ARTIFICIAL INTELLIGENCE BASED FLEXIBLE PREAMBLE ALLOCATION FOR RADIO ACCESS NETWORK SLICING IN 5G NETWORKS

5G AĞLARINDAKİ RADYO ERİŞİM AĞI DİLİMLEMLEME İÇİN YAPAY ZEKA TABANLI ESNEK BAŞLANGIÇ SİNYALİ PAYLAŞTIRMA

AHMET MELİH GEDİKLİ

ASSOC. PROF. DR. SEVİL ŞEN
Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering

2021

ÖZET

5G AĞLARINDAKİ RADYO ERİŞİM AĞI DİLİMLEMLEME İÇİN YAPAY ZEKA TABANLI ESNEK BAŞLANGIÇ SİNYALİ PAYLAŞTIRMA

Ahmet Melih GEDİKLİ

Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Doç. Dr. Sevil ŞEN
İkinci Danışman: Doç. Dr. Mehmet KÖSEOĞLU
Haziran 2021, 112 sayfa

Radyo erişim ağı dilimlemedeki en büyük zorluklardan birisi, ağa erişmek için bağlantı kurulma aşamasında birden fazla cihazın aynı radyo erişim kanalını kullanmalarından kaynaklanmaktadır. Halihazırda, rