb}} \frac{r^{\mathsf{req}}_{\mathsf{embb}} - r_{\mathsf{embb}}(n)}{r^{\mathsf{req}}_{\mathsf{embb}}}, & \text{if } r_{\mathsf{embb}}(n) < r^{\mathsf{req}}_{\mathsf{embb}} \\ 0, & \text{if } r_{\mathsf{embb}}(n) \geq r^{\mathsf{req}}_{\mathsf{embb}} \end{cases}, \tag{3.20}$$

with w^{r}_{embb} being a weight that defines the intent importance in relation to the other ones. The average buffer latency reward contribution is

$$\mathsf{RW}^{\ell}_{\mathsf{embb}}(n) = \begin{cases} w^{\ell}_{\mathsf{embb}} \frac{\ell_{\mathsf{embb}}(n) - \ell^{\mathsf{req}}_{\mathsf{embb}}}{l^{\mathsf{max}} - \ell^{\mathsf{req}}_{\mathsf{embb}}}, & \text{if } \ell_{\mathsf{embb}}(n) > \ell^{\mathsf{req}}_{\mathsf{embb}} \\ 0, & \text{if } \ell_{\mathsf{embb}}(n) \leq \ell^{\mathsf{req}}_{\mathsf{embb}} \end{cases}, \tag{3.21}$$

with w^{ℓ}_{embb} working as a weight for average buffer latency intent. The packet loss rate reward contribution is

$$\mathsf{RW}^{p}_{\mathsf{embb}}(n) = \begin{cases} w^{p}_{\mathsf{embb}} \frac{p_{\mathsf{embb}}(n) - p^{\mathsf{req}}_{\mathsf{embb}}}{1 - p^{\mathsf{req}}_{\mathsf{embb}}}, & \text{if } p_{\mathsf{embb}}(n) > p^{\mathsf{req}}_{\mathsf{embb}} \\ 0, & \text{if } p_{\mathsf{embb}}(n) \leq p^{\mathsf{req}}_{\mathsf{embb}} \end{cases}, \tag{3.22}$$

with w^{p}_{embb} being a weight for packet loss rate intent.

3.3.4.2 URLLC reward contribution

The served throughput $\mathsf{RW}^{r}_{\mathsf{urllc}}(n)$, average buffer latency $\mathsf{RW}^{\ell}_{\mathsf{urllc}}(n)$, and packet loss rate $\mathsf{RW}^{p}_{\mathsf{urllc}}(n)$ rewards compose the URLLC slice reward contribution

$$\mathsf{RW}_{\mathsf{urllc}}(n) = -(\mathsf{RW}^{r}_{\mathsf{urllc}}(n) + \mathsf{RW}^{\ell}_{\mathsf{urllc}}(n) + \mathsf{RW}^{p}_{\mathsf{urllc}}(n)), \tag{3.23}$$

where these contributions are defined in the same manner as eMBB reward contributions, having its own weights w^{r}_{urllc}, $w^{\ell}_{\mathsf{urllc}}$ and w^{p}_{urllc}.

3.3.4.3 BE reward contribution

The long-term throughput $\mathsf{RW}^{g}_{\mathsf{be}}(n)$ and fifth-percentile throughput $\mathsf{RW}^{f}_{\mathsf{be}}(n)$ rewards compose the BE slice reward contribution

$$\mathsf{RW}_{\mathsf{be}}(n) = -(\mathsf{RW}^{g}_{\mathsf{be}}(n) + \mathsf{RW}^{f}_{\mathsf{be}}(n)), \tag{3.24}$$

where the long-term throughput is defined as

$$\mathsf{RW}^{g}_{\mathsf{be}}(n) = \begin{cases} w^{g}_{\mathsf{be}} \frac{g^{\mathsf{req}}_{\mathsf{be}} - g_{\mathsf{be}}(n)}{g^{\mathsf{req}}_{\mathsf{be}}}, & \text{if } g_{\mathsf{be}}(n) < g^{\mathsf{req}}_{\mathsf{be}} \\ 0, & \text{if } g_{\mathsf{be}}(n) \geq g^{\mathsf{req}}_{\mathsf{be}} \end{cases}, \tag{3.25}$$

with w^{g}_{be} being the long-term throughput intent weight. The fifth-percentile throughput is

$$\mathsf{RW}^{f}_{\mathsf{be}}(n) = \begin{cases} w^{f}_{\mathsf{be}} \frac{f^{\mathsf{req}}_{\mathsf{be}} - f_{\mathsf{be}}(n)}{f^{\mathsf{req}}_{\mathsf{be}}}, & \text{if } f_{\mathsf{be}}(n) < f^{\mathsf{req}}_{\mathsf{be}} \\ 0, & \text{if } f_{\mathsf{be}}(n) \geq f^{\mathsf{req}}_{\mathsf{be}} \end{cases}, \tag{3.26}$$

with w^{f}_{be} being the fifth-percentile throughput intent weight.

Due to the channel and requested traffic variation, the proposed system may not meet the requirements defined in the QoS intents, even with the RL agent performing the best possible

actions. Therefore, slice prioritization is implemented using weights. Each weight for a specific QoS intent in a slice type defines the importance of its requirement for the other intents (including all slices). The eMBB and URLLC receive higher weight values because they have higher priority over the BE slice; hence, when the network cannot fulfill all intents, it should prioritize the eMBB and URLLC intents. It is a good practice to keep the summation of all weights as the value of one since it would keep a limited reward with the maximum reward value $\mathsf{RW}(n)$ possible as 0 and the minimum as -1, facilitating the interpretation of a QoS fulfillment.

3.3.5 Baselines

Two RRS baselines were adapted for RAN slicing using RL from [12, 14]. It is important to emphasize that the related works contain different simulated/emulated scenarios assumptions, and they do not provide their RRS implementation codes. Therefore, the reward calculation from these baselines [12, 14] was implemented, focusing on the inter-slice RRS for comparison with the proposed solution. Moreover, an adaptation of the PF algorithm [22] for RAN slicing that considers each slice as a UE is implemented.

3.3.5.1 PF scheduling

It keeps a balance between trying to maximize the total network throughput and providing all slices with a minimal level of service. The PF action can be defined as $\boldsymbol{A}_n^{\mathsf{pf}} = [a_1, a_2, \ldots, a_\Upsilon]$, where action a_υ is

$$a_\upsilon = \frac{\min(\mathsf{SE}_\upsilon(n)B, b_\upsilon^{\mathsf{occ}}(n)b^{\mathsf{max}})}{\overline{e_\upsilon(n)}} \tag{3.27}$$

and $\overline{e_\upsilon(n)}$ represents the average effective throughput obtained by UEs in the slice υ. Finally, the action factors are mapped to one of the $\boldsymbol{R}_n(n)$ options using (3.17), generating an action $\boldsymbol{R}_n^{\mathsf{index}}(n)$.

3.3.5.2 Sched-slicing scheduling

Adapted from [12], utilizing a PPO RL agent to perform inter-slice scheduling and RR algorithm for intra-slice scheduling. It utilizes the same action and observation as the proposed agent but uses the reward function from [12]

$$\mathsf{RW}^{\mathsf{ss}}(n) = r_{\mathsf{embb}}(n) + f_{\mathsf{be}}(n) - b_{\mathsf{urllc}}^{\mathsf{occ}}(n)b^{\mathsf{max}}\mathsf{PS}, \tag{3.28}$$

that maximizes the served throughput $r_{\mathsf{embb}}(n)$ for eMBB and the fifth-percentile served throughput $f_{\mathsf{be}}(n)$ for BE, and minimizes the buffer occupancy $b^{\mathsf{occ}}_{\mathsf{urllc}}(n)$ for URLLC.

3.3.5.3 Lower-level scheduling

Adapted from [14] lower-level control policy, utilizing a DDPG RL agent to perform inter-slice scheduling and RR for intra-slice scheduling. The DDPG agent uses the same action and observation space as the proposed agent, but the reward function is the same from [14], which is

$$\mathsf{RW}^{\mathsf{lls}}(n) = \sum_{\upsilon \in \Upsilon} w^{\ell}_{\upsilon} \exp(-\ell_{\upsilon}(n)) + w^{p}_{\upsilon} \exp(-p_{\upsilon}(n)) + w^{r}_{\upsilon} r_{\upsilon}(n), \tag{3.29}$$

where weight values w were defined in [14] as $w^{\ell}_{\mathsf{embb}} = 1$, $w^{p}_{\mathsf{embb}} = 0.5$, $w^{r}_{\mathsf{embb}} = 2 \times 10^{-4}$, $w^{\ell}_{\mathsf{urllc}} = 2$, $w^{p}_{\mathsf{urllc}} = 1$, $w^{r}_{\mathsf{urllc}} = 4 \times 10^{-4}$, $w^{\ell}_{\mathsf{be}} = 0.2$, $w^{p}_{\mathsf{be}} = 0.1$ and $w^{r}_{\mathsf{be}} = 0.25 \times 10^{-4}$.

3.4 Simulation results

The simulations use the Python programming language, with the Stable Baselines3 library [59] implementing the RL algorithm and normalization and the Optuna library [60] implementing hyperparameter optimization. The communication network parameters used in the simulations are presented in Table 3.1.

3.4.1 Hyperparameter optimization, training, and testing

There are $ep_{\mathsf{max}} = 50$ episodes available that contain $n_{\mathsf{ep}} = 2000$ steps each, where each step is equivalent to $1\,\mathrm{ms}$, with $ep_{\mathsf{train}} = 45$ episodes used for training and $ep_{\mathsf{test}} = 5$ episodes used for testing the RL agent's performance. The number of epochs $ec = 10$ defines how many times the RL agent should train over the entire set of training episodes. Therefore, the total number of steps in the training process is defined as $n_{\mathrm{train}} = ep_{\mathsf{train}} n_{\mathsf{ep}} ec = 45 \cdot 2000 \cdot 10 = 900000$ steps. The hyperparameter optimization process uses the same episodes from training but with a different number of epochs, totalizing 500 thousand steps per hyperparameter combinations trial (when the trial is not pruned), totalizing 100 trials. Table 3.2 shows the optimized hyperparameters obtained using the full and limited observation space after the optimization process.

During the training, in every 10000 steps, the RL model is evaluated over 5 episodes, collecting the average normalized reward obtained over these episodes. This evaluation process

Table 3.1: Simulation parameters from network communication scenario.

Parameter:	Value:
# eMBB slices	1
# URLLC slices	1
# BE slices	1
# eMBB UEs (U_{embb})	4
# URLLC UEs (U_{urllc})	3
# BE UEs (U_{be})	3
RBGs available (R)	17
Window interval (w)	10
Bandwidth (B)	$100\,\mathrm{MHz}$
Carrier frequency (f_{c})	$28\,\mathrm{GHz}$
# transmission antennas (N_{t})	64
# receive antennas (N_{u})	1
Maximum buffer size (b^{max})	$8.38\,\mathrm{Mbytes}$ (1024 packets)
Packet size (PS)	$8192\,\mathrm{bytes}$
BE Steps (n_{be})	200

Table 3.2: Optimized hyperparameters obtained from optimization process for full and limited observation space.

Hyperparameter	Full obs. Space	Limited obs. Space
Discount factor	0.9	0.9
Learning rate	0.00281	0.00228
Batch size	128	128
Buffer size	10^5	10^5
Learning starts	0	0
Training Frequency	1	32
Polyak coefficient	0.05	0.08
Network architecture	Medium (256x256)	Small (64x64)

helps assess the RL model performance during the training and choose the best model, which is the model with the biggest average normalized reward obtained during the evaluation. The test process uses the best model obtained during the training process with a total of $n_{\text{test}} = ep_{\mathsf{test}} n_{\mathsf{ep}} = 5 \cdot 2000 = 10000$ steps. Training and tests are executed with both full and limited observation space.

The simulation follows the 3GPP specifications, including $\{12, 19\}$ clusters and $\{20, 20\}$ rays per cluster for LOS and NLOS cases, respectively [54]. It obtains channel realizations every $T_{\mathsf{s}} = 1\,\text{ms}$, which is the same value for TTI, with simulation episodes that last $T_e = 2\,\text{s}$. The UEs are randomly placed within $150\,\text{m}$ in relation to the target base station in the target cell, moving in a random direction with a speed $\left|\mathcal{N}(10, 3)\right|\,\text{m}\,\text{s}^{-1}$. The UE can turn its direction with a probability $P_{\text{turn}} = 0.2$ in each second.

Each UE has the requested throughput behavior defined by its slice type. However, the intensity of traffic is defined according to moderate and heavy traffic. With moderate traffic parameters defined as $\mu_{\mathsf{embb}} = 15\,\text{Mbps}$, $\mu_{\mathsf{urllc}} = 1\,\text{Mbps}$, and $\mu_{\mathsf{be}} = 15\,\text{Mbps}$. And heavy traffic parameters as $\mu_{\mathsf{embb}} = 25\,\text{Mbps}$, $\mu_{\mathsf{urllc}} = 5\,\text{Mbps}$, and $\mu_{\mathsf{be}} = 25\,\text{Mbps}$. The simulation always starts using moderate traffic parameters. In every 200 steps, the traffic switches between moderate and heavy traffic with half probability for each one. Table 3.3 defines the slice intents for each slice and traffic type, so every time the traffic switches between moderate and heavy traffic, the slice intents also switch. As defined previously in the description of BE UEs, the BE slice intents are all zero when their UEs are deactivated.

Table 3.4 establishes the reward weights for each slice's intent used in the reward calculation of Equation (3.18). These weight values were manually defined according to the slice intents' importance concerning other intents (similarly to [14]). For instance, the URLLC slice intents for the packet loss rate and buffer latency received the highest values because URLLC applications could be associated with critical applications, such as autonomous driving and factory automation [61]. The served throughput intent is more important than the latency and packet loss rate in the eMBB intents. Moreover, both eMBB and URLLC applications received higher weight values in relation to the BE, which has a low priority concerning the other slice types. These weight values could change according to the network operators' policies.

Table 3.3: Slice intents for moderate and heavy network traffic.

Slice type	Intents	Moderate traffic	Heavy traffic
eMBB:	Served throughput ($r^{\mathsf{req}}_{\mathsf{embb}}$)	10 Mbps	20 Mbps
	Avg buffer latency ($\ell^{\mathsf{req}}_{\mathsf{embb}}$)	20 ms	20 ms
	Packet loss rate ($p^{\mathsf{req}}_{\mathsf{embb}}$)	0.2	0.2
URLLC:	Served throughput ($r^{\mathsf{req}}_{\mathsf{urllc}}$)	1 Mbps	5 Mbps
	Avg buffer latency ($\ell^{\mathsf{req}}_{\mathsf{urllc}}$)	1 ms	1 ms
	Packet loss rate ($p^{\mathsf{req}}_{\mathsf{urllc}}$)	1×10^{-5}	1×10^{-5}
BE:	Long-term thr. ($g^{\mathsf{req}}_{\mathsf{be}}$)	5 Mbps	10 Mbps
	Fifth-perc. Thr. ($f^{\mathsf{req}}_{\mathsf{urllc}}$)	2 Mbps	5 Mbps

Table 3.4: Slice intent weights used in the reward calculation.

Intent weight	Weight values
eMBB served throughput $\left(w^{r}_{\mathsf{embb}}\right)$	0.2
eMBB latency $\left(w^{\ell}_{\mathsf{embb}}\right)$	0.05
eMBB packet loss rate $\left(w^{p}_{\mathsf{embb}}\right)$	0.05
URLLC served throughput $\left(w^{r}_{\mathsf{urllc}}\right)$	0.1
URLLC latency $\left(w^{\ell}_{\mathsf{urllc}}\right)$	0.25
URLLC packet loss rate $\left(w^{p}_{\mathsf{urllc}}\right)$	0.25
BE long-term thr. $\left(w^{g}_{\mathsf{be}}\right)$	0.05
BE fifth-perc. Thr. $\left(w^{f}_{\mathsf{be}}\right)$	0.05

3.4.2 Results

The simulation generates results using the test episodes with full and limited observation space. For conciseness, the network metric results show only the limited observation space. It also presents a reward comparison with the baselines and SAC RL agents using full and limited observation space. Fig. 3.4 shows the average requested throughput for each of the slices for episode 46, where each slice type defines the network traffic generated following its defined characteristics in Subsection 3.2.4.

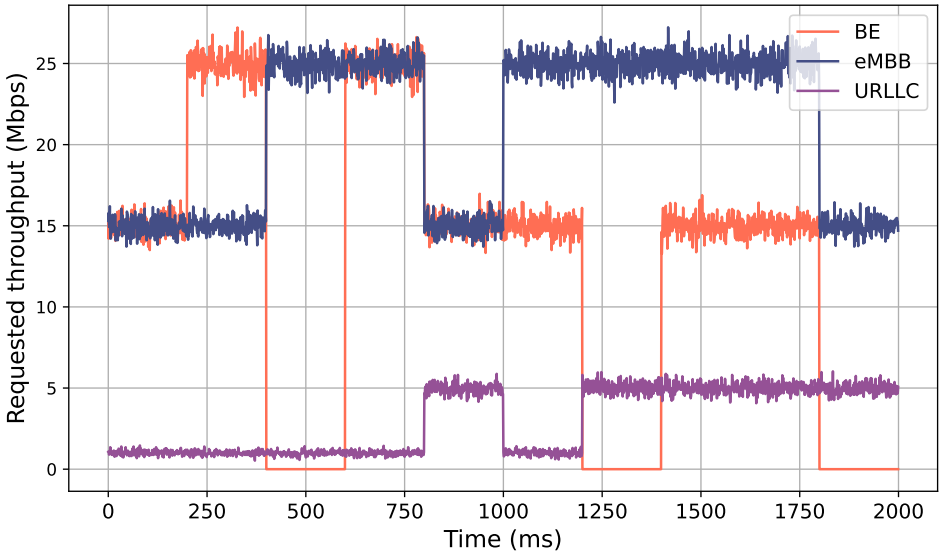

Figure 3.4: Average requested throughput for each slice in the test episode number 46 alternating between heavy and moderate traffic. In every 200 steps, the BE UEs can be activated or deactivated with half probability.

The two main QoS intents described in Table 3.3 for BE slice are long-term and fifth percentile throughput, and Fig. 3.5 and 3.6 show the respective results for these metrics. The long-term throughput result in Fig. 3.5 shows that the SAC RL agent learned to adapt its allocation behavior following QoS intents defined in Table 3.3, avoiding the allocation of radio resources when there is no requested traffic to the BE UEs (in accordance with Fig. 3.4). There are periods in which SAC agent is below the intent of $10\,\mathrm{Mbps}$. However, intent unfulfillment cannot be analyzed alone since the main SAC agents' goal is to maximize the reward so that it can sacrifice some QoS intent fulfillment in favor of the most important ones. Fig. 3.6 shows the result for the fifth-percentile throughput to the UEs BE, which presents similar results to the long-term throughput with the SAC RL agent being near the intents of 2 and $5\,\mathrm{Mbps}$ for

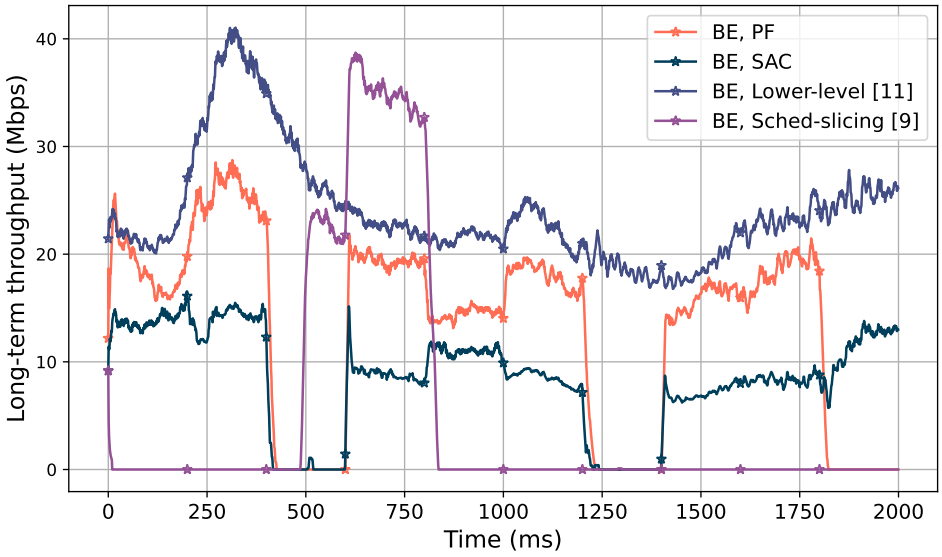

Figure 3.5: BE results for long-term throughput in the test episode number 46. QoS intents for long-term throughput are 5 Mbps and 10 Mbps for moderate and heavy traffic.

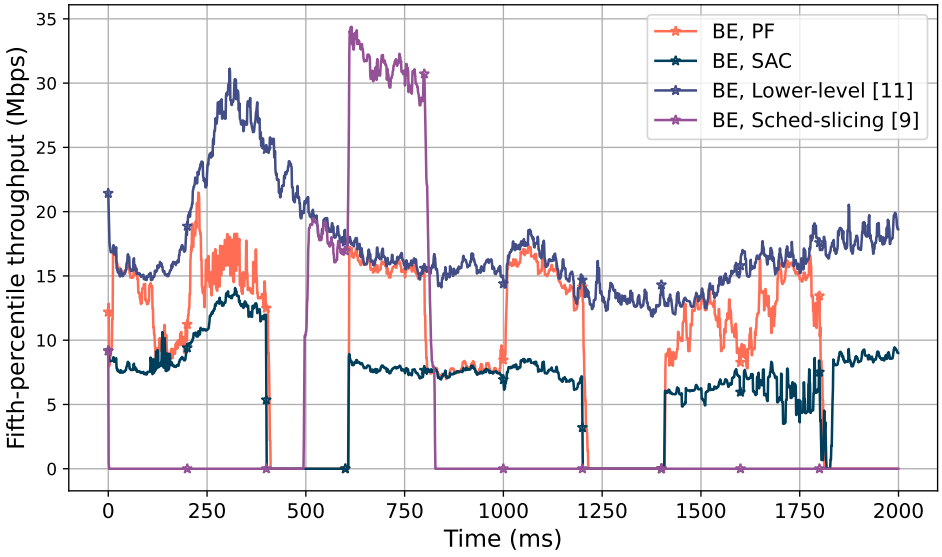

Figure 3.6: BE results for fifth-percentile throughput in the test episode number 46. QoS intent for fifth-percentile throughput are 2 Mbps and 5 Mbps for moderate and heavy traffic.

moderate and heavy traffic, respectively.

Fig. 3.7 shows the cumulative distribution function (CDF) of the served throughput for UEs from eMBB and URLLC slices. The eMBB intents to the served throughput are 10 and 20 Mbps for moderate and heavy traffic, and the URLLC intents are 1 and 5 Mbps for moderate and heavy traffic. Fig. 3.7 indicates that almost all values obtained using the proposed SAC RL for the eMBB served throughput are above the QoS intents since the throughput for eMBB is one of the intents with higher priorities. Analyzing the URLLC, the SAC also keeps most traffic above the minimum intents. The Lower-level baseline prioritized the URLLC and BE, providing much more throughput than needed (as shown in Fig. 3.5 for BE) and keeping the eMBB slice with almost zero served throughput even with the definition of higher weight values for eMBB than for BE in Subsection 3.3.5. The small weight value for throughput compared to packet loss and latency weights justifies this behavior and better network conditions for BE UEs in the test set. In contrast, the Sched-slicing baseline prioritized the eMBB and URLLC slices, providing high served throughput values in relation to their QoS intents.

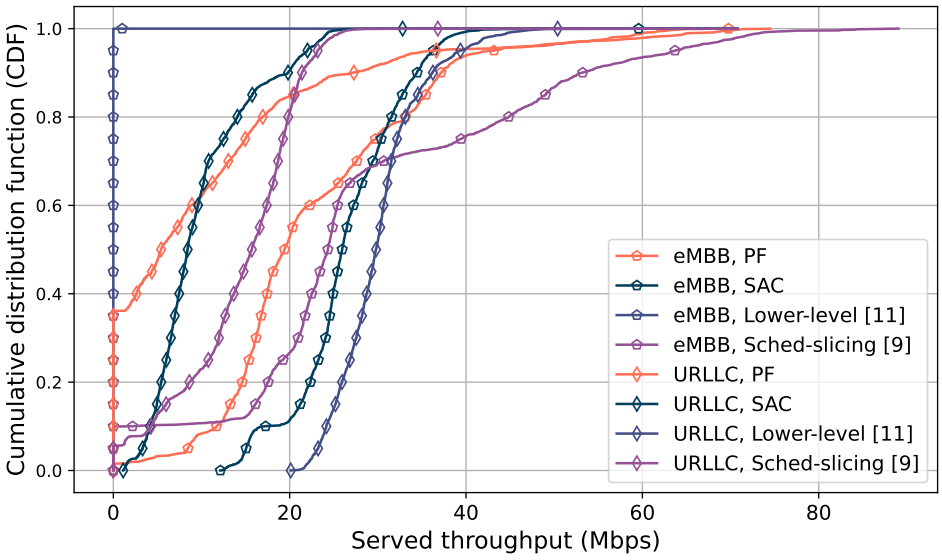

Figure 3.7: CDF of the served throughput for eMBB and URLLC slices in the test episode number 46. URLLC and eMBB slices' intents define a served throughput of 1 Mbps and 10 Mbps, for moderate traffic. For heavy traffic, URLLC is 5 Mbps, and eMBB is 20 Mbps. The proposed RL agent provided the served throughput defined in the intents for both URLLC and eMBB.

Fig. 3.8 shows the average buffer latency result for both eMBB and URLLC slices with an intent of 20 ms to the former and 1 ms to URLLC. The Lower-level baseline results were

removed for eMBB due to the high latency value obtained (about the maximum latency allowed of $100\,\mathrm{ms}$) due to the small served throughput (Fig. 3.7) and the high packet loss rate that will be presented later. For the URLLC, the Lower-level scheduling, PF, and RL SAC methods keep the latency lower than required over the simulation, with only the Sched-slicing baseline obtaining latency results above $1\,\mathrm{ms}$ in the period from $0\,\mathrm{ms}$ to $800\,\mathrm{ms}$. When analyzing the results for the eMBB slice, all baselines fulfill the intent of $20\,\mathrm{ms}$ with the exception of the Lower-level baseline.

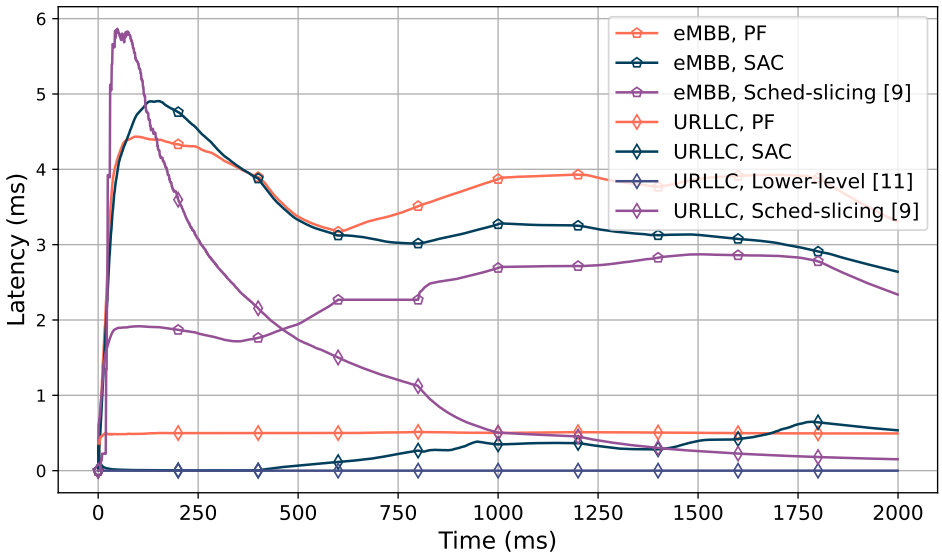

Figure 3.8: Average buffer latency for eMBB and URLLC slices in the test episode number 46. URLLC and eMBB slices' intents define an average buffer latency of $1\,\mathrm{ms}$ and $20\,\mathrm{ms}$, respectively, for moderate and heavy traffic. The RR did not fulfill the latency intent to URLLC, and the RL agent fulfilled both URLLC and eMBB latencies with the smallest latencies obtained in relation to other baselines.

The packet loss rate is presented in Fig. 3.9. All baselines and the SAC agent keep a zero packet loss rate, fulfilling the QoS intent to URLLC UEs. When evaluating eMBB slice that has higher requested traffic by UEs, both the baselines and the SAC agent cannot keep the packet loss rate below 0.2 (specified in the intents for eMBB) most of the time. Moreover, the SAC RL agent presents better performance among the baselines with the lowest packet loss rate for eMBB. The Lower-level baseline presented a high packet loss rate that corroborates the preference for the URLLC and BE slices, as previously stated in the served throughput and latency analysis.

Equation (3.18) defines the intent drift reward value as a distance to fulfill the QoS intents

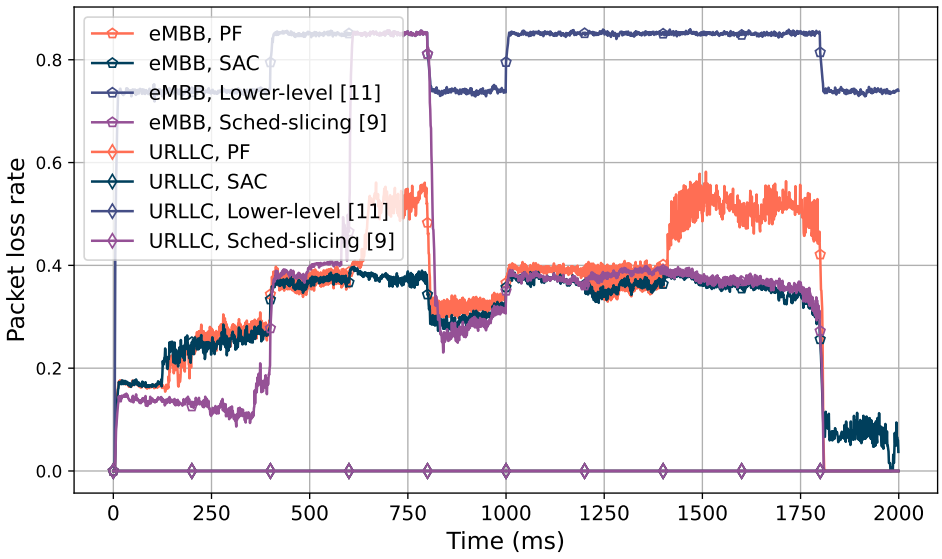

Figure 3.9: The average packet loss rate for eMBB and URLLC slices in the test episode number 46. All baselines fulfilled the URLLC intent. URLLC and eMBB slices' intents define a packet loss rate of 1×10^{-5} and 0.2, respectively, for moderate and heavy traffic. When considering the eMBB slice, none of the agents could fulfill the packet loss intent during all the simulation time, but the RL agent provided the best performance.

weighted by their importance/priorities defined in Table 3.4. It is challenging to assess the SAC RL and the baseline agents' performance in a multi-objective scenario with many intents to fulfill, so the defined reward is an important metric to evaluate the distance to satisfy the QoS intents specified by the operator. Fig. 3.10 shows the cumulative reward obtained for the five test episodes using the baselines and the proposed SAC RL agent. The Lower-level baseline obtained the worst result due to its inability to prioritize the eMBB slice. The Lower-level and Sched-slicing agents obtained a lower reward than the proposed SAC RL and PF agents due to the designed reward aiming to maximize specific network metrics instead of learning to fulfill the slice intents. The SAC RL agent outperformed the baselines over all the test episodes, fulfilling the slices' intents or being near to fulfilling them compared to the baselines. However, in the slice intent result, the proposed agent performed worse in specific moments and metrics. However, considering all the intents, the agent balanced which intents to fulfill when more radio resources were needed to meet all slices.

Fig. 3.10 also makes a comparison of the proposed agent using full and limited observation space. It shows that even using less information, the limited observation space obtained

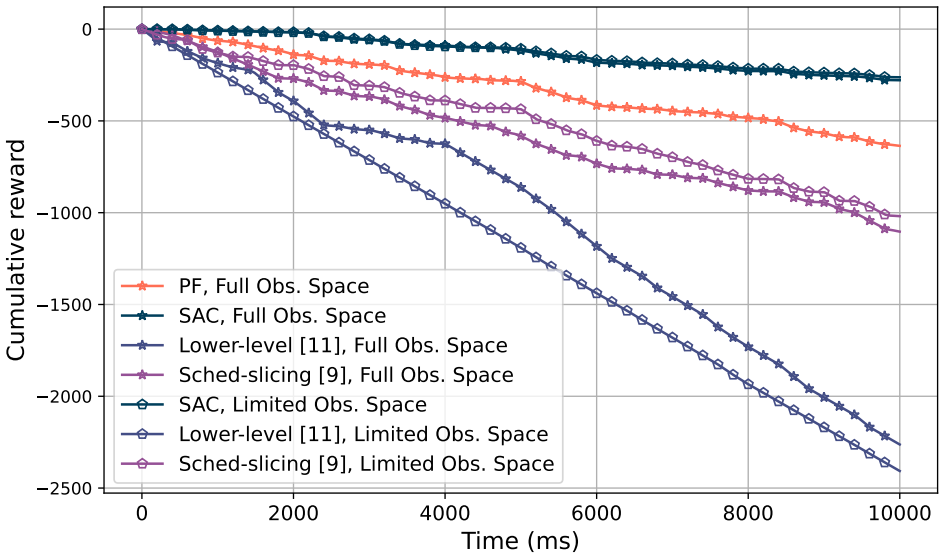

Figure 3.10: The cumulative reward for the five test episodes using SAC RL and other baseline agents, considering limited and full observation space. The proposed RL agent outperformed the baselines using full and limited observation space.

about the same performance as the full observation space. With a limited observation space, the number of variables in the observation space becomes independent of the number of UEs in the system, making the system scalable to support a greater and variable number of UEs in the scenario. With a reduced number of variables in the observation space, the hyperparameters' optimization process uses a four times smaller neural network architecture than the architecture used in the full observation space (as shown in Table 3.2), decreasing the computational costs and increasing the processing speed.

In case the number of slices increases to 5 (2 eMBB, 2 URLLC, and 1 BE), the number of variables using the limited observation space would be $|\boldsymbol{O}_n| = 2|\boldsymbol{r}_{\text{embb}}|+2|\boldsymbol{r}_{\text{urllc}}|+|\boldsymbol{r}_{\text{be}}|+5|\boldsymbol{s}_v| = 2 \cdot 3 + 2 \cdot 3 + 2 + 5 \cdot 9 = 59$ that is still less than half the number of variables in the full observation space, as discussed in Subsection 3.3.2. Another aspect to consider is that even with a high number of slices in the whole network, the number of RAN slices supported per base station tends to be limited by its available channel capacity. For instance, in the presented simulation scenario, the requested QoS intents are not fulfilled throughout the simulation period, as presented in Fig. 3.10 (when the reward is not zero), indicating that it is not recommended adding another RAN slicing in this base station to compete with the slices that already exist to avoid intent drifts.

Chapter 4

Intent-based RRS using MARL for various network scenarios

In the previous chapter, an intent-aware RRS was proposed to deal only with fixed network scenarios containing eMBB, URLLC, and BE slice types. Here, this chapter presents an enhanced intent-based RRS using MARL to perform inter- and intra-slice scheduling in RAN slicing for various network scenarios. Unlike the proposed method in the previous chapter, here, the proposed method considers network scenarios that comprehend combinations of 10 slice types, channel trajectories, number of active slices and UEs, and UE characteristics. The proposed method utilizes an RL agent for inter-slice scheduling, distributing radio resources among slices. It uses a MARL with shared parameters to the intra-slice schedulers, where each slice has its intra-slice scheduler with an RL agent, selecting a scheduling algorithm among round-robin, proportional-fair, and maximum throughput to distribute the assigned radio resources among the slice UEs. First, the intent-based RRS learns to fulfill the slice intents of the high-priority slices and then learns to meet other regular slices. Prioritizing high-priority slices is implemented in the reward mechanism without needing weight optimization for each network scenario in opposition to the method presented in the previous chapter, eliminating the need for weight optimization when dealing with different network scenarios. In this chapter, the method described in Chapter 3 will be cited as [40].

A realistic simulator was developed to assess the effectiveness of the proposed method under varying conditions and network configurations. For channel generation, this simulator uses QuaDRiGa, which creates spatially consistent channels. The developed simulator also supports various traffic models and UEs characteristics according to the types of slices. The

proposed method and baselines are tested in different network scenarios to assess their capacity to deal with various applications and intents. There are three different evaluation scenarios: (i) Training several agents, each one specialized in handling a specific network scenario; (ii) Training a single agent on all network scenarios; and (iii) Using transfer learning when dealing with unseen network conditions. The first approach evaluates whether specialized RRSs can handle unseen channel conditions. The second approach determines the generalization capabilities of RRSs trained on a large set of network scenarios. Finally, the third approach understands if the RRS can use experiences gathered from different network scenarios to learn how to handle unseen network scenarios. The developed software is publicly available to facilitate future work and comparisons.

The main contributions in this chapter are summarized as follows:

- Design and development of an intent-based RRS using an RL agent handling inter-slice scheduling, and MARL with shared parameters to implement intra-slice schedulers. The proposed method prioritizes high-priority slices without optimizing predefined weights for each network scenario.

- The proposed method handles various network scenarios, fulfilling their intents and prioritizing the high-priority slices when needed.

- Improvement of the intent-drift reward method proposed in [40] to observe intent drift variations when intents are fulfilled and avoid future intent violations.

- Explore generalization for diverse network scenarios and the use of previous network scenario experiences in training for unseen network scenarios.

- Evaluate the proposed method against baselines using various network scenarios with multiple slice types, number of active slices and UEs, UEs characteristics, and channel trajectories.

- The simulation code is publicly available to facilitate reproducibility and comparison with other methods[1].

This chapter is organized as follows: Section 4.1 describes the related works, emphasizing the RL RRS methods for RAN slicing. It also compares the main contributions of this proposed

[1] https://github.com/lasseufpa/intent_radio_sched_multi_slice

method concerning previous work. Section 4.2 presents the adopted communication system model and RRS system in a scenario with RAN slicing. Section 4.3 presents the proposed intent-based RRS using a MARL agent to perform inter- and intra-slice allocation. Section 4.4 presents the results obtained in three different evaluation scenarios that assess the capacity of the proposed method to deal with varying network scenarios.

4.1 Related work

The ability of data-driven approaches to learn models directly from the data produced by the network is extremely valuable when considering RAN functions. The RAN is a data-rich environment that collects data through radio measurements as well as user devices and core network [11]. Data-driven RRS methods usually utilize RL techniques that can learn from large amounts of data efficiently and represent an unsupervised learning method that does not require correct pre-computed labels [6].

The employment of RRS using RL techniques for RAN slicing was approached in related works [12–19, 40]. The solutions proposed in [12–15] focus on maximizing/minimizing network metrics, such as maximizing the achieved throughput of eMBB slices and minimizing buffer occupancy or latency of URLLC slices. For example, work [12] proposes an O-RAN compliant RL PPO method for both inter- and intra-slice schedulers. Considering the types of slices of eMBB, URLLC, and mMTC. The inter-slice scheduler using RL is responsible for selecting the number of RBGs for each slice while it also selects the intra-slice scheduling algorithm for each slice among the options round-robin, proportional-fair, and maximum-throughput. The implemented reward function, responsible for guiding the RL agent learning process, focuses on maximizing the throughput rate to the eMBB slice and the transport blocks to the mMTC slice while minimizing the buffer size to the URLLC slice.

The problem with RRS methods focusing on the minimization/maximization of network metrics is the unclear slice objectives. There is no specification of slice intents through network requirements that enables verifying whether the intents have been fulfilled. For example, when the RRS method in [12] maximizes the throughput rate to the eMBB slice, it is impossible to verify if the technique is reaching its maximization objective because there is no specification of a minimum throughput rate to serve the eMBB slices. Therefore, the throughput rate value obtained by the method varies according to the network conditions. It is difficult to verify

whether the maximization objective is met since there is no optimum method for comparison. Moreover, when considering intent-based systems based on TM Forum [9], these minimization/maximization objectives are incompatible with the intent's definition of clearly specifying the requirements the intent-based system should fulfill.

The prior work [16–19] consider RL RRSs with reward functions based on the SLA satisfaction rate. These methods specify the slice objectives using QoS metrics in a SLA. The work [18] considers the maximization of spectral efficiency while meeting throughput and latency requirements in a scenario with voice over LTE (VoLTE), Video, and URLLC slices. In [19], the presented method considers the latency requirements for two slices with different latency requirement values specified in the SLA while minimizing the number of allocated RBGs, reducing energy consumption. In [17], it considers the eMBB and mMTC slices. the eMBB requirements are the maximum average queue per guaranteed bit rate (GBR) and non-GBR, the authorized and compliant capacity for GBR in RBs per subframe. The mMTC slice considers the maximum delay per user. The results are assessed in four different network scenarios that comprehend a maximum of five slices with different combinations of the same eMBB and mMTC slice types. In [16], it defines the throughput and latency requirements for each slice in a network scenario with five slices representing five different applications: messaging, app, audio, video, and best effort.

Although the related works [16–19] provide mechanisms for dealing with the slice requirements defined in a SLA, they are still insufficient to provide support for slice intents when considering an intent-based system. In [16–18], the SLA violations are represented as a binary value indicating the fulfillment or not of a requested QoS requirement. However, there is no indication of how far the RRS agent is from fulfilling their requirements, which is an essential point of intent-based systems [9, 34], usually represented by an intent drift [40], since it gives the RRS agent the possibility to assess whether a given action improves/deteriorates the current slice condition. In addition, it enables the report of a more accurate status to the intent owners interested in the slice intent fulfillment. Another critical aspect left out is the slice intent prioritization to cope with high-priority slices in scenarios where the experienced channel capacity is insufficient to fulfill all the requested slice intents.

The previous work [40] (presented in Chapter 3) proposes the first intent-aware RRS using RL for RAN slicing with support for eMBB, URLLC, and BE slice types. It focuses on fulfilling the slice intents defined in a common intent model [10] and allows for changing the

slice intents without retraining the RRS agent. The proposed intent-drift reward offers the RL agent a distance to fulfill the intents, which allows learning to minimize the distance even when the available radio resources are insufficient to fulfill all the requested intents. It also provides a weight-based prioritization for slice intents to protect high-priority intents in scenarios with high concurrency for radio resources.

Despite the support introduced for slice intents in [40], it does not investigate the generalization capabilities of the proposed method to different network scenarios. The UEs characteristics and mobility pattern do not change in the episodes, and the trained RL policy can only deal with variations in the slice intents of the same slice types predefined in the simulation. In addition, using weights to define intent priorities requires joint training and optimization to find the best weight combinations for each network scenario. Therefore, whenever the method is implemented in a different network scenario, the predefined weights must change to reflect the new intent priorities. The proposed intent-drift reward accounts only for the distance to fulfill requirements when slice intents are not fulfilled. However, there is no indication of degrading performance in fulfilled slice intents (helping to prevent future slice violations). Finally, the presented solution uses a fixed round-robin algorithm in the intra-slice scheduler instead of exploring different strategies that could enhance the RRS performance.

The main problems related to the support of slice intents in [12–19, 40] are summarized below:

- The methods focusing on maximizing/minimizing specific network metrics [12–15] do not provide sufficient mechanisms to deal with slice intents since it is impossible to define when a given slice intent is fulfilled. Moreover, these maximization/minimization objectives do not comply with the TM forum definition of intents [9].

- Although related works [16–19] provide RRS methods based on the SLA satisfaction rate with QoS requirements, these mechanisms offer incomplete support to slice intents since they do not provide a metric to observe intent performance and visualize improvement/degradation over time. In addition, they also do not provide prioritization mechanisms to protect high-priority slice intents when the amount of radio resources at a given moment of the network conditions is insufficient to fulfill all the slice intents.

- The previous work [40] defines an intent-aware RRS using RL for RAN slicing. Besides its support to slice intents through the intent-drift reward and weight-based priorities, it

lacks support for different network scenarios since it requires redefining the weight values for each intent in each network scenario, and there is no clear direction of how adapting these methods for different combinations of slice types other than the predefined ones.

- None of the related works [12–19, 40] discuss the usage of the proposed methods in different network scenarios in which the slice types, number of active slices, number of UEs in the slices, UEs characteristics, and channel generation varies. There is no investigation about the generalizability of the presented methods to different network scenarios or even the use of previously learned experiences in network scenarios that were not seen during the training.

Concerning the highlighted issues above, it is proposed an intent-based RRS for RAN slicing utilizing MARL. It improves the calculation of intent-drift reward from [40] to refine violation prevention and design a RRS with homogeneous entries and outputs to support different network scenarios. It investigates the generalizability of the proposed method and baselines in network scenarios not seen in the training. It also explores the transfer learning from different network scenario experiences to fine-tune the agent to deal with unseen network scenarios.

4.2 Communication system model and problem formulation

To simplify notation, in the following, a single-input and single-output (SISO) system is assumed with a single base station operating at carrier frequency f_{c} and providing service to $\upsilon = 1, 2, 3, \ldots, \Upsilon$ RAN slices. The base station has $u = 1, 2, 3, \ldots, U$ UEs connected at the same time, where each UE u has a single antenna and is assigned to a specific slice υ. Each slice υ contains a set of U_{υ} UEs. Despite the SISO assumption, the proposed model is general and applies to other RF transmission schemes, such as MIMO, without changes in the proposed system.

The base station has a B MHz bandwidth, divided into G RBs, and RBs are grouped into R RBGs that is considered the minimum radio resource allocation unit in the scheduling system. The TTI t is measured in ms and represents the minimum time unit considered in the scheduling process. In each TTI, the scheduler allocates all RBGs in a specific simulation step n. Each step n takes t ms with $t_n = t \cdot n$ representing the time from step 1 to n.

The RRS system with RAN slicing performs inter- and intra-slice scheduling. The inter-slice scheduler distributes the available R RBGs to the active slices Υ_{act}, and the intra-slice

scheduler is responsible for distributing the RBGs assigned by the inter-slice to the slice UEs. A MARL system is proposed to perform the inter- and intra-slice scheduling in a system with RAN slicing. It assumes an uplink RRS formulation where the base station defines the RBGs each UE uses, but it also applies for downlink.

4.2.1 Channel modeling

A TDD transmission protocol is used that receives pilots from UEs sent through the uplink to obtain the base station's CSI. Each base station obtains a perfect channel estimation for each UE. UEs have their spectral efficiency values varying with time and frequency, hence different from [12, 40, 46, 47] where the same UE's spectral efficiency value is assumed for all RBGs.

The channel simulation utilizes QuaDRiGa software [51] to consistently generate UE channels in space and frequency. It generates spatially and correlated channels from statistical models, including experimentally validated channel models using 3GPP 38.901 UMa [53] statistical models based on dual-slope path loss with significant inter-parameter correlations. Furthermore, given that the base station allocates the g-th RB to the u-th UE, the signal-to-noise ratio (SNR) perceived by the UE can be expressed as follows

$$\gamma_{u,g} = \frac{\alpha_u \, p_{u,g} \, |h_{u,g}|^2}{\sigma_u^2}, \tag{4.1}$$

where $p_{u,g}$ is the allocated transmit power to the UE u in the g-th RB, α_u is the effect of path gain and shadowing experienced by the u-th UE, $h_{u,g}$ is the effective channel for UE u in the g-th RB, and σ_u is the noise power experienced by the u-th UE. This way, the spectral efficiency $\mathsf{SE}_{u,g}(n)$ to RB g and UE u is defined as

$$\mathsf{SE}_{u,g}(n) = \log_2(1 + \gamma_{u,g}). \tag{4.2}$$

This work employs the LOS and NLOS using the QuaDRiGa channel model, as documented in [62].

4.2.2 Radio resource scheduling with RAN slicing

The main objective of a RRS in a RAN slicing scenario is to distribute the RBGs among the slices and UEs to fulfill the SLA defined through a group of slice intents, giving high priority to the most important slices through resource reservation and avoiding, or ideally eliminating,

any slice intent violation. The number of RBGs allocated by the inter-slice scheduler for each slice in a simulation step n is represented in the vector

$$\boldsymbol{R}^{\mathsf{inter}}(n) = [R_1(n), R_2(n), \ldots, R_\Upsilon(n)], \tag{4.3}$$

with $R_\upsilon(n)$ representing the number of RBGs allocated to slice υ at step n.

For simplicity, it is assumed that all RBGs are allocated during the scheduling process. Therefore, the RRS obeys to

$$\sum_{\upsilon=1}^{\Upsilon} R_\upsilon(n) = R. \tag{4.4}$$

In case the slice υ does not have sufficient data to use all the allocated RBGs, the extra RBGs are reserved for the slice but not used. These allocation decisions can be easily converted to the 3GPP specification to the radio resource management (RRM) policy ratio [63].

The inter-slice scheduler defines the number of RBGs $R_\upsilon(n)$ to the slice υ at step n, and then the intra-slice scheduler allocates this $R_\upsilon(n)$ RBGs available among the U_υ slice UEs in the vector

$$\boldsymbol{R}_\upsilon^{\mathsf{intra}}(n) = [R_\upsilon^1(n), R_\upsilon^2(n), \ldots, R_\upsilon^{U_\upsilon}(n)], \tag{4.5}$$

where $R_\upsilon^u(n)$ represents the number of RBGs allocated to the UE u in the slice υ at step n. The intra-slice scheduler also obeys to

$$\sum_{u=1}^{U_\upsilon} R_\upsilon^u(n) = R_\upsilon(n). \tag{4.6}$$

The RRS performance is evaluated considering the network metrics defined in the proposed slice intents. The served throughput is defined as the maximum throughput in megabits per step (Mbps) that could be achieved by a UE u in the slice υ taking into account the number of RBGs $R_\upsilon^u(n)$ allocated and its spectral efficiency values $\mathsf{SE}_u(n)$

$$r_\upsilon^u(n) = \left\lfloor \frac{B \sum_{g \in G^u} \mathsf{SE}_{u,g}(n)}{\mathsf{PS}_\upsilon \, G \, 10^6} \right\rfloor \mathsf{PS}_\upsilon, \tag{4.7}$$

where G^u represents the RBs allocated to a UE u, PS_υ is the packet size in bits for UEs in the slice υ. It does not consider the MCS from each RB in the throughput calculation for clearness.

The data available to send in the UE buffer limits the served throughput. Therefore, the effective throughput $e_\upsilon^u(n)$ represents the data throughput sent over the network

$$e_\upsilon^u(n) = \min(r_\upsilon^u(n), b_u(n)), \tag{4.8}$$

with $b_u(n)$ representing the data available of UE u at step n in Mbit. As a consequence, the effective throughput is always $e_\upsilon^u(n) \leq r_\upsilon^u(n)$ with $e_\upsilon^u(n) = r_\upsilon^u(n)$ when $b_u(n) \geq r_\upsilon^u(n)$.

The buffer occupancy rate $b_u^{\mathsf{occ}}(n)$ is defined as

$$b_u^{\mathsf{occ}}(n) = \frac{b_u(n)}{b_\upsilon^{\mathsf{max}}}, \tag{4.9}$$

where $b_\upsilon^{\mathsf{max}}$ is the maximum UE buffer capacity in slice υ. Packets are discarded whenever the buffer is full or the packet latency exceeds the maximum allowed latency $l_\upsilon^{\mathsf{max}}$. For this reason, these packets are accounted for in the dropped data $d_u(n)$ that represents the size of the packets dropped in step n. The packet loss rate $p_u(n)$ is calculated over a window interval of w steps

$$p_\upsilon^u(n) = \begin{cases} \frac{\sum_{i=(n-w)}^{n} d_u(i)}{b_u(n-w)+\sum_{i=(n-w)}^{n} \iota_\upsilon^u(i)}, & \text{if } n \geq w \\ \frac{\sum_{i=1}^{n} d_u(i)}{b_u(1)+\sum_{i=1}^{n} \iota_\upsilon^u(i)}, & \text{Otherwise} \end{cases}, \tag{4.10}$$

where $\iota_\upsilon^u(n)$ is the requested throughput by UE u in slice υ at step n. The requested throughput depends on the slice υ that the UE is associated with since the traffic behavior of UEs of the same slice is similar.

The average time that packets have waited in the buffer of UE u is represented as

$$\ell_u^\upsilon(n) = \frac{\sum_{i=0}^{l_\upsilon^{\mathsf{max}}} i \boldsymbol{l}_n^u(i)}{\sum_{i=0}^{l_\upsilon^{\mathsf{max}}} \boldsymbol{l}_n^u(i)}, \tag{4.11}$$

where $\boldsymbol{l}_n^u = [l_0, l_1, \ldots, l_{l_\upsilon^{\mathsf{max}}}]$ is a vector with size $l_\upsilon^{\mathsf{max}} + 1$ representing the packets' latency on buffer for user u at step n, and $l_{l_\upsilon^{\mathsf{max}}}$ represents the number of packets that have waited for $l_\upsilon^{\mathsf{max}}$ TTIs in the buffer.

Slice metrics average the UEs metrics associated with the target slice. For example, the effective throughput $e_\upsilon(n)$ for slice υ is the average effective throughput from slice υ UEs U_υ.

4.2.3 Slice intents and requirements

Related works [12–19,40] usually define three different slice types based on 5G use cases: eMBB, URLLC, and mMTC. This definition hides the diversity of the application inside each of these use cases. For example, the eMBB use-case can contain applications such as video streaming and cloud gaming, where their network requirements are very distinct. An application running a 4K video streaming on Netflix has a throughput requirement of $15\,\mathrm{Mbps}$ [64]. At the same time, a Nvidia cloud gaming application has a throughput requirement of $45\,\mathrm{Mbps}$ and also a latency requirement of $40\,\mathrm{ms}$ latency for the best experience [65]. The adopted RRS

needs to deal with these applications differently to fulfill their intents instead of grouping them in the same slice and fitting them with identical objectives.

This work proposes a definition of the slice type coupled with end-user applications, and then the RRS needs to deal with different network slice intents. The main goal is to handle these slice intents independently of the network's combination of active slice types. To accomplish that, each type of slice has to define its intents based on three main network metrics: effective throughput, buffer latency, and packet loss rate. Each slice type must have one or more intent definitions, but the intent requirements do not need to consider the same metrics since each application has its characteristics. The diversity of applications is essential to ensure the proposed RRS scheme can deal with different types of slices and combinations of active slices.

Each slice υ can have up to three different slice intents in each simulation step n: the effective throughput intent requirement $e_{\upsilon}^{\mathsf{req}}$ in Mbps, the buffer latency $\ell_{\upsilon}^{\mathsf{req}}$ in ms, and the packet loss rate $p_{\upsilon}^{\mathsf{req}}$. Each slice υ has a binary active intent indicator for effective throughput m_{υ}^{e}, buffer latency m_{υ}^{ℓ}, and packet loss rate m_{υ}^{p}, which indicates if the slice considers the target network metric in its intents. Therefore, the slice υ intents are fulfilled every time the following conditions are achieved:

$$\begin{aligned}
\frac{\sum_{u=1}^{U_{\upsilon}} e_{\upsilon}^{u}(n)}{U_{\upsilon}} &\geq e_{\upsilon}^{\mathsf{req}}, \text{if } m_{\upsilon}^{e} = 1 \\
\frac{\sum_{u=1}^{U_{\upsilon}} \ell_{\upsilon}^{u}(n)}{U_{\upsilon}} &\leq \ell_{\upsilon}^{\mathsf{req}}, \text{if } m_{\upsilon}^{\ell} = 1 \\
\frac{\sum_{u=1}^{U_{\upsilon}} p_{\upsilon}^{u}(n)}{U_{\upsilon}} &\leq p_{\upsilon}^{\mathsf{req}}, \text{if } m_{\upsilon}^{p} = 1
\end{aligned} \tag{4.12}$$

Whenever one or more slice intents are not fulfilled, the slice SLA accounts for a violation. The RRS function in an intent-based system with RAN slicing is to prevent/minimize the intent violations.

4.3 Proposed Intent-based RRS agent using MARL

This work proposes an intent-based RRS for a scenario with RAN slicing using MARL based on TM Forum IG1253 [9] definitions for an intent-based network. TM Forum defines intent as the formal specification of all expectations, including requirements, goals, and constraints given to a technical system. Furthermore, intents should be expressed with formality and complete semantics and vocabulary, avoiding ambiguities in the meaning of an intent (the

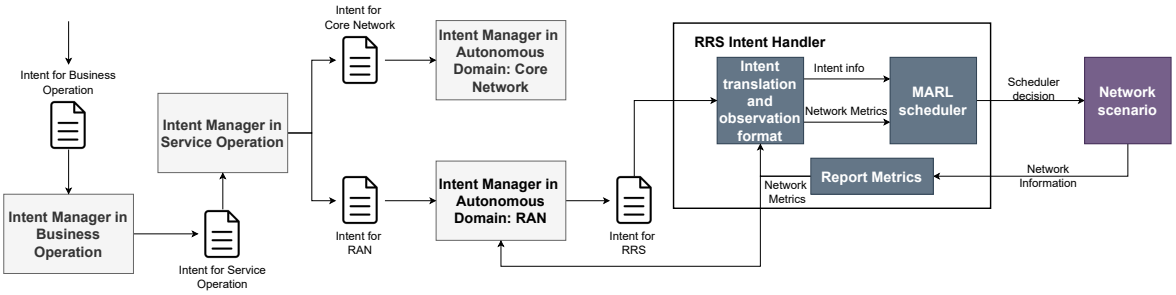

Figure 4.1: An intent-based system with an RRS intent handler responsible for receiving slice intents and network information to generate schedule actions to fulfill the provided intents. The RRS intent handler is based on MARL.

sender and receiver of intent must be in perfect agreement about its interpretation).

Fig. 4.1 shows an intent-based system focusing on the RRS for RAN slicing. The intent manager in business operations receives a high-level intent that can contain the slice type description and generates an intent for service operation containing requirements about service KPIs and customer experience metrics. The intent manager in the service operation receives the previous intent and creates intents to the RAN and core network intent managers so that the generated intents could satisfy the intent for the service operation. Finally, in the RAN domain, the intent manager receives the RAN intents and coordinates their different functions, such as the RRS, to fulfill the intent for RAN requirements. Communication between intent managers is possible through a common intent model [10] that specifies the intent description and reports using common vocabulary and semantics.

This work proposes an RRS intent handler that receives the intent for RRS (following the common intent model [10]) specifying the intents for each slice to be fulfilled by the RRS operations. The RRS intent handler is responsible for receiving the intent for RRS and generating radio scheduler decisions to fulfill the intents or minimize the number of violations in scenarios where the radio resources available are insufficient to meet all the slice's intents. The intent translation and observation format function is responsible for translating the received slice intents to the intent information represented in the effective throughput intent requirement e_v^{req}, the buffer latency ℓ_v^{req}, and the packet loss rate p_v^{req} described in Subsection 4.2.3. In addition, it also creates a vector containing network metrics and intent fulfillment information to the MARL scheduler.

The proposed MARL scheduler utilizes the intent information and network metrics to

generate a scheduler decision indicating the RBGs to allocate for each slice's UE aiming at fulfilling the slice intents. The RRS intent handler applies the scheduler decision in the network scenario, providing updated network information due to the scheduler action. Finally, the report metrics function is responsible for calculating network metrics to report to the RAN intent manager, provide information about the slice's intent fulfillment, and also provide network metrics to the observation format used as input in the MARL scheduler.

4.3.1 Intent-based RRS using MARL

This work proposes a MARL method to perform inter- and intra-slice scheduling in a RAN slicing scenario similar to [40]. The inter-slice scheduler provides an action representing the number of RBGs for each slice, and the intra-slice scheduler selects an algorithm method among round-robin, maximum-throughput, and proportional-fair [22]. The proposed MARL agent aims to fulfill the intents of active slices in the network, considering different scenario combinations, including variations in the number of slices, type of slices, and number of UEs associated with each slice.

The proposed method for RRS implements an inter-slice scheduler utilizing a PPO RL method with a dedicated policy. In contrast, intra-slice schedulers utilize a MARL PPO with parameter sharing (shared policy) [66] as depicted in Fig. 4.2. Concerning the RL inter-slice scheduler, it formulates the system as an MDP using a tuple $(S, \boldsymbol{A}^{\mathsf{inter}}, \mathsf{RW}^{\mathsf{inter}}, P, \rho_0)$, where S is the set of all valid states, $\boldsymbol{A}^{\mathsf{inter}}$ is the set of all valid actions for the inter-slice scheduler, $\mathsf{RW}^{\mathsf{inter}}$ is the reward function, P is the transition probability function, and ρ_0 is the initial state distribution of the system [36]. In a time step t, the agent in a state s_t takes action a_t and reaches the next state s_{t+1} receiving the reward $\mathsf{RW}^{\mathsf{inter}}(t)$. Rewards are numerical values given to the agent's actions to represent if the chosen action was effective, and the agent aims to maximize the long-term cumulative reward [36]. In the inter-slice case, the reward represents the fulfillment of the slices' intents. The inter-slice agent follows a policy $\pi(\cdot|s_t)$ defined as a distribution over actions given the current state s_t.

A PPO RL method is adopted for being a well-established method to deal with continuous control tasks and the stability and reliability of trust-region methods with simpler implementation [37]. More specifically, it adopts the PPO implementation using a clipped surrogate objective

$$L_t^{\mathsf{clip}}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \mathsf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big], \tag{4.13}$$

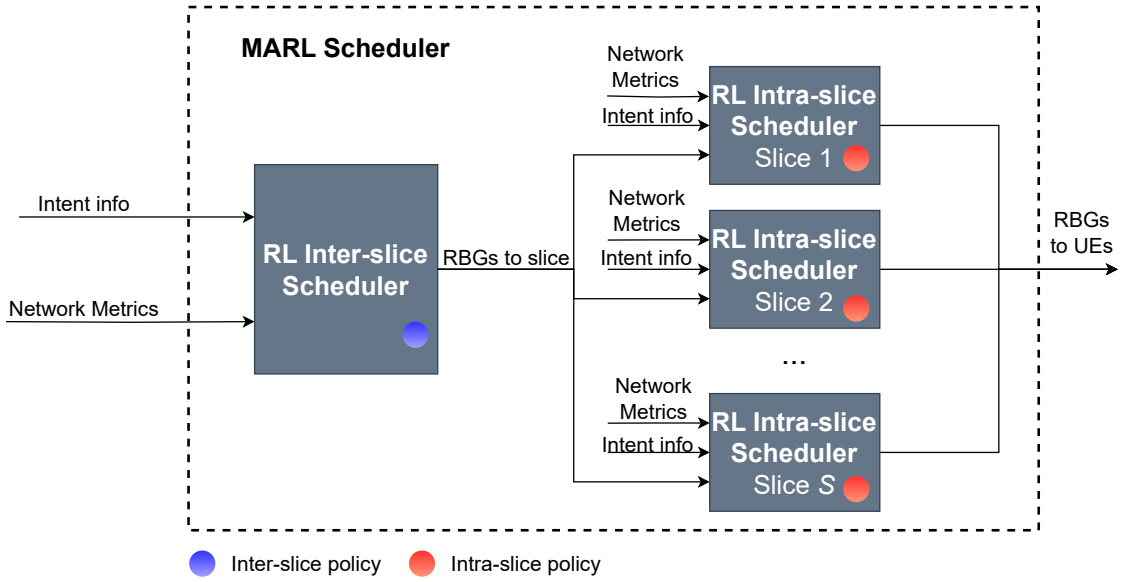

Figure 4.2: Proposed intent-based MARL architecture to perform inter- and intra-slice scheduling in different network scenarios. The RL inter-slice scheduler has a dedicated policy, while the RL intra-slice schedulers utilize a shared policy.

where the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)} \tag{4.14}$$

represents the current policy π_θ changes in relation to the old policy $\pi_{\theta\text{old}}$. The estimated advantage function A_t utilizes generalized advantage estimation [37], measuring how much better or worse a particular action is compared to the agent's average performance. It clips the $r_t(\theta)$ value outside the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is an hyper-parameter. It takes the minimum between the clipped and unclipped objective, so the final objective is a lower bound on the unclipped objective. Clipping the objective between the defined interval improves the stability and reliability when updating the policy values.

Finally, the PPO total loss is

$$L_{\text{total}}(\theta) = \mathbb{E}_t \left[-L_t^{\text{clip}}(\theta) + c_1 L_t^{\text{VF}}(\theta) \right], \tag{4.15}$$

which includes the value function loss $L_t^{\text{VF}}(\theta)$ and its coefficient c_1 [37]. The total loss represents the overall objective that the training process seeks to minimize.

The method proposed for the intra-slice scheduler utilizes MARL in which there is one RL agent for each slice υ totaling Υ intra-slice agents. The MARL is formulated using a partially

observable Markov decision process (POMDP) defined by a tuple [67]

$$(\Upsilon, S, \{\boldsymbol{A}_{\upsilon}^{\mathsf{intra}}\}, \{\boldsymbol{O}_{\upsilon}^{\mathsf{intra}}\}, \{\mathsf{RW}_{\upsilon}^{\mathsf{intra}}\}, T, O), \tag{4.16}$$

where Υ represents the total number of slices in the system, which is equal to the number of intra-slice agents, $\{\boldsymbol{A}_{\upsilon}^{\mathsf{intra}}\}$ is a set of actions for each agent from slice υ, $\{\boldsymbol{O}_{\upsilon}^{\mathsf{intra}}\}$ is a set of observations for each agent from slice υ, $\{\mathsf{RW}_{\upsilon}^{\mathsf{intra}}\}$ is a set of reward signals for each agent from slice υ, T and O are the joint transition and observation models. The parameter-sharing approach is used since the intra-slice scheduler agents are homogeneous, allowing them to share the parameters of a single policy [67]. This allows the policy to obtain the experiences of all agents simultaneously, but it is still different agents since they receive different observations. The PPO clipped surrogate objective when using shared parameters to the intra-slice scheduler is

$$L_t^{\mathsf{clip}}(\theta) = \frac{1}{\Upsilon}\sum_{\upsilon=1}^{\Upsilon}\hat{\mathbb{E}}_t\Big[\hat{A}_t^{\upsilon}\min(r_t(\theta), \mathsf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon))\Big]. \tag{4.17}$$

A network scenario is defined as a specific combination of active slices, slice types, number of UEs assigned for each slice, UE characteristics, and the different UEs channel trajectories. Fig. 4.3 illustrates all the possible combinations of network scenario definition. The variation in the number of UEs per slice was omitted to simplify the visualization. Related works [12–19, 40] usually consider a unique network scenario to train and test the designed methods. Considering these training/testing conditions, these methods can deal with any channel episode if they keep the same network scenario characteristics intended for. Due to the high diversity of slice types (applications), training and testing the RRS methods under different network scenarios is essential to evaluate their ability to deal with various applications and fulfill slice intents.

When using an RL method to deal with different network scenarios, the observation space, the action space, and the reward function of the RL agent need to be able to deal with this variation since the number of entries in the neural network is fixed [68]. Related works [12–19, 40] usually consider a set of different variables per slice type, making it impossible to use these methods for different combinations of network scenarios, since a different number of active slices and slice types would lead to a variable number of entries in the RL agent, requiring a change in the number of entries in the inputs of the neural network or the reward calculation.

When based on state-of-art RL techniques such as [37, 39], using the same RL RRS method for different network scenarios requires a homogeneous input per slice type where each

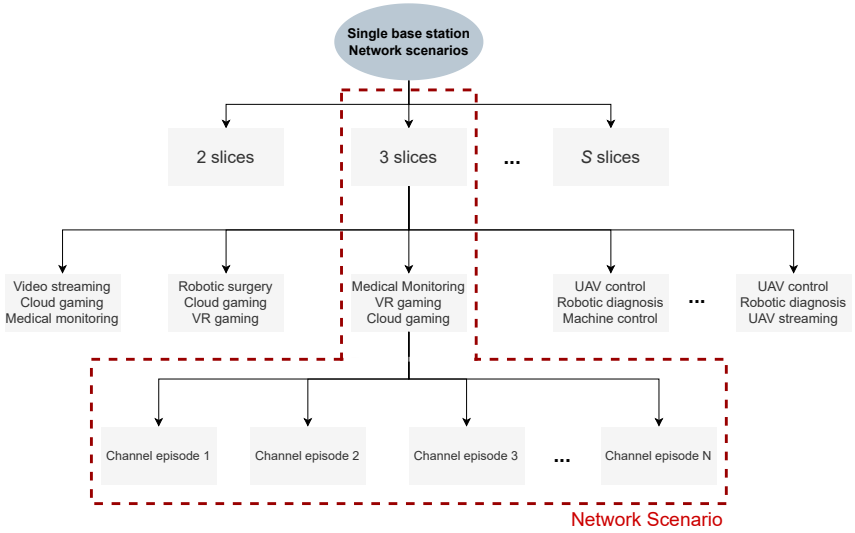

Figure 4.3: Single base station network scenario possibilities considering a different number of active slices, slice types, and UE channel trajectories. A network scenario is a specific combination of slices, slice types, and different channel conditions.

slice type should be represented by the same set of variables, keeping the same position in the entries of the neural network. Therefore, the same neural network structures from RL agents could be used for different network scenarios. Moreover, the contribution of the slice to the reward function must be calculated similarly to enable the RL agent to understand the different goals needed for each network scenario without changing the RL structure. The proposed method obeys these characteristics, enabling its use in various network scenarios.

4.3.2 Intent-drift calculation

An intent drift occurs when a system initially meets the defined intent but gradually, over time, allows its behavior to change or be affected until it no longer does or does so in a less effective manner [34]. Related work [40] represents the intent drift as a value between -1 and 0 with 0 representing the fulfilled intention and -1 representing the worst performance (most considerable distance from the current metric value and the requirement). These intent drift values show how distant the RL agent policy is from fulfilling the requirements. However, it also lacks information on performance degradation when intents are still fulfilled. For example, a slice with an effective throughput intent of $e_v^{\mathsf{req}} = 10\ \mathrm{Mbps}$ might receive $11\,\mathrm{Mbps}$ in a given moment. However, changes in network conditions can slightly decrease effective throughput, leading to an unfulfilled intent in the future. The representation of intent drift in [40] does not

account for this performance degradation that could give the RL agent important information about avoiding future intent violations.

This work represents the intent drift as a distance to fulfill the intent requirements (similarly as [40]) and as a representation of degrading metrics even if the slice intents are still fulfilled. In addition to the distance to fulfill the intent requirements, it also accounts for the distance between the requirement and an overfulfillment state, represented as a percentage above the requirement. Fig 4.4 shows an example of the intent drift values for an effective throughput intent with a requirement of $100\,\mathrm{Mbps}$ and an overfulfillment rate of 10%. Every time the effective throughput is under the specified requirement of $100\,\mathrm{Mbps}$, the intent drift accounts for a value between -1 and 0. If effective throughput is a value between $100\,\mathrm{Mbps}$ and 110 (over-fulfilled throughput), a value between 0 and 1 is taken into account. In case the effective throughput is greater than $110\,\mathrm{Mbps}$, the intent drift is 1. In step n, the intent drift is 1 since it is receiving an effective throughput equal to or greater than $110\,\mathrm{Mbps}$. However, in step $n+1$, the effective throughput decreased in value but still met the intent requirements of $100\,\mathrm{Mbps}$. The intent drift can provide information about performance degradation even in fulfilled intents so the RL agent can avoid unsafe fulfillment zones that can lead to unfulfilled intents in the future.

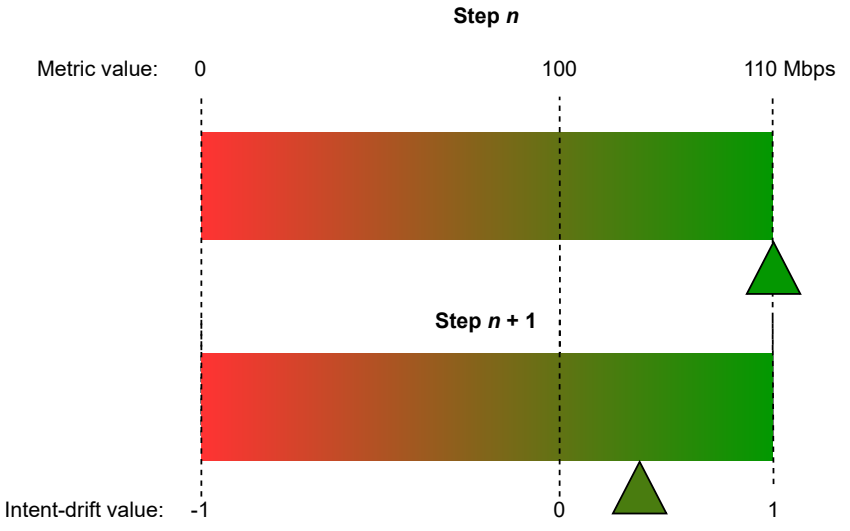

Figure 4.4: Intent-drift example for an effective throughput intent with a requirement of 100 Mbps and an overfulfillment rate of 10%. In step n, the served throughput is equal to or greater than the requested intent requirement. In step $n+1$ the served throughput decreases but the intent still fulfilled.

The intent drift is calculated for three intent requirements: effective throughput, buffer

latency, and packet loss rate. The intent drift for effective throughput i_u^e in a simulation step n for UE u is

$$i_u^e(n) = \begin{cases} \frac{e_u(n) - e_u^{\mathsf{req}}}{e_u^{\mathsf{req}} q_u^e(n)}, \text{ if } e_u(n) < e_u^{\mathsf{req}}(1 + q_u^e(n)) \\ \qquad\qquad \text{and } b_u^{\mathsf{occ}}(n) > 0 \\ 1, \qquad\quad \text{Otherwise} \end{cases}, \tag{4.18}$$

where the over-fulfilled requirement indicator $q_u^e(n)$ is

$$q_u^e(n) = \begin{cases} \zeta, \text{ if } e_u(n) \geq e_u^{\mathsf{req}} \\ 1, \text{ Otherwise} \end{cases}. \tag{4.19}$$

The overfulfillment rate ζ is a constant between 0 and 1, representing the maximum over-fulfillment value considered for network metrics. Therefore, there are three different cases to be covered. The first represents a scenario where the effective throughput requirement is not met. The $q_u^e(n)$ assumes the value one and $i_u^e(n)$ becomes a negative value between -1 and 0 representing the distance to fulfill the requirement. The second case occurs when the intent requirement is satisfied and below the over-fulfilled value $e_u^{\mathsf{req}}(1 + q_u^e(n))$, then $q_u^e(n)$ assumes the value ζ, and $i_u^e(n)$ computes a positive value between 0 and 1. The last case occurs when the effective throughput exceeds the overfulfillment value or the buffer of the UE u is empty (indicating that the throughput requirement could not be fulfilled since there is not enough data available in the buffer to be sent).

The intent drift for buffer latency i_u^ℓ in a simulation step n for UE u is calculated as

$$i_u^\ell(n) = \begin{cases} \frac{\ell_u^{\mathsf{req}} - \ell_u(n)}{l_u^{\mathsf{max}} - \ell_u^{\mathsf{req}} - q_u^\ell(n)}, \text{ if } \ell_u(n) > \ell_u^{\mathsf{req}}(1 - q_u^\ell(n)) \\ 1, \qquad\quad \text{Otherwise} \end{cases}, \tag{4.20}$$

where the over-fulfilled requirement indicator $q_u^\ell(n)$ is

$$q_u^\ell(n) = \begin{cases} l_u^{\mathsf{max}} - \ell_u^{\mathsf{req}}(1 + \zeta), \text{ if } \ell_u(n) \leq \ell_u^{\mathsf{req}} \\ 0, \qquad\qquad \text{Otherwise} \end{cases}. \tag{4.21}$$

Equations 4.18 and 4.20 are similar since they represent distances to fulfill the intent requirement or a distance between the fulfillment and overfulfillment cases. However, the intent drift for buffer latency considers that the buffer latency needs to be smaller than the requirements rather than greater than the effective throughput.

Finally, the intent drift for packet loss rate i_u^p in a simulation step n for UE u is calculated similarly to the intent drift for the buffer latency since it also requires to have a packet loss rate below a given requirement:

$$i_u^p(n) = \begin{cases} \frac{p_u^{\mathsf{req}} - p_u(n)}{1 - p_u^{\mathsf{req}} - q_u^p(n)}, \text{ if } p_u(n) > p_u^{\mathsf{req}}(1 - q_u^p(n)) \\ 1, \qquad\qquad \text{Otherwise} \end{cases}, \tag{4.22}$$

where the over-fulfilled requirement indicator $q_u^p(n)$ is

$$q_u^p(n) = \begin{cases} 1 - p_u^{\mathsf{req}}(1+\zeta), \text{ if } p_u(n) \leq p_u^{\mathsf{req}} \\ 0, \qquad\qquad\qquad \text{Otherwise} \end{cases}. \tag{4.23}$$

The slice intent drift for the effective throughput i_υ^e, buffer latency i_υ^ℓ and packet loss rate i_υ^p are defined as the average of the intent drift of the UEs assigned to the slice υ

$$i_\upsilon^x(n) = \frac{\sum_{u=1}^{U_\upsilon} i_u^x(n)}{U_\upsilon}, \text{for } x = e, \ell \text{ or } p. \tag{4.24}$$

4.3.3 Observation space

4.3.3.1 Inter-slice scheduler

The assumption of dealing with different network scenarios becomes a challenge to the RL design since the observation space needs to represent different network slice-type combinations and their intents, as depicted in Fig. 4.3. The proposed MARL agent utilizes a fully connected neural network for each RL agent with a fixed input size, and to deal with different network scenarios, it represents each type of slice by the same set of variables. The interchangeability of the observation space is ensured by representing the different types of slices with the same number of variables. Therefore, the number of required inputs in the RL agent is always the same, even when changing the number of active slices or slice types in the network. It also considers the same metric position for each slice in the observation space, ensuring that each input of the RL agent is always connected to the same metric. Therefore, the meaning of each input is kept the same even when changing the network scenario.

The RL observation space is defined as

$$\boldsymbol{O}^{\mathsf{inter}} = [\boldsymbol{s}_1^{\mathsf{inter}}, \boldsymbol{s}_2^{\mathsf{inter}}, \ldots, \boldsymbol{s}_\Upsilon^{\mathsf{inter}}] \tag{4.25}$$

where each $\boldsymbol{s}_{\upsilon}^{\mathsf{inter}}$ represents the metrics for slice υ as a vector with common slice metrics represented by:

$$\boldsymbol{s}_{\upsilon}^{\mathsf{inter}} = [m_{\upsilon}^{e} i_{\upsilon}^{e}, m_{\upsilon}^{\ell} i_{\upsilon}^{\ell}, m_{\upsilon}^{p} i_{\upsilon}^{p}, m_{\upsilon}^{e}, m_{\upsilon}^{\ell}, m_{\upsilon}^{p}, p_{\upsilon}, \frac{e_{\upsilon}^{\mathsf{req}}}{e_{\max}^{\mathsf{req}}}, \frac{U_{\upsilon}}{U_{\max}}, \frac{\mathsf{SE}_{\upsilon}}{\mathsf{SE}_{\max}}], \tag{4.26}$$

with active intent indications $m_{\upsilon}^{e,\ \ell \text{ or } p}$ representing a binary value which indicates if the intent requirement is active for slice υ. For example, in case $m_{\upsilon}^{e} = 1$, $m_{\upsilon}^{\ell} = 0$, and $m_{\upsilon}^{p} = 1$, slice υ has intents for effective throughput and packet loss rate, but not buffer latency. The high-priority indication p_{υ} is a binary value that indicates if the slice has a high priority (value 1), usually associated with critical applications.

The number of variables in the observation space depends on the maximum number of slices Υ allowed in the system. The proposed method assumes a fixed maximum number of slices and, hence, a fixed number of entries in the observation space. So, it is possible to handle a variable number of active slices from 2 to Υ. It fills the vector $\boldsymbol{s}_{\upsilon}^{\mathsf{inter}}$ with zero values every time a slice s is not active. The intent drift values $i_{\upsilon}(n)$ give information about which slice intents are unfulfilled, fulfilled, and over-fulfilled to the RL agent, enabling a better distribution of the RBs. The active intent indicator $m_{\upsilon}(n)$ shows which intents are enabled for each slice if the slice does not consider all of them. The $i_{\upsilon}(n)$ is a normalized value between -1 and 1 independent of the magnitude values of the effective throughput, buffer latency, and packet loss rate, but it still has to provide these magnitude indications so the RL agent can differentiate slice types. Therefore, the normalized effective throughput requirement, the number of active UEs, and the average spectral efficiency value are included in the observation space.

Considering a fully connected neural network with multilayer perceptron used in the RL algorithms, each entry in the observation space has a group of parameters whose values are changed during the RL training according to the location of the entries in the observation space [68]. Therefore, if an RL agent is trained with an observation space $\boldsymbol{O}^{\mathsf{inter}} = [\boldsymbol{s}_{1}^{\mathsf{inter}}, \boldsymbol{s}_{2}^{\mathsf{inter}}, \boldsymbol{s}_{3}^{\mathsf{inter}}]$ with 3 slices, it may not be able to handle an observation space $\boldsymbol{O}^{\mathsf{inter}} = [\boldsymbol{s}_{3}^{\mathsf{inter}}, \boldsymbol{s}_{2}^{\mathsf{inter}}, \boldsymbol{s}_{1}^{\mathsf{inter}}]$ during the test phase even if the same group of slice information is fed to the neural network due to changes in the location of the entries, therefore representing a different state.

When dealing with multiple network scenarios in which the maximum number of slices and entries in the neural network is fixed, the entry of each slice $\boldsymbol{s}_{\upsilon}^{\mathsf{inter}}$ in the observation space $\boldsymbol{O}^{\mathsf{inter}}$ can be associated with different types of slices. Consequently, the RL agent needs to be trained not only in the group of slice types but also in a different combination of the same

group to increase the chances of performing well during the test phase. Taking into account the maximum number Υ_{type} of types of slices and the maximum number of slices Υ in the network, the total number of combinations is $(\Upsilon_{\mathsf{type}}+1)^{\Upsilon}$. The slice entries $\boldsymbol{s}_{\upsilon}^{\mathsf{inter}}$ are ordered in the observation space $\boldsymbol{O}^{\mathsf{inter}}$ according to the requested throughput. Therefore, the total number of combinations becomes $\binom{\Upsilon+\Upsilon_{\mathsf{type}}-1}{\Upsilon}$, representing a reduction of $100 \times \left(1-\frac{\binom{\Upsilon+\Upsilon_{\mathsf{type}}-1}{\Upsilon}}{(\Upsilon_{\mathsf{type}}+1)^{\Upsilon}}\right)$. If considering, for example, a maximum number of slices $\Upsilon = 5$ and slice types $\Upsilon_{\mathsf{type}} = 10$, the reduction using ordered entries is 98.757%.

4.3.3.2 Intra-slice scheduler

The observation space to the intra-slice scheduler is

$$\boldsymbol{O}_{\upsilon}^{\mathsf{intra}} = [m_{\upsilon}^{e} i_{\upsilon}^{e}, m_{\upsilon}^{\ell} i_{\upsilon}^{\ell}, m_{\upsilon}^{p} i_{\upsilon}^{p}, m_{\upsilon}^{e}, m_{\upsilon}^{\ell}, m_{\upsilon}^{p}, \frac{R_{\upsilon}(n)}{R}, \frac{e_{\upsilon}^{\mathsf{req}}}{e_{\max}^{\mathsf{req}}}, \frac{U_{\upsilon}}{U_{\max}}, \boldsymbol{b}_{\upsilon}^{\mathsf{occ}}(n), \frac{\mathbf{SE}_{\upsilon}}{\mathsf{SE}_{\max}}], \tag{4.27}$$

where $\boldsymbol{b}_{\upsilon}^{\mathsf{occ}}(n) = [b_1^{\mathsf{occ}}(n), b_2^{\mathsf{occ}}(n), \ldots, b_{U_{\upsilon}}^{\mathsf{occ}}(n)]$ and $\mathbf{SE}_{\upsilon} = [\mathsf{SE}_1, \mathsf{SE}_2, \ldots, \mathsf{SE}_{U_{\upsilon}}]$. The observation space includes the inter-slice scheduler decision on the number of allocated RBs to the slice $R_{\upsilon}(n)$. The intra-slice scheduler's performance depends on the inter-slice scheduler's decisions since the number of RBs to distribute among the slice's UEs is limited by the inter-slice scheduler. Therefore, using the inter-slice scheduler decisions in the intra-slice observation space is important so the intra-slice scheduler can compute the best action given the RBs constraints.

4.3.4 Action space

4.3.4.1 Inter-slice scheduler

The action space of the inter-slice scheduler $\boldsymbol{A}^{\mathsf{inter}}$ has one output per slice

$$\boldsymbol{A}^{\mathsf{inter}} = [a_1^{\mathsf{inter}}, a_2^{\mathsf{inter}}, \ldots, a_{\Upsilon}^{\mathsf{inter}}], \tag{4.28}$$

where $a_{\upsilon}^{\mathsf{inter}}$ represents an action factor for slice υ with value in a range $[-1, 1]$ to match the output of the Gaussian distribution for continuous actions used [39].

The proposed agent uses a mask for invalid actions [69] to avoid selecting invalid actions since the number of active slices in a given step n varies over time. Using the PPO RL method with a continuous action space, the output of the policy network is a probability distribution over the values of the action factor [37]. Taking into account the maximum number of slices Υ in the system, there are Υ mean and standard deviation values. Every time a slice υ is inactive,

its associated mean and standard deviation values are set to -1 and 0. Hence, its action factor $a_{\upsilon}^{\mathsf{inter}}$ will receive a value of -1, and as a consequence, the number of allocated RBs to the slice υ will be zero.

The number of allocated RBs is

$$\boldsymbol{R}(n) = \chi\left(\frac{(\boldsymbol{A}^{\mathsf{inter}}+1)R}{\sum_{\upsilon=1}^{\Upsilon} a_{\upsilon}^{\mathsf{inter}}+1}\right), \tag{4.29}$$

with χ representing a function that rounds fraction numbers to integers and checks if all RBGs were allocated. If the summation of RBGs for all slices is larger/smaller than the number of available RBGs R, it adds/removes one RBG for each slice starting from the slice with the highest number of assigned RBGs to the slice with the smallest number until the number of allocated RBGs is equal to R. In case $\sum_{\upsilon=1}^{\Upsilon} a_{\upsilon}^{\mathsf{inter}}+1=0$, the available RBGs R are equally distributed among the active slices.

4.3.4.2 Intra-slice scheduler

The action space of the intra-slice scheduler $\boldsymbol{A}_{\upsilon}^{\mathsf{intra}}$ for slice υ has a unique output

$$\boldsymbol{A}_{\upsilon}^{\mathsf{intra}} = [a_{\upsilon}^{\mathsf{intra}}], \tag{4.30}$$

where $a_{\upsilon}^{\mathsf{intra}}$ represents an action factor for slice υ with an integer value from 0 to 2, which is mapped for round-robin, proportional-fair or maximum throughput algorithms. Therefore, the agent proposed for the intra-slice scheduler is responsible for selecting a scheduler algorithm to allocate the RBs assigned by the inter-slice scheduler to their UEs.

4.3.5 Reward function

4.3.5.1 Intra-slice scheduler

The reward calculation to the intra-slice scheduler from slice υ is

$$\mathsf{RW}_{\upsilon}^{\mathsf{intra}}(n) = \min(i_{\upsilon}^{e}(n), i_{\upsilon}^{\ell}(n), i_{\upsilon}^{p}(n)). \tag{4.31}$$

The proposed intra-slice scheduler minimizes the distance to fulfill its intents in each step n when one of the slice intents is not fulfilled. If all the slice intents are fulfilled, the RL maximizes the intent-drift values. The intra-slice scheduler only has information about the associated target slice υ. Therefore, it tries to maximize its reward value independently of other slice statuses.

4.3.5.2 Inter-slice scheduler

The main objective of the RL agent to the inter-slice scheduler is to avoid/minimize slice intent violations by fulfilling the slice requirements defined in Equation 4.12. Keeping the intent drift values $i_v^e(n)$, $i_v^\ell(n)$ and $i_v^p(n)$ between 0 and 1. In opposition to the intra-slice scheduler (Equation 4.31), the inter-slice scheduler has a global view of all slices and coordinates the RBGs allocation to avoid/minimize intent violations. The inter-slice reward $\mathsf{RW}^{\mathsf{inter}}(n)$ function is

$$\mathsf{RW}^{\mathsf{inter}}(n) = \begin{cases} \frac{\sum_{s\in\Upsilon_{\mathsf{act}}} \mathsf{RW}_v^{\mathsf{intra}}(n)}{\sum_{s\in\Upsilon_{\mathsf{act}}} 1}, & \text{if } \mathsf{cv}_{\mathsf{act}} = 0 \\ \frac{\sum_{s\in\Upsilon_{\mathsf{hpu}}} \mathsf{RW}_v^{\mathsf{intra}}(n)}{\sum_{s\in\Upsilon_{\mathsf{hpu}}} 1} - 1, & \text{if } \mathsf{cv}_{\mathsf{hp}} < 0 \\ \frac{\sum_{s\in\Upsilon_{\mathsf{actu}}} \mathsf{RW}_v^{\mathsf{intra}}(n)}{\sum_{s\in\Upsilon_{\mathsf{actu}}} 1}, & \text{Otherwise} \end{cases}, \tag{4.32}$$

where $\mathsf{cv}_{\mathsf{gr}} = \sum_{s\in\Upsilon_{\mathsf{gr}}} \min(\min(i_v^e, i_v^\ell, i_v^p), 0)$ with gr representing the active act or high-priority hp slice group. The Υ_{actu} and Υ_{hpu} represent the active and high-priority slices with unfulfilled intents. The high-priority slices receive an indication $p_v = 1$ in the observation space as explained in 4.3.3.1.

When all network slice intents are met, the reward function considers the average of all active slices, resulting in a positive value between 0 and 1. Suppose that there are one or more high-priority slices with unfulfilled intents. In that case, the reward assumes the average reward value of the unfulfilled high-priority slices minus one, resulting in a negative value between -1 and -2. If there are no high-priority slice violations, the reward accounts for the average reward among the slices with unfulfilled intents (it does not include high-priority slice intents), obtaining a value between 0 and -1. Every time a high-priority slice intent is not fulfilled, the proposed reward calculation accounts for only the high-priority intent values. Therefore, the proposed agent learns to fulfill the high-priority slices first and only after trying to reduce the distance to meet the requirements of the regular slices.

4.3.6 Baselines

Two RRS baselines for RAN slicing using RL from [12, 40] were adapted. It is important to emphasize that the related works contain different simulated/emulated scenario assumptions. Therefore, this work implemented the reward calculation from these baselines [12, 14] RRS for comparison with the proposed solution. Moreover, it also implements an adaptation of the PF and RR algorithms [22] using multi-agent for RAN slicing that considers each slice as a UE.

4.3.6.1 Multi-agent round-robin

The multi-agent round-robin (MARR) allocates the same number of RBGs to all the slices in the inter-slice scheduler. Each intra-slice scheduler equally distributes their available RBGs among the slice UEs.

4.3.6.2 Multi-agent proportional-fair

The multi-agent proportional fair (MAPF) balances maximizing the total network and provides all slices with minimal service. In the inter-slice scheduler, the PF action is

$$\boldsymbol{A}_{\mathsf{mapf}}^{\mathsf{inter}} = [a_1^{\mathsf{inter}}, a_2^{\mathsf{inter}}, \ldots, a_{\Upsilon}^{\mathsf{inter}}], \tag{4.33}$$

where the action factor $a_{\upsilon}^{\mathsf{inter}}$ is

$$a_{\upsilon}^{\mathsf{inter}} = \frac{b_{\upsilon}^{\mathsf{occ}}(n) b^{\mathsf{max}}}{\overline{e_{\upsilon}(n)}} \tag{4.34}$$

and $\overline{e_{\upsilon}(n)}$ represents the average effective throughput obtained by UEs in the slice υ. Finally, the action factors are mapped to the number of RBGs using Equation 4.29. The intra-slice schedulers use the same process to allocate the RBGs to the slice UEs but consider the UE metrics instead of the slice metrics.

4.3.6.3 Intent-aware RRS

Utilize an adaptation from [40] (presented in Chapter 3) that was originally designed to deal with eMBB, URLLC and BE slices with pre-specified intents. Since it considers a varying number of slices and intents in different network scenarios, the observation space, action space, and reward calculation were adapted to support until Υ slices in the system, utilizing intents for effective throughput $e_{\upsilon}^{\mathsf{req}}$, buffer latency $\ell_{\upsilon}^{\mathsf{req}}$ and packet loss rate $p_{\upsilon}^{\mathsf{req}}$ instead of the pre-specified intents for eMBB, URLLC and BE. The observation space is

$$\boldsymbol{O}_{\mathsf{ia}}^{\mathsf{inter}} = [\boldsymbol{s}_1^{\mathsf{inter}}, \boldsymbol{s}_2^{\mathsf{inter}}, \ldots, \boldsymbol{s}_{\Upsilon}^{\mathsf{inter}}] \tag{4.35}$$

where each $\boldsymbol{s}_{\upsilon}^{\mathsf{inter}}$ represents the metrics for slice υ as a vector with common slice metrics represented by:

$$\boldsymbol{s}_{\upsilon}^{\mathsf{inter}} = [e_{\upsilon}^{\mathsf{req}}, \ell_{\upsilon}^{\mathsf{req}}, p_{\upsilon}^{\mathsf{req}}, \mathsf{SE}_{\upsilon}(n), r_{\upsilon}(n), e_{\upsilon}(n), b_{\upsilon}^{\mathsf{occ}}(n), \ell_{\upsilon}(n), p_{\upsilon}(n), \iota_{\upsilon}(n)]. \tag{4.36}$$

It utilizes the same action space as the proposed method described in Subsection 4.3.4.1. The reward function of the inter-slice scheduler is

$$\mathsf{RW}_{\mathsf{ia}}^{\mathsf{inter}}(n) = \frac{\sum_{s\in\Upsilon_{\mathsf{act}}}\sum_{x\in[e,\ell,p]} w_\upsilon \min(i_\upsilon^x(n),0)}{\sum_{s\in\Upsilon_{\mathsf{act}}} w_\upsilon}, \tag{4.37}$$

with w_υ being a weight that defines the importance of intents from slice υ concerning other slices. It defines $w_\upsilon = 2$ for high-priority slices and $w_\upsilon = 1$ for regular slices. These weights were manually assigned in [40], but it is unclear how to define them when considering more than one network scenario. The high-priority slice intent weights were defined as double the regular slice intent values.

4.3.6.4 Sched-slicing RRS

Utilize the adaptation from the original method [12] presented in [40], utilizing a PPO RL agent to perform inter-slice scheduling and RR algorithm for intra-slice scheduling. This method was designed to deal with eMBB and URLLC slices through the minimization and maximization of network metrics. Since this work considers varying slices and intents, each slice is classified as eMBB or URLLC to apply the specified method. Therefore, it considers slices with a buffer latency requirement smaller than $20\,\mathrm{ms}$ as URLLC and slices with throughput requirements bigger than $20\,\mathrm{Mbps}$ as eMBB. All slices that meet both conditions are classified as eMBB and URLLC.

The same observation (Subsection 4.3.3.1 and action space (Subsection 4.3.4.1) were utilized as the proposed RRS but using the reward function

$$\mathsf{RW}_{\mathsf{sched}}^{\mathsf{inter}}(n) = \sum_{\upsilon\in\Upsilon_{\mathsf{embb}}} (r_\upsilon(n)) - \sum_{\upsilon\in\Upsilon_{\mathsf{urllc}}} (b_\upsilon^{\mathsf{occ}}(n) b_\upsilon^{\mathsf{max}} \mathsf{PS}_\upsilon), \tag{4.38}$$

that maximizes the served throughput $r_\upsilon(n)$ for eMBB slices and minimizes the buffer occupancy $b_\upsilon^{\mathsf{occ}}(n)$ for URLLC.

4.4 Simulation results and analysis

The proposed MARL agent was implemented with shared parameters using Ray Rllib [70] and RL baselines using the Stable Baselines3 library [59]. The RRS simulation was implemented using Python [71] and the simulation of the wireless channel using the QuaDRiGa simulator [62]. Table 4.1 shows the default hyperparameter values used for the proposed MARL method and RL baselines using PPO RL method.

Table 4.1: PPO RL hyperparameter values.

Hyperparameter	Value
SGD minibatch size	64
Learning rate	$3 \cdot 10^{-4}$
Batch size	2048
Gamma	0.99
Number SDG iterations	10
Lambda	0.95
Clip parameter ϵ	0.2
Entropy coefficient	0.01
Value function loss coefficient	0.5
Gradient clip	0.5
Network architecture	$[64, 64]$

4.4.1 Network scenario generation

A network scenario is defined as a combination of a specific number of active slices and slice types. The number of active slices for a network scenario Υ_{sce} is a random value between $\Upsilon_{\min} = 3$ and $\Upsilon = 5$. The network scenario generator randomly selects the slice indexes to use. For example, given a network scenario with $\Upsilon_{\mathsf{sce}} = 4$ active slices, the slice indexes used could be 1, 3, 4, and 5 while the slice index 2 is inactive. In addition, $\Upsilon^{\mathsf{hp}}_{\mathsf{sce}}$ represents the number of high-priority slices in the scenario. Each active slice has a unique slice type randomly selected from the options in Table 4.2. Each slice type has a high-priority indication and at least one associated intent for served throughput, latency, and reliability. The simulation parameters indicate the characteristics of the slice type UEs: buffer size, maximum buffer latency, message size, mobility, and requested traffic. The requested traffic for each UE of a specific slice type is a Poisson distribution with a mean equal to μ_{υ}. Finally, the number of UEs assigned for each slice type is randomly selected between the minimum and maximum number of UEs defined in Table 4.2.

The channel simulator QuaDRiGa [62] is used for channel episode generation for each network scenario (as depicted in Fig 4.3). It considers a single-input, single-output transmission

Table 4.2: Intents and simulation parameter characteristics for each slice type. The values were adapted from the indicated references.

Slice type	High-priority	Intents			Simulation parameters							
		Served throughput ϵ_v^{req}	Latency ℓ_v^{req}	Reliability ρ_v^{req}	UE's buffer size	UE's max buffer latency	Packet size	Mobility	Requested Traffic μ_v	Min. number UEs	Max. number UEs	Ref
Control case 2	Yes	-	50 ms	99.999999%	10240 packets	100 ms	8192 bits	0 Km/h	5 Mbps	4	5	[72]
Monitoring case 1	No	10 Mbps	-	-	10240 packets	100 ms	8192 bits	72 Km/h	10 Mbps	4	5	[72]
Robotic surgery case 1	Yes	20 Mbps	20 ms	99.9999%	1024000 packets	40 ms	16000 bits	0 Km/h	30 Mbps	4	5	[72]
Robotic diagnosis	No	15 Mbps	20 ms	99.999%	1024000 packets	40 ms	640 bits	0 Km/h	15 Mbps	4	5	[72]
Medical monitoring	No	10 Mbps	100 ms	99.9999%	10240 packets	200 ms	8000 bits	0 Km/h	10 Mbps	4	5	[72]
UAV app case 1	Yes	100 Mbps	200 ms	-	1024000 packets	400 ms	65536 bits	30 Km/h	100 Mbps	2	4	[73]
UAV control non-VLOS	Yes	20 Mbps	140 ms	99.99%	10240 packets	300 ms	65536 bits	30 Km/h	20 Mbps	4	5	[73]
VR gaming	No	100 Mbps	10 ms	99.99%	1024000 packets	20 ms	65536 bits	0 Km/h	100 Mbps	2	4	[72]
Cloud gaming	No	50 Mbps	80 ms	-	10240 packets	160 ms	65536 bits	0 Km/h	50 Mbps	2	5	[65]
Video streaming 4K	No	30 Mbps	-	-	10240 packets	100 ms	65536 bits	0 Km/h	30 Mbps	2	5	[64]

Table 4.3: Network and channel generation parameters used in the simulation.

Parameters	Range
Carrier frequency (f_{c})	$2.6\,\text{GHz}$
Bandwidth (B)	$100\,\text{MHz}$
Transmission power	$100\,\text{Watts}$
Window interval (w)	10
RBs available (G)	135
RBGs available (R)	27
3GPP scenario	38.901 Urban Macro-cell
Max. # of slices (Υ)	5
Max. # of UEs (U)	25
Overfulfillment rate (ζ)	0.1

system with a unique omnidirectional antenna to the base station and UEs. A channel realization is obtained in every $T_{\mathsf{s}} = 1\,\text{ms}$, which is the same value considered to the TTI, with simulation episodes that last $T_e = 1\,\text{s}$. The UE position is randomly defined in a range from 35 to $250\,\text{m}$ from the base station, moving in a random direction with speed defined according to the UE's slice type, but always respecting the minimum and maximum distance from the base station. The UEs can turn their direction with a probability $P_{\text{turn}} = 0.5$ in each $2\,\text{ms}$ or in case they reach the maximum distance from the base station to avoid off-limit movements. Table 4.3 shows the simulation parameters considered in the RRS system and the channel generation.

There are 200 randomly generated network scenarios. The first 10 network scenarios contain 100 different channel episodes each. The other 190 network scenarios have one channel episode each. Therefore, the simulation contains $10 * 100 + 190 = 1190$ RL episodes available

for training and testing. There are three simulation scenarios to evaluate the proposed method: Training for a single network scenario, generalizing for multiple network scenarios, and transfer learning for unseen network scenarios.

4.4.2 Training for a single network scenario

The proposed RL agent and baselines are trained and tested in the same network scenario. It uses the first 10 network scenarios that each contains 100 different channels. Therefore, for each network scenario containing $ep_{\mathsf{max}} = 100$ episodes, the agents train over $ep_{\mathsf{train}} = 60$ and utilize $ep_{\mathsf{val}} = 20$ for validation and $ep_{\mathsf{test}} = 20$ for testing. In the training phase, it utilizes $ec = 10$ epochs, representing the number of times the RL agent trains throughout the training dataset. Each episode contains $n_{\mathsf{ep}} = 1000$ steps. Therefore, the training phase for the proposed agent and the baselines contains $n_{\mathrm{train}} = ep_{\mathsf{train}} n_{\mathsf{ep}} ec = 60 \cdot 1000 \cdot 10 = 600000$ steps.

For each ten trained episodes, the agent is validated over the $ep_{\mathsf{val}} = 20$ episodes to evaluate the agent's capacity to generalize to different channel episodes. Each episode differs in only the UEs channel trajectories in the same network scenario. The agent parameters were selected from the best validation iteration since the agent needs to provide a good generalization capacity for different channel episodes. This simulation scenario assesses the capacity of RRS methods to be utilized for different network scenarios when trained and tested specifically for each of them. In the related works [12, 40], the presented methods were designed for specific network scenarios. Still, here, the simulation considers ten different network scenarios to evaluate whether the same technique could be applied to other network scenarios without changing the employed method.

From the 10 different network scenarios, Fig. 4.5 shows the lowest, median, and highest demanding network scenarios based on the number of RBs needed to satisfy the requested traffic μ_v. In each step n, it accounts for the minimum, average, and maximum spectral efficiency in the slice UEs RBs and calculates how many RBs would be needed to reach the requested traffic considering these values. Fig. 4.5 shows the number of required RBs to satisfy the traffic requested in each step n from the first episode of the selected network scenario.

The lowest demanding network scenario is the number 2 that needs an average number of RBs about to 53 out of $R = 135$ available. The median demanding scenario is the number 1, which needs an average number of RBs about to 85 with maximum values that get near from 100 RBs. The highest demanding network scenario, scenario 3, requires an average number of

RBs about to 190, which surpasses the available number of RBs $R = 135$, indicating that there are not sufficient resources for all slices. Therefore, the RRS will need to prioritize the slices with higher priority. There is a low variation in the number of required RBs for scenarios 0 and 1 due to the low mobility in the selected slice types of these network scenarios. The RRS allocates RBGs, therefore RBs are allocated in groups of $\frac{G}{R} = 5$ RBs as defined in Table 4.3. Table 4.4 shows the slice types for each selected network scenario.

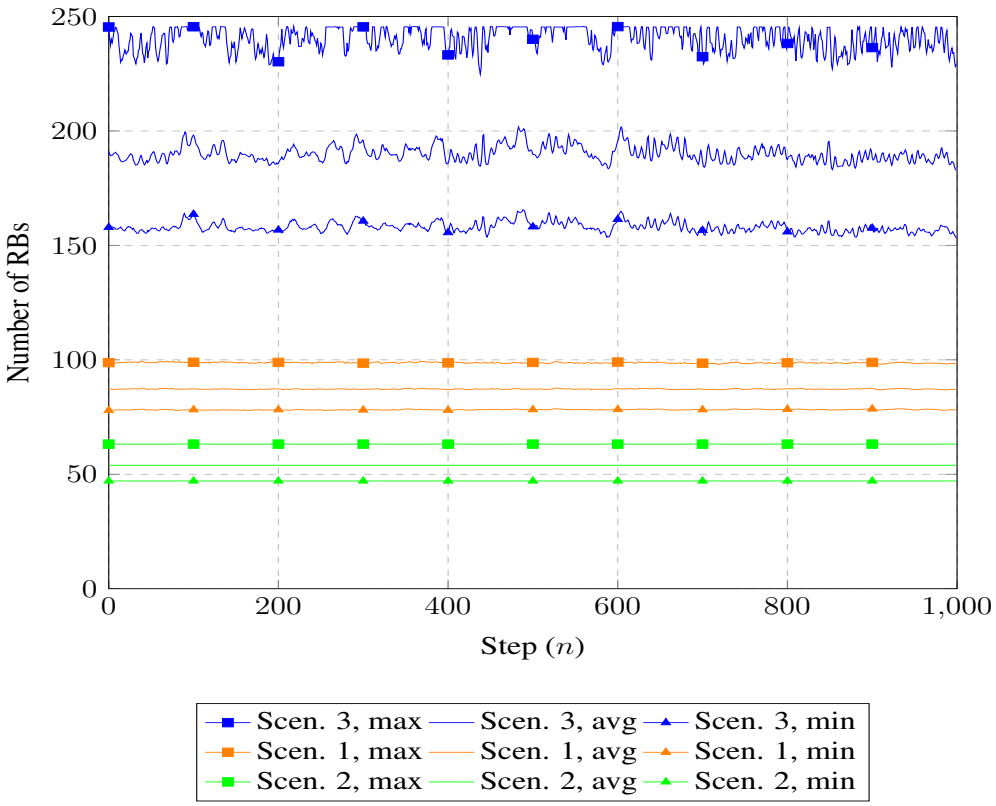

Figure 4.5: Lowest, median, and highest demanding network scenarios based on the number of RBs needed to satisfy the requested traffic μ_v.

Table 4.4: Slice types and number of UEs for the network scenarios 1, 2, and 3

Slice Index	Scenario 1 (21 UEs)	Scenario 2 (13 UEs)	Scenario 3 (23 UEs)
1	Robotic Diagnosis (4 UEs)	Robotic surgery case 1 (5 UEs)	Monitoring case 1 (5 UEs)
2	UAV control non-VLOS (5 UEs)	Medical monitoring (4 UEs)	UAV app case 1 (4 UEs)
3	Cloud gaming (5 UEs)	-	Robotic surgery case 1 (5 UEs)
4	Monitoring case 1 (5 UEs)	-	VR gaming (4 UEs)
5	VR gaming (2 UEs)	Cloud gaming (4 UEs)	Medical monitoring (5 UEs)

Fig 4.6 shows the inter-slice reward during training and validation to the highest demanding network scenario 3. The inter-slice reward is considered since it contains the contributions

of all slices in the network. The training accounts for the summation of inter-slice rewards $\mathsf{RW}^{\mathsf{inter}}$ in each episode, while the validation accounts for the average summation of reward values $\mathsf{RW}^{\mathsf{inter}}$ in the validation episodes in every 10 training episodes. The proposed method improves its ability to generalize to different channel episodes over time, as depicted in the validation performance. The training performance also improves over time but has a more unstable behavior concerning the evaluation value because their values are calculated in a single episode instead of an average in a group of episodes as made in the validation. Fig 4.7 shows the total loss to the inter-slice PPO RL agent. Similarly to the inter-slice rewards, the total loss also improves over training steps. The total loss still has variations over time since the proposed method trains with different channel episodes. Therefore, the policy parameters are adapted for each channel episode. The important aspect is finding a balance between the various channel episodes to reach a policy that can deal with all of them.

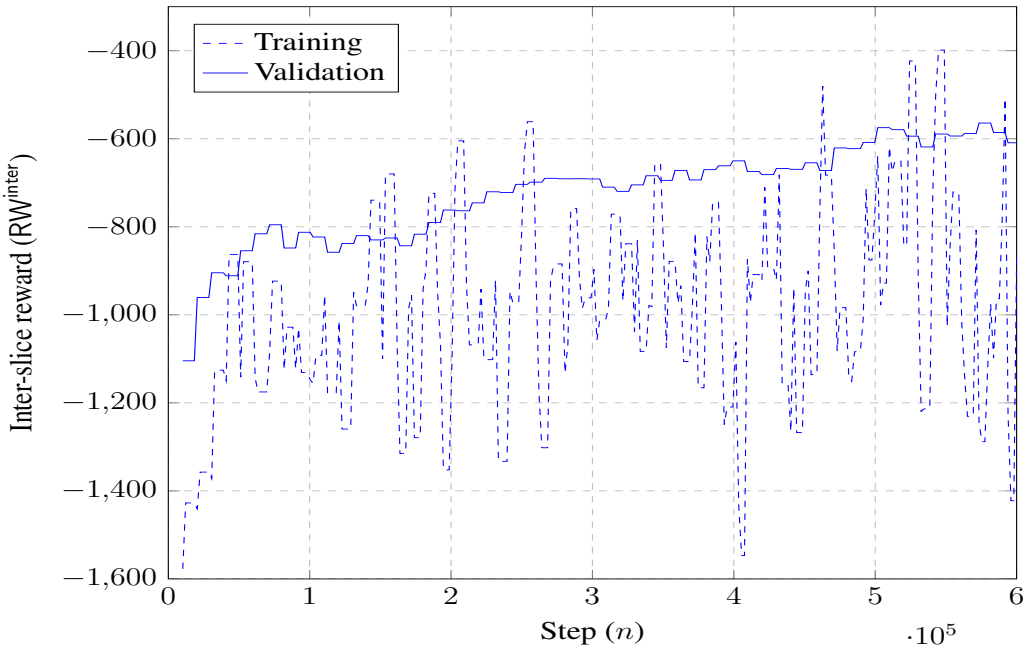

Figure 4.6: Inter-slice reward for training and validation during the $n_{\text{train}} = 600000$ training steps in the network scenario 3.

Fig. 4.8 shows the normalized distance to fulfill the slice intents of the network scenario and the normalized number of slice violations for each test episode considering the lowest, median, and highest demanding network scenarios. The number of active slices Υ_{sce} normalizes the results of the scenario when considering all active slices (total) and $\Upsilon^{\mathsf{hp}}_{\mathsf{sce}}$ when considering high-priority slices. This normalization facilitates the comparison between scenarios with dif-

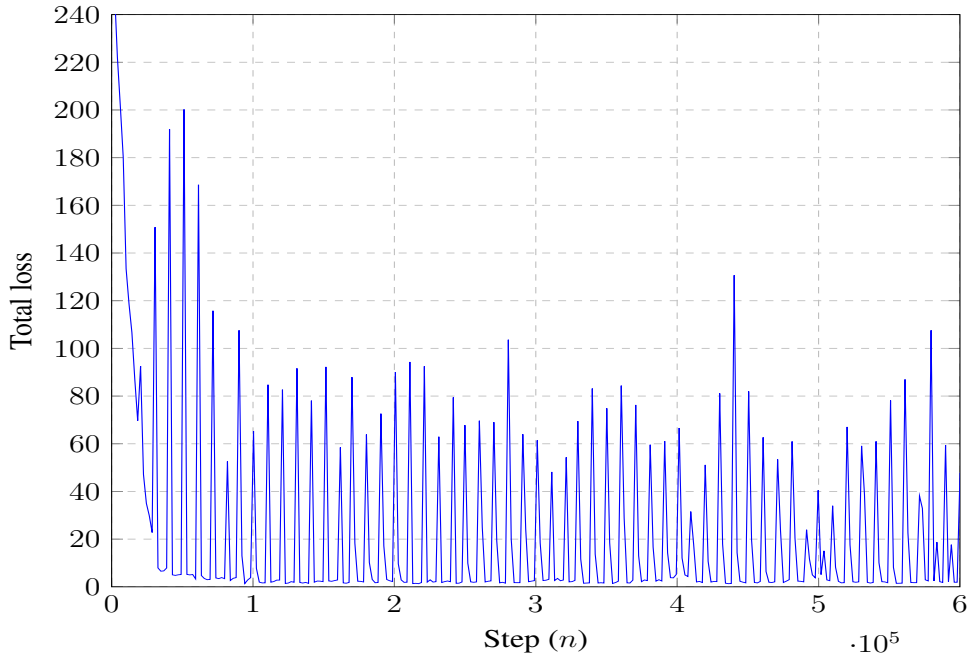

Figure 4.7: Inter-slice RRS total loss during the $n_{\text{train}} = 600000$ training steps in the network scenario 3.

ferent numbers of active slices. Each network scenario has results for the $ep_{\text{test}} = 20$ episodes tested. The first 20 episodes represent the result for the lowest demanding scenario, the median demanding scenario from episode 20 to 39, and the highest demanding scenario from episodes 40 to 59.

The normalized distance to fulfill is the inter-slice reward (Equation 4.32) but considering zero values when all the slices are fulfilled. Therefore, zero is the maximum value obtained in a simulation step n. The interpretation is how far the worst intent metric is from fulfilling its requirement. In the lowest-demand scenario, the proposed method and the baselines kept a zero distance, indicating the fulfillment of all slice intents. In the median scenario (episodes 20 to 39), the methods start to account for values different from zero, indicating that not all the intents are fulfilled at every step of the episodes. Still, the proposed method registers the smaller distance to fulfill the high-priority and total slices.

In the scenario with the highest demand (episodes 40 to 59), the number of available RBs $R = 135$ is insufficient to meet all the intent requirements. In this case, the RRS methods should first satisfy the high-priority slices and then the others. The proposed method presented more robust results when considering high-priority slice protection with a smaller cumulative distance to fulfill the requirements. Due to the high priority preference, the regular slices increased their

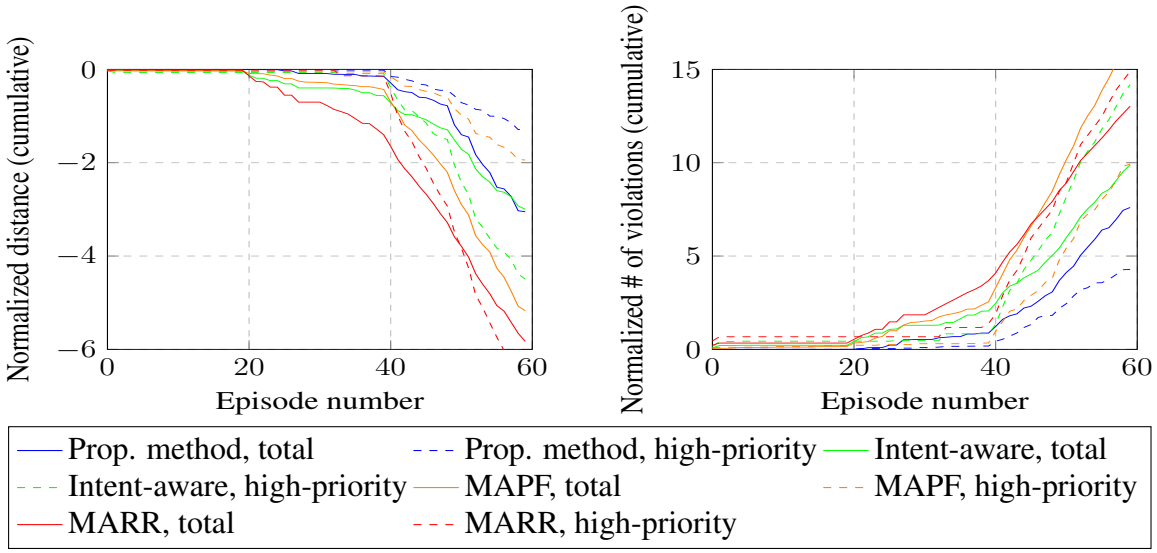

Figure 4.8: Normalized distance to fulfill intents and number of violations to the lowest, median, and highest demanding network scenarios. The proposed method and RL baselines train over $ep_{\mathsf{train}} = 60$ episodes and utilize $ep_{\mathsf{val}} = 20$ for validation and $ep_{\mathsf{test}} = 20$ in each network scenario.

distance, as the number of RBs available is insufficient to fulfill all intents. The boolean indication of high priority incorporated in the observation and reward calculation (Subsection 4.3.3.1 and 4.3.5.2) of the proposed method provides better performance in protecting high-priority slices even in different network scenarios compared to the weight-based method used in the Intent-aware RRS [40]. Still, the proposed method obtained the second-best performance when considering all slices with a performance close to the Intent-aware baseline.

Fig 4.8 also shows the normalized number of violations, where a slice violation occurs every time one or more slice intents are not fulfilled. Slice violation indicates a break in the SLA while the distance to achieve intents indicates how close the unfulfilled slices are to fulfilling their requirements when there is a slice violation. The proposed method obtained the best performance for high-priority slices and total slices. The distance to fulfill intents accounts for the distance of unfulfilled slices; still, the number of fulfilled slices is higher when using the proposed method while minimizing the high-priority slice violations. This explains why it obtained the best violation results concerning all slices, although it was the second-best in the normalized distance.

Fig. 4.9 shows the normalized distance to fulfill the slice intents and the normalized number of slice violations, but now concerning the ten different network scenarios. Each network scenario has $ep_{\mathsf{test}} = 20$ test episodes, totaling 200 episodes. Again, the proposed method

obtained the best performance for high-priority slices in the normalized distance to fulfill and the number of violations. In addition, it also obtained the best performance in the normalized distance and number of violations for all slices.

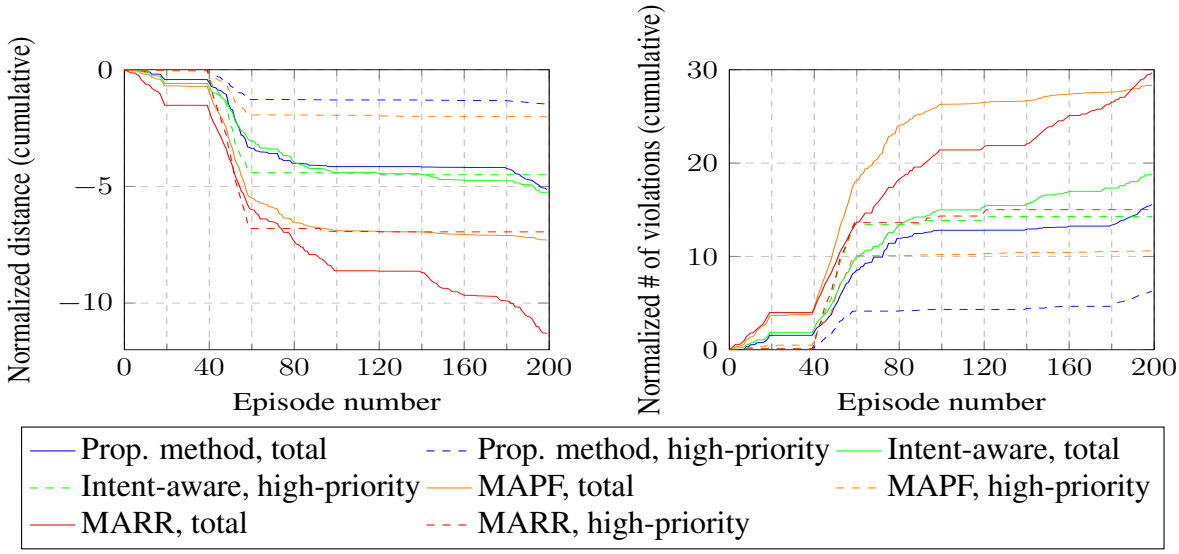

Figure 4.9: Normalized distance to fulfill intents and number of violations considering ten different network scenarios. The proposed method and RL baselines train over $ep_{\mathsf{train}} = 60$ episodes and utilize $ep_{\mathsf{val}} = 20$ for validation and $ep_{\mathsf{test}} = 20$ in each network scenario.

The Sched-slicing RRS baseline was omitted from the previous results due to its poor results in the tested network scenarios. The cumulative normalized number of violations obtained in the same simulation of Fig. 4.9 was -16 and -30 for the high-priority and total slices, which represents the highest number of violations compared to the other methods. In [40], the simulation results were limited to one network scenario with one eMBB, one URLLC, and one mMTC slices. The result of the Sched-slicing RRS baseline was worse than the proposed method due to its inability to deal with the network intents since it was designed to maximize and minimize metrics and not fulfill intents. The Sched-slicing RRS baseline was adapted to deal with different network scenarios, making this approach even more difficult. When considering an intent-based network, the RRS to be adopted must be specifically designed to deal with slice intents.

When trained for each network scenario, the proposed method performed best both in protecting high-priority and regular slices and minimizing the total number of violations in different network scenarios. It is suitable for future mobile networks because of its ability to deal with many network scenarios, simplifying the need for specific algorithms for each network

scenario. In addition, the intent-based approach enables the use of the proposed method in an intent-based network to deal with high-level intents and provide the intent manager in the RAN domain a capability of fulfilling local RAN objectives. Still, training a RL RRS from scratch for each network scenario can be time-consuming, and general performance could be improved by using previously learned experiences from other network scenarios. Therefore, alternatives to speed up the training of the proposed method are vital to reduce the time to deploy a new RRS policy.

4.4.3 Generalizing for multiple network scenarios

The proposed MARL agent and baselines are trained and tested in different network scenarios to evaluate their generalizability. It generates 200 different network scenarios where each network scenario contains unique UE trajectories totaling $ep_{\mathsf{max}} = 200$ episodes in the simulation. The RL agents train over $ep_{\mathsf{train}} = 180$ episodes and utilize $ep_{\mathsf{val}} = 10$ for validation and $ep_{\mathsf{test}} = 10$ for testing. In the training phase, it utilizes $ec = 5$ epochs. Each episode contains $n_{\mathsf{ep}} = 1000$ steps. Therefore, the training phase for the proposed agent and the baselines contains $n_{\mathrm{train}} = ep_{\mathsf{train}} n_{\mathsf{ep}} ec = 180 \cdot 1000 \cdot 5 = 900000$ steps.

For each ten trained episodes, the agent is validated over the $ep_{\mathsf{val}} = 10$ to evaluate the agent's capacity to generalize to different network scenarios. Therefore, each episode differs in both the UEs channel trajectories and the network scenario. The agent parameters utilized in the test phase are selected from the best validation iteration since it gives the agent the best performance to generalize to different network scenarios. This simulation scenario assesses the capacity of RRS methods to generalize to different and unseen network scenarios without retraining for each specific network scenario. Using an agent that does not require retraining is very convenient since there is no further action to deal with new/unseen network scenarios.

Fig 4.10 shows the inter-slice reward during training and validation. Unlike the behavior depicted in Fig 4.6, here the proposed method can hardly improve its capacity of generalizing to different network scenarios over time as shown in the validation performance, reaching its best performance in the first validation after $10\,\mathrm{k}$ trained steps. Training performance has a higher value variation that expresses instability when learning to deal with different network scenarios. Fig 4.11 shows the total loss to the inter-slice RL agent. The total loss still has high values even when the number of steps increases. The training process should contain $n_{\mathrm{train}} = 900000$ steps, but due to the instability in the training with high loss values, the simulation stops before the

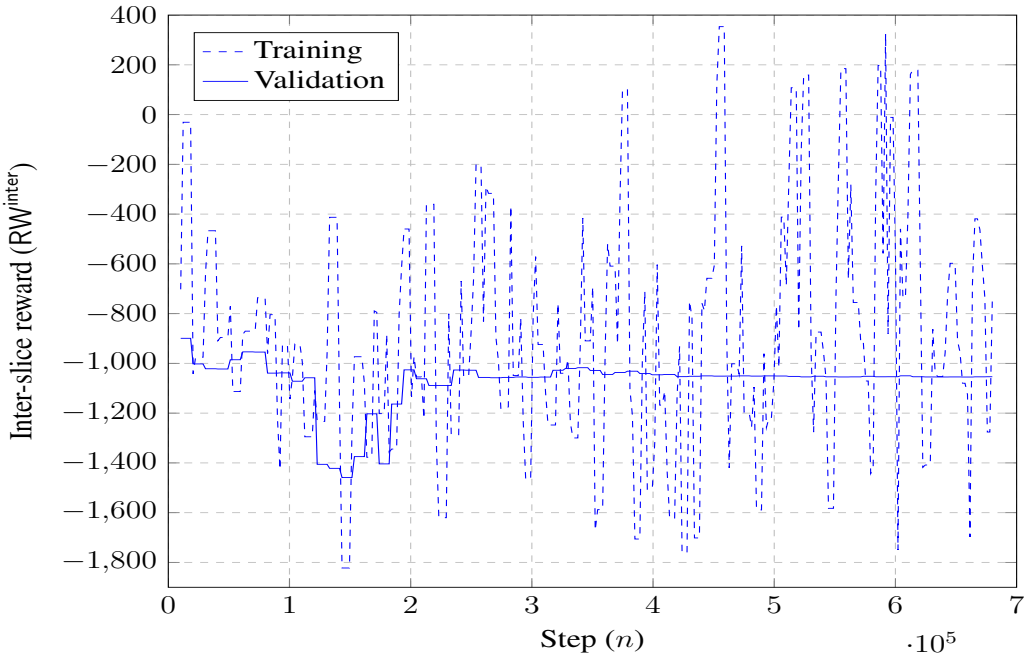

Figure 4.10: Inter-slice reward for training and validation during the $n_{\text{train}} = 900000$ training steps.

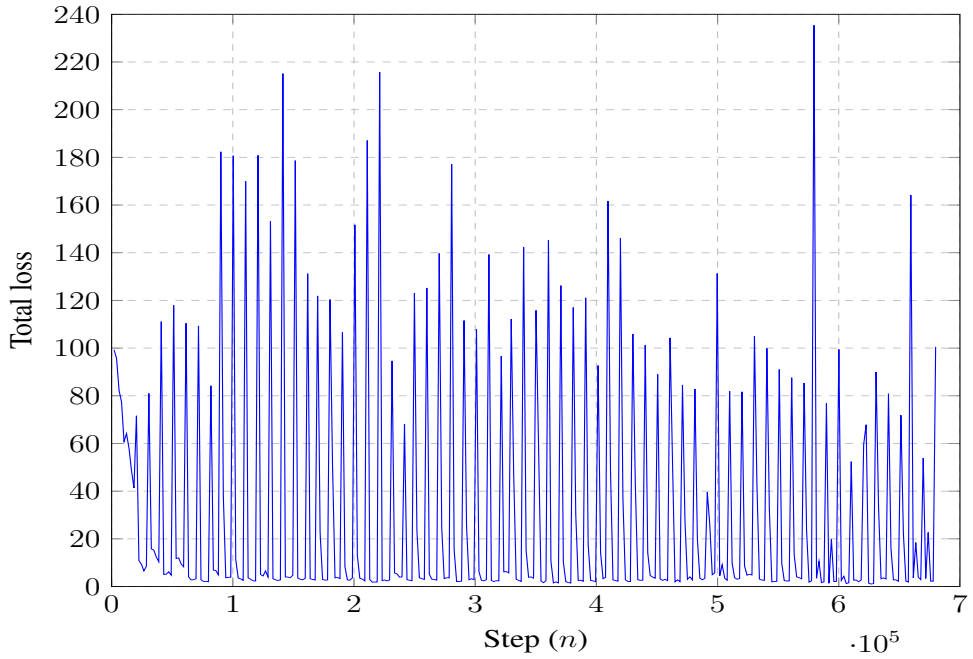

Figure 4.11: Inter-slice RRS total loss during the $n_{\text{train}} = 900000$ training steps.

total steps.

Fig. 4.12 shows the normalized distance to fulfill the slice intents and the normalized number of slice violations for $ep_{\text{test}} = 10$ test episodes considering ten different and unseen network scenarios. When evaluating the normalized distance, the proposed method obtained the worst performance for the high-priority slices and the second-worst performance when considering all slices. When considering the RL baselines, they performed poorly compared to the MAPF method. The normalized number of violations shows results similar to those of the proposed method, obtaining poor performance among the baselines.

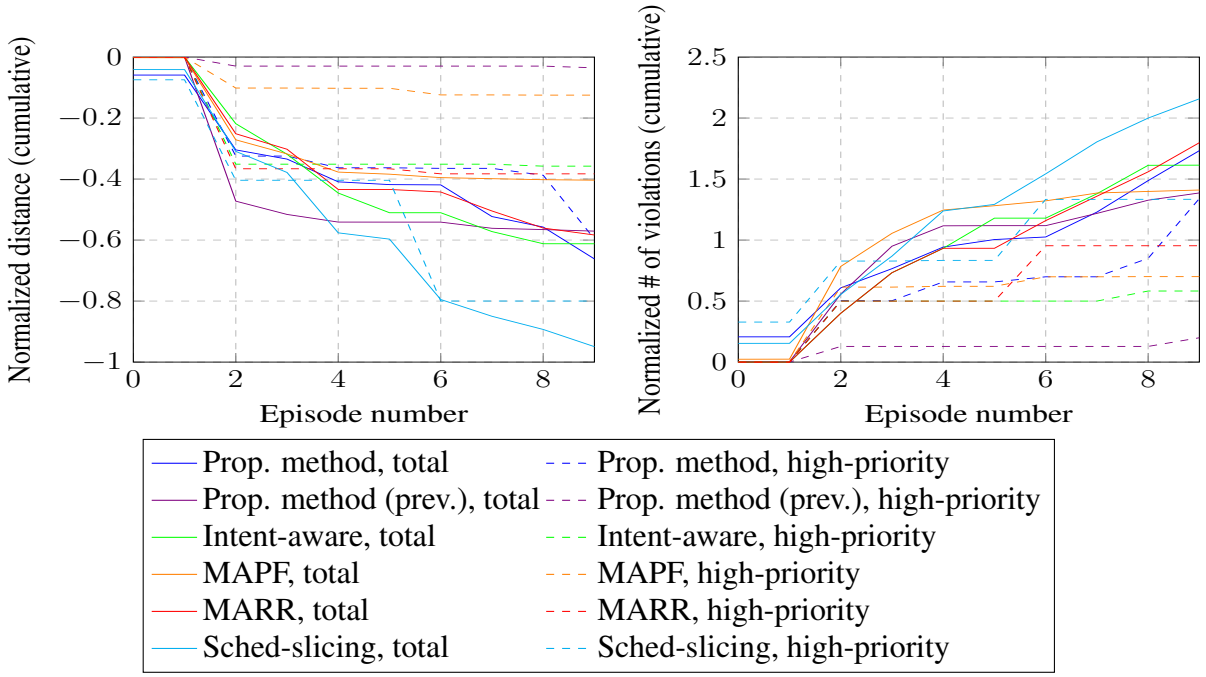

Figure 4.12: Normalized distance to fulfill intents and number of violations considering 160 different network scenarios in the training, 10 in the validation, and 10 in the test.

The same ten network scenarios for testing from the results generated were utilized in Figure 4.9. It is possible to compare the results of the proposed method with those trained for each specific scenario. The proposed method trained for each scenario, named "Prop. method (prev.)", shows the best performance among all the options, and not only the proposed method but all the baselines could not reach a similar performance level. These results show that the proposed method and RL baselines cannot generalize to unseen scenarios and perform poorly compared to agents trained for each network scenario.

Since the RL-based methods could not generalize to unseen network scenarios, another experiment is proposed in which the RL models are trained, evaluated, and tested in the same

episodes in a reduced dataset. The objective is to evaluate if the RL-based methods can overfit in the training dataset to deal with different seen network scenarios. It uses 10 different network scenarios totaling $ep_{\text{max}} = ep_{\text{train}} = ep_{\text{val}} = ep_{\text{test}} = 10$ episodes in the simulation. The same episodes used for training are also used for validation and testing. In the training phase, it utilizes $ec = 100$ epochs, totaling $n_{\text{train}} = ep_{\text{train}} n_{\text{ep}} ec = 10 \cdot 1000 \cdot 100 = 1000000$ training steps. The objective is to overfit the proposed method and RL baselines to evaluate whether dealing with multiple seen network scenarios is possible. The best agent weights are selected on the basis of the validation performance; in this case, the validation set is the same as the test set.

Fig 4.13 shows the inter-slice reward during training and validation. Using a smaller training set and the same set for validation and testing, the validation and training results were slightly better when compared to 4.10. However, the proposed method cannot achieve performance similar to that demonstrated in Fig. 4.6 when it trains the agent for each specific network scenario. The total loss depicted in Fig. 4.14 obtained high values even when the training steps were increased. Again, due to the training instability, it was not able to complete the defined $n_{\text{train}} = 1000000$ training steps.

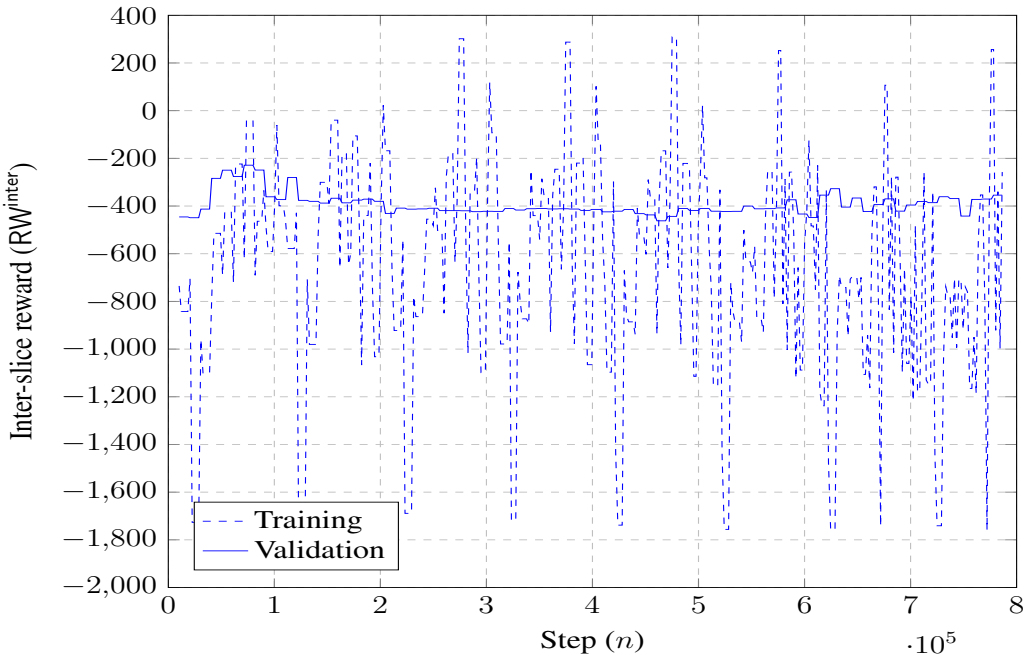

Figure 4.13: Inter-slice reward for training and validation during the $n_{\text{train}} = 1000000$ training steps.

Fig. 4.15 shows the normalized distance to fulfill the slice intents and the normalized number of slice violations for the $ep_{\text{test}} = 10$ test episodes, considering that all network scenar-

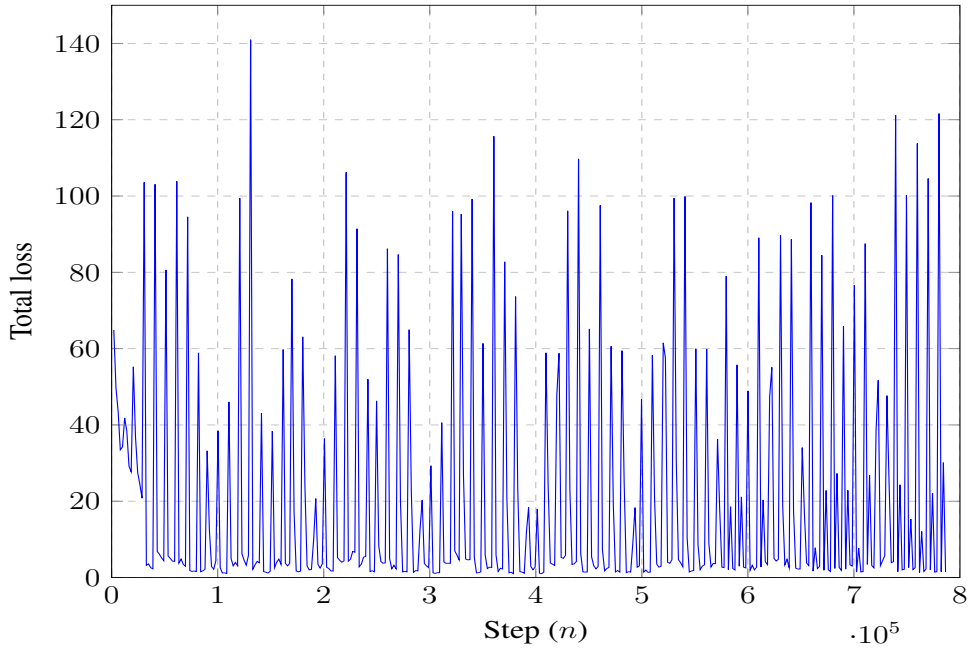

Figure 4.14: Inter-slice RRS total loss during the $n_{\text{train}} = 1000000$ training steps.

ios were seen during the training and validation phase. Even reducing the number of network scenarios from 200 to 10 and using the same dataset for training, validation, and testing, the proposed agent and RL baselines presented poor performance compared to the proposed agent trained for each specific network scenario. The policies for each network scenario are very different, which justifies the high variation in total loss since the MARL agent still receives large policy updates even after a considerable number of training steps. Therefore, the proposed agent cannot generalize to different network scenarios without retraining, indicating that the proposed method and baselines cannot overcome these challenges using a unique pre-trained agent to deal with all the possible network scenarios.

4.4.4 Using transfer learning for unseen network scenarios

The proposed method and the baselines cannot generalize to different unseen network scenarios and do not have the capacity to handle a reduced number of trained scenarios as demonstrated in the previous Subsection 4.4.3. Therefore, the proposed method must be trained specifically for each network scenario. Retraining the proposed MARL from scratch for each network scenario can take a significant amount of training steps, and the retraining frequency depends entirely on the network scenario variations faced during tests and actual deployments. Therefore, reducing the training time to achieve satisfactory performance with the proposed

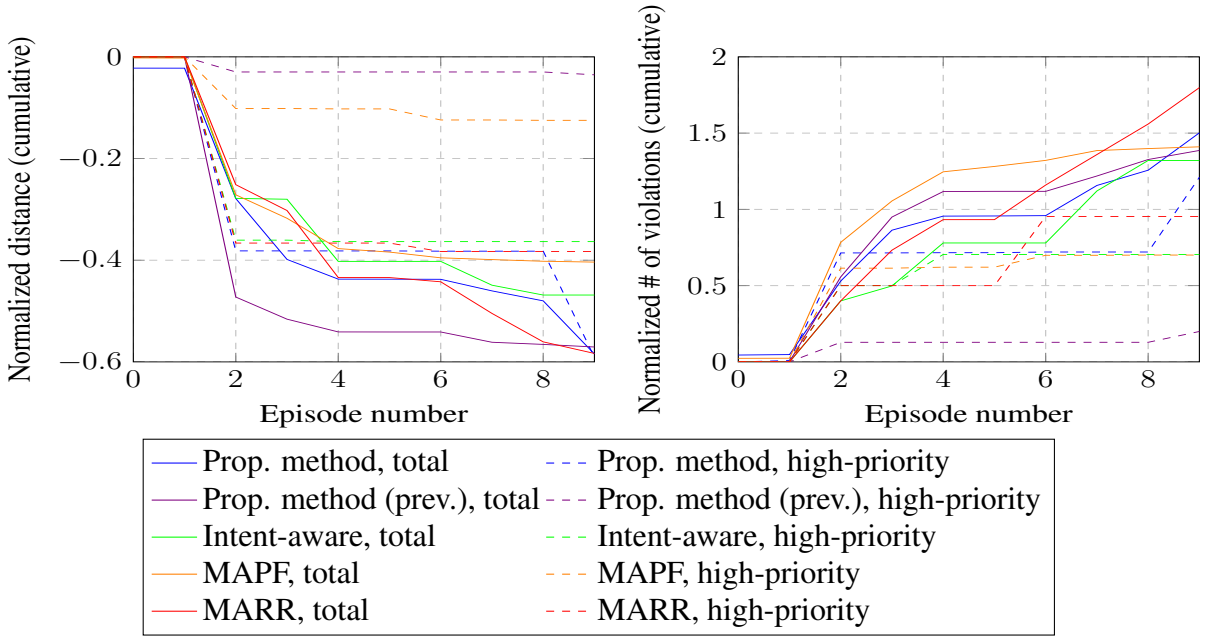

Figure 4.15: Normalized distance to fulfill intents and number of violations considering 10 different network scenarios in the training and the same network scenarios for validation and test.

agent and minimizing the deployment duration in realistic environments is essential.

Due to the homogeneous observation and action space described in Subsections 4.3.3.1 and 4.3.4.1, the proposed agent can use the same neural network structures of the MARL for different combinations of slice types and intents that characterize a network scenario. Using transfer learning was proposed to accelerate the training process in the requested new network scenarios and improve the performance of the proposed method. Transfer learning uses previously learned experiences while fine-tuning the RL agent on new scenarios [74]. It is usually more efficient than learning from scratch and requires less time to perform satisfactorily.

The first 10 network scenarios were used containing 100 different channel episodes each (the same as in Section 4.4.2). For each network scenario that contains $ep_{\mathsf{max}} = 100$ episodes, agents train in $ep_{\mathsf{train}} = 80$ and utilize the same $ep_{\mathsf{val}} = ep_{\mathsf{test}} = 20$ episodes for evaluation and testing. The same episodes are set for testing and evaluation to assess how many training steps agents can take to reach their best performance. In the training phase, it utilizes $ec = 10$ epochs, totaling $n_{\text{train}} = ep_{\mathsf{train}} n_{\mathsf{ep}} ec = 80 \cdot 1000 \cdot 10 = 800000$ trained steps. The trained RL model on Subsection 4.4.3 utilizing 200 network scenarios in the simulation is considered as a base model for fine-tuning whose parameters are used as initial parameters for the model to be fine-tuned.

Fig. 4.16 shows the average inter-slice scheduler reward (Equation 4.32) obtained in the evaluation over $ep_{\text{val}} = 20$ episodes for the network scenario 1. This compares the performance of the proposed method trained from scratch with the fine-tuned agent. The proposed fine-tuned agent obtained the best performance in the evaluation considering all the trained episodes, reaching its best performance around $389\,\text{k}$ trained steps. There is no practical method to define how many steps the proposed agent could take to converge to its best performance, and this number of required trained steps varies according to the evaluated network scenario. To reduce the required time to deploy the method, and since there is no general number of trained steps it can ensure the convergence of the proposed method, it considers a reduction of 8 times in the trained steps, totaling $100\,\text{k}$ trained steps for analysis.

When considering the best performance obtained by the proposed fine-tuned method and the proposed method trained from scratch in the first $100\,\text{k}$ trained steps, the proposed fine-tuned agent obtained an average reward of 246.5 with about $92\,\text{k}$ trained steps while the proposed method trained from scratch obtained an average reward value of 217.1 with $51\,\text{k}$ steps. The fine-tuned agent obtained its best average reward value (in all training episodes) of 270.4 with $389\,\text{k}$ trained steps. Therefore, the best average reward took near 4 times more trained steps to obtain an increase of only 8.8% in the average reward.

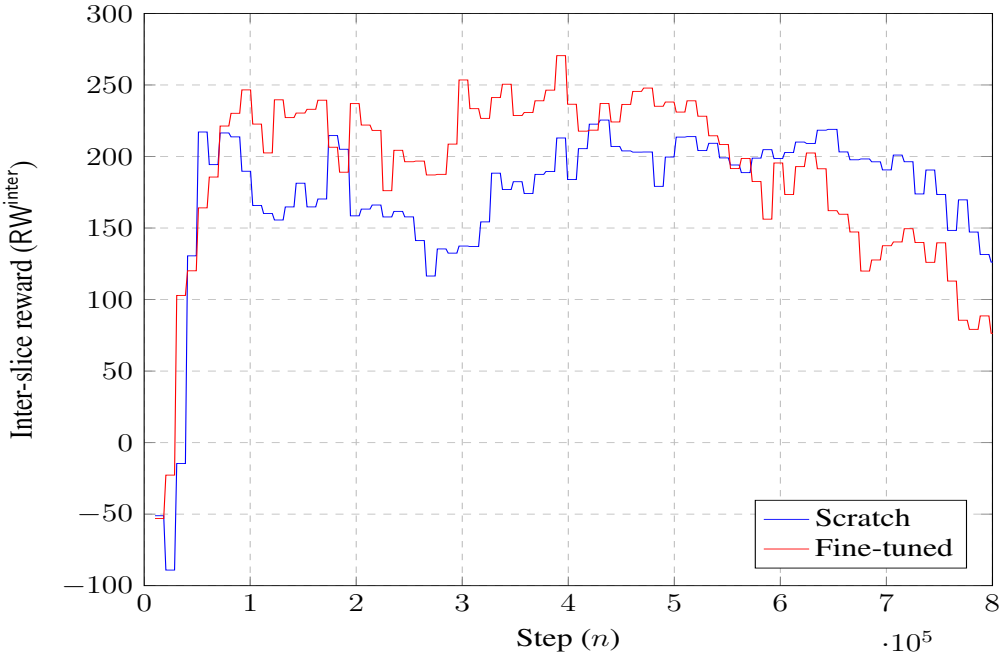

Figure 4.16: Inter-slice reward obtained in the evaluation over 20 episodes in the network scenario 1 considering a proposed agent trained from scratch with a fine-tuned agent.

Table 4.5: Comparison between the proposed method trained from scratch and the fine-tuned agent in ten network scenarios over the first 100 and all trained episodes.

Scenario index	First 100 episodes				All episodes					
	Scratch		Fine-tuned		Scratch			Fine-tuned		
	Avg. Reward	Steps	Avg. Reward	Steps	Avg. Reward	Steps	Improve. (%)	Avg. Reward	Steps	Improve. (%)
1	217.1	51k	246.5	92k	225.4	430k	3.7	270.4	389k	8.8
2	388.8	10k	389.5	30k	388.8	10k	0	389.9	296k	0.1
3	-813.3	92k	-693.7	92k	-638.1	727k	27.4	-648	409k	7
4	-12.6	40k	-10.1	71k	17.3	727k	173.1	6.1	747k	266.1
5	37.2	40k	11.6	51k	179.3	307k	79.2	187.15	358k	93.7
6	190.7	81k	198.7	92k	197	266k	3.1	198.7	92k	0
7	573	30k	572.1	61k	575.4	225k	0.4	572.6	163k	0
8	161.1	40k	159.7	40k	172.3	706k	6.5	161.23	194k	0.9
9	361.9	40k	369.8	30k	361.9	40k	0	369.8	30k	0
10	-1097.8	92k	-1037.1	92k	-14.2	634k	7594.6	-46.8	757k	2112.8

Evaluating the results in a unique network scenario is insufficient to assess the transfer learning capacity of reducing the required steps to obtain satisfactory performance and improve overall method performance. Therefore, Table 4.5 summarizes the results for the ten different network scenarios. It presents the best average inter-slice reward value and the number of trained steps to accomplish it when considering the first 100 episodes and all episodes. In the first 100 episodes, the fine-tuned agent obtained the best performance compared to the agent trained from scratch in 7 network scenarios. The unique significant difference in network scenarios that the fine-tuned agent obtained a smaller average reward occurs in the network scenario 5. However, in the network scenarios 7 and 8, the fine-tuned and scratch agents showed a slight difference in performance.

When comparing performance in all trained episodes, the fine-tuned and scratch methods obtained the best performance in each of the 5 network scenarios, and the average rewards obtained had similar values, indicating that both the agent trained from scratch and the fine-tuned agent can obtain good results when trained in a large number of steps. Table 4.5 also shows the percentage of improvement in the average inter-slice reward obtained when comparing the best result obtained in the first 100 episodes and all episodes for the fine-tuned and scratch agents. The fine-tuned agent obtained an improvement of less than 10% in 7 out of the 10 network scenarios. Indicating that in some network scenarios, training with a large number of steps may not lead to a substantial increase in the average reward obtained. However, in the network scenarios 4, 5 and 6, the percentage of improvement is higher than 90%, obtaining 2112% in the network

scenario 10.

The policy obtained in the generalization for multiple network scenarios (Subsection 4.4.3) represents a group of common neural network parameters trained to deal with different network scenarios. Although the poor performance presented in Fig. 4.12, it is possible to interpret that the obtained policy represents an average policy to handle different network scenarios. Therefore, for most network scenarios, the parameters provided by this trained policy are far from satisfactory performance. However, it is still closer to the desired policy than the method trained from scratch. This justifies the better performance obtained in the first 100 trained episodes. However, it does not lead to a faster trajectory to the best parameters as presented in the comparison of steps to obtain the scratch and fine-tuned best performances in all episodes.

The proposed fine-tuned method performs best in the first 100 episodes and can achieve the best or near optimum performance compared to an agent trained from scratch for all episodes. Therefore, to reduce the time required to implement the RRS for a new network scenario, the proposed method could be trained in 100 episodes and begin to use the agent in production. However, training in all episodes should still run in parallel, so it can substitute production RRS for the proposed method trained in all episodes when it finishes to guarantee the best performance in a higher number of network scenarios.

Chapter 5

Conclusion

This thesis proposed the use of an intent-based RRS using RL for RAN slicing scenarios. The RRS is divided into inter and intra-slice schedulers responsible for allocating the radio resources among slices and slice UEs, respectively. The investigation of the proposed intent-based RRS was split into two different problems: The first investigates the usage of an intent-based RRS in a RAN slicing scenario with a fixed number of slices containing eMBB, URLLC and BE slice types. There is variation in the UEs trajectories and intents over the simulations. The second problem investigates the proposed method's ability to deal with different network scenarios in which there is variation in the number of active slices, slice types, number of UEs, UEs trajectories, and device characteristics.

In the network scenario with a fixed number of slices, which is a typical scenario explored in the related work literature, the RRS using RL has an architecture specifically designed for the network scenario containing eMBB, URLLC, and BE slice types. The proposed intent-aware method utilizes the RL SAC technique in the inter-slice scheduler and adopts a round-robin technique in the intra-slice scheduler. The RRS orchestrated the radio resources among the slices to fulfill the slice intents defined in the SLA. The proposed SAC RL agent outperformed the baselines with a greater cumulative reward during the test episodes, fulfilling the slice requirements most of the time with respect to the baselines. It also prioritizes the most important slices' intents. Moreover, a limited observation space was presented using only slice information based on UEs' average metrics, which favors the algorithm concerning scaling with the number of UEs. The proposed limited observation space presented a similar performance to the full observation space, even using a small neural network to perform RRS operations.

The results demonstrated the importance of designing intent-based RRS methods to en-

able intent-based systems in the RAN domain. All baselines not intended for intent could not perform satisfactorily in the proposed scenarios. Some aspects, such as using RL in the intraslice scheduler, assigning intent weight values for different network scenarios, varying spectral efficiency values, and evaluating scenarios with a variable number of slices and UEs, are not approached in this problem.

The second problem investigated in this thesis is the proposed method's capacity to deal with multiple network scenarios. The proposed intent-based RRS used MARL for inter- and intra-slice scheduling in scenarios with RAN slicing. The RL agent used in the inter-slice scheduler allocates the available RBGs among the slices, while the intra-slice scheduler utilizes a MARL scheme with one RL agent per slice, which allocates the slice RBGs to the UEs. The proposed method improved the intent-drift reward from the previous problem investigation and removed the need for weight optimization for each network scenario. It implemented the spectral efficiency variation on the frequency and multiple network scenarios with different numbers of active slices, slice types, number of UEs, UEs trajectories, and device characteristics.

The proposed method outperformed the baselines in protecting the slices with higher priority and considering all the slices when training for each specific network scenario in ten different network scenarios. The results of training and testing in different network scenarios showed that the proposed method and baselines cannot generalize to unseen network scenarios or even create policies to handle different trained network scenarios. The proposed method used transfer learning to reduce the training steps required in each network scenario. The results show that the required number of steps could be reduced by 8 times by using transfer learning. The proposed method first used the fine-tuned agent trained in 100 episodes while completing the whole training in all episodes in parallel. When the fine-tuning process is completed, the proposed method deploys the final fine-tuned agent in the production to increase the method's performance. Aspects such as improving generalizability for unseen network scenarios and utilizing better-refined transfer learning methods still need further investigation.

5.1 Future work

Besides the proposed method's contributions, some topics still need to be investigated in future works, and the following subsections give a brief idea about them.

5.1.1 Attention mechanisms to deal with variation in the observation space

Even with the reduction in the number of combinations of slices in the observation space by ordering slice inputs, the position of the slice inputs still influences and generates different states for the same group of slices. A possible alternative is implementing a self-attention-based mechanism to deal with variations by transforming the slice entries to an intermediate domain (attention domain). An example of the usage of self-attention is presented in [75] where a permutation-invariant neural network for reinforcement learning is proposed, and results show that the RL agent can perform their assigned tasks even when changing the input position.

5.1.2 Improving critical slice protection using Safe-RL

The utilization of a safe-RL approach [76] could improve the protection of high-priority slices in the system by a risk-sensitive criterion to avoid parameter updates that may lead to higher rewards but also increase the risk of violations.

5.1.3 RAN slicing with multiple base stations

Investigate network scenarios in which slice intents must be ensured at multiple base stations. A cooperative distributed MARL method [77] could be implemented where different agents (base stations) learn to cooperate to fulfill the intents.

5.1.4 Digital twin of the RRS system

The proposed method trains and tests its performance directly in the target network scenario (in this case, a simulation), which may lead to unexpected results when the RL agent training steps are insufficient to achieve satisfactory performance in the network scenario. Designing a digital twin of the proposed simulated RRS system is essential to assess the proposed method's performance without deploying it in the target environment. Generative AI techniques could be explored to generate this digital twin [78].

Bibliography

[1] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The road towards 6G: A comprehensive survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[2] V. K. Quy, A. Chehri, N. M. Quy, N. D. Han, and N. T. Ban, "Innovative trends in the 6G era: A comprehensive survey of architecture, applications, technologies, and challenges," *IEEE Access*, vol. 11, pp. 39 824–39 844, 2023.

[3] X. Lin, "An overview of 5G advanced evolution in 3GPP release 18," *IEEE Communications Standards Magazine*, vol. 6, no. 3, pp. 77–83, 2022.

[4] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[5] B. Khodapanah *et al.*, "Radio resource management in context of network slicing: What is missing in existing mechanisms?" in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–7.

[6] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, L. Hanzo, and P. Soldati, "Learning radio resource management in RANs: Framework, opportunities, and challenges," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 138–145, 2018.

[7] M. Lin and Y. Zhao, "Artificial intelligence-empowered resource management for future wireless communications: A survey," *China Communications*, vol. 17, no. 3, pp. 58–77, 2020.

[8] GSMA, "E2E Network Slicing Requirements," Global System for Mobile Communications, Tech. Rep. 135, 6 2023, version 3.0.

[9] TM Forum IG1253, “Intent in Autonomous Networks v.1.3.0,” 2022.

[10] TM Forum IG1253A, “Intent Common Model v.1.1.0,” 2023.

[11] T. Wang, S. Wang, and Z.-H. Zhou, “Machine learning for 5G and beyond: From model-based to data-driven mobile wireless networks,” *China Communications*, vol. 16, no. 1, pp. 165–175, 2019.

[12] M. Polese, L. Bonati, S. D’Oro, S. Basagni, and T. Melodia, “ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms,” *IEEE Transactions on Mobile Computing*, 2022.

[13] M. Yan, G. Feng, J. Zhou, Y. Sun, and Y.-C. Liang, “Intelligent resource scheduling for 5G radio access network slicing,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7691–7703, 2019.

[14] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, “Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach,” *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, 2021.

[15] R. Li, C. Wang, Z. Zhao, R. Guo, and H. Zhang, “The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility,” *IEEE Communications Letters*, vol. 24, no. 9, pp. 2005–2009, 2020.

[16] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, “Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning,” *IEEE Access*, vol. 8, pp. 68 183–68 198, 2020.

[17] J. J. Alcaraz, F. Losilla, A. Zanella, and M. Zorzi, “Model-based reinforcement learning with kernels for resource allocation in RAN slices,” *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 486–501, 2022.

[18] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, “GAN-powered deep distributional reinforcement learning for resource management in network slicing,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, 2019.

[19] R. Raftopoulos, S. D'Oro, T. Melodia, and G. Schembra, "DRL-based latency-aware network slicing in O-RAN with time-varying SLAs," *arXiv preprint arXiv:2401.05042*, 2024.

[20] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The next generation wireless access technology*. Academic Press, 2020.

[21] B. Li, C. Lin, and S. T. Chanson, "Analysis of a hybrid cutoff priority scheme for multiple classes of traffic in multimedia wireless networks," *wireless networks*, vol. 4, no. 4, pp. 279–290, 1998.

[22] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE communications surveys & tutorials*, vol. 15, no. 2, pp. 678–700, 2012.

[23] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 358–380, 2014.

[24] Y.-T. Mai and C.-C. Hu, "Design of dynamic resource allocation scheme for real-time traffic in the LTE network," *EURASIP Journal on Wireless Communications and Networking*, vol. 2022, no. 1, pp. 1–24, 2022.

[25] T. Akhtar, C. Tselios, and I. Politis, "Radio resource management: approaches and implementations from 4G to 5G and beyond," *Wireless Networks*, vol. 27, no. 1, pp. 693–734, 2021.

[26] S. Niknam, A. Roy, H. S. Dhillon, S. Singh, R. Banerji, J. H. Reed, N. Saxena, and S. Yoon, "Intelligent O-RAN for beyond 5G and 6G wireless networks," *arXiv preprint arXiv:2005.08374*, 2020.

[27] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE communications magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[28] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*. GEN LTC, 2002.

[29] 3GPP TR 23.501 v16.7.0, "System architecture for the 5G System (5GS), stage 2 release 16," 2020.

[30] N. Alliance, "Description of network slicing concept," *NGMN 5G P*, vol. 1, no. 1, 2016.

[31] ITU-T Rec. Y.3112, "Framework for the support of network slicing in the IMT-2020 network," 2018.

[32] N. Alliance, "Description of network slicing concept, NGMN 5G p1, requirements & architecture," *v1. 0.8. Accessed: Nov*, vol. 2, 2017.

[33] S. E. Elayoubi, S. B. Jemaa, Z. Altman, and A. Galindo-Serrano, "5G RAN slicing for verticals: Enablers and challenges," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 28–34, 2019.

[34] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions," RFC 9315, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9315

[35] ZSM, "Intent-driven autonomous networks: Generic aspects," Zero-touch network and Service Management, Group report (GR) 135, 2 2023, version 1.1.1.

[36] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[39] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. of International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[40] C. V. Nahum, V. H. Lopes, R. M. Dreifuerst, P. Batista, I. Correa, K. V. Cardoso, A. Klautau, and R. W. Heath, "Intent-aware radio resource scheduling in a RAN slicing scenario using reinforcement learning," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2023.

[41] C. V. Nahum, L. D. N. M. Pinto, V. B. Tavares, P. Batista, S. Lins, N. Linder, and A. Klautau, "Testbed for 5G connected artificial intelligence on virtualized networks," *IEEE Access*, vol. 8, pp. 223 202–223 213, 2020.

[42] T. Guo and A. Suárez, "Enabling 5G RAN slicing with EDF slice scheduling," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2865–2877, 2019.

[43] I. Vilà, J. Pérez-Romero, O. Sallent, and A. Umbert, "Characterization of radio access network slicing scenarios with 5G QoS provisioning," *IEEE access*, vol. 8, pp. 51 414–51 430, 2020.

[44] B. Khodapanah, A. Awada, I. Viering, A. noll Barreto, M. Simsek, and G. Fettweis, "Framework for slice-aware radio resource management utilizing artificial neural networks," *IEEE Access*, vol. 8, pp. 174 972–174 987, 2020.

[45] R. Schmidt, C.-Y. Chang, and N. Nikaein, "Slice scheduling with QoS-guarantee towards 5G," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–7.

[46] R. Chen, F. Sun, L. Chen, K. Li, L. Wu, J. Wang, and Y. Yang, "Adaptive Multi-objective Reinforcement Learning for Pareto Frontier Approximation: A Case Study of Resource Allocation Network in Massive MIMO," in *Proc. of 29th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 1631–1635.

[47] L. Feng, Y. Zi, W. Li, F. Zhou, P. Yu, and M. Kadoch, "Dynamic resource allocation with RAN slicing and scheduling for URLLC and eMBB hybrid services," *IEEE Access*, vol. 8, pp. 34 538–34 551, 2020.

[48] X. Guo, Z. Li, P. Liu, R. Yan, Y. Han, X. Hei, and G. Zhong, "A novel user selection massive MIMO scheduling algorithm via real time DDPG," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2020, pp. 1–6.

[49] R. Maslennikov *et al.*, "Azimuth and elevation sectorization for the stadium environment," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2013, pp. 3971–3976.

[50] Y.-H. Nam, M. S. Rahman, Y. Li, G. Xu, E. Onggosanusi, J. Zhang, and J.-Y. Seol, "Full dimension MIMO for LTE-advanced and 5G," in *Proc. of Information Theory and Applications Workshop (ITA)*. IEEE, 2015, pp. 143–148.

[51] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "QuaDRiGa: A 3-D multi-cell channel model with time evolution for enabling virtual field trials," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 6, pp. 3242–3256, 2014.

[52] B. Mondal, T. A. Thomas, E. Visotsky, F. W. Vook, A. Ghosh, Y.-H. Nam, Y. Li, J. Zhang, M. Zhang, Q. Luo *et al.*, "3D channel model in 3GPP," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 16–23, 2015.

[53] Q. Zhu *et al.*, "3GPP TR 38.901 Channel Model," *Wiley 5G Ref: The Essential 5G Reference Online*, pp. 1–35, 2019.

[54] 3GPP, "Study on 3D channel model for LTE," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 36.873, 06 2017, version 12.5.0.

[55] ——, "3GPP TS 138.133 requirements for support of radio resource management," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2018, release 15 version 15.3.0.

[56] R. W. Heath Jr and A. Lozano, *Foundations of MIMO communication*. Cambridge University Press, 2018.

[57] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of International conference on machine learning*. PMLR, 2015, pp. 448–456.

[59] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[60] T. Akiba *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. of 25th SIGKDD international conference on knowledge discovery & data mining*. ACM, 2019, pp. 2623–2631.

[61] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, "5G URLLC: Design challenges and system concepts," in *Proc. of international symposium on wireless communication systems (ISWCS)*. IEEE, 2018, pp. 1–6.

[62] S. Jaeckel, L. Raschkowski, L. Borner, K. Thiele, F. Burkhardt, and E. Eberlein, "QuaDRiGa - quasi deterministic radio channel generator, user manual and documentations," 2021.

[63] 3GPP, "Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 28.541, 6 2024, version 17.15.0.

[64] "Internet connection speed recommendations — help.netflix.com," https://help.netflix.com/en/node/306, [Accessed 16-09-2024].

[65] "System Requirements for GeForce NOW Cloud Gaming — nvidia.com," https://www.nvidia.com/en-us/geforce-now/system-reqs/#windows-pc, [Accessed 16-09-2024].

[66] J. K. Terry, N. Grammel, S. Son, B. Black, and A. Agrawal, "Revisiting parameter sharing in multi-agent deep reinforcement learning," *arXiv preprint arXiv:2005.13625*, 2020.

[67] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.

[68] A. Charu C, *Neural networks and deep learning: A textbook*. Springer, 2018.

[69] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[70] R. Team, "Rllib algorithms - ppo," https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#ppo, 2023, accessed: 2024-07-07.

[71] C. Nahum, "6G Radio Scheduler Simulator," September 2024. [Online]. Available: https://github.com/lasseufpa/sixg_radio_mgmt

[72] 3GPP, "Service requirements for the 5G system," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 22.261, 12 2023, version 19.5.0.

[73] ——, "Unmanned Aerial System (UAS) support in 3GPP," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 22.125, 12 2023, version 19.1.0.

[74] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[75] Y. Tang and D. Ha, "The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 574–22 587, 2021.

[76] J. Garcia and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[77] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[78] A. Celik and A. M. Eltawil, "At the Dawn of Generative AI Era: A tutorial-cum-survey on new frontiers in 6G wireless intelligence," *IEEE Open Journal of the Communications Society*, 2024.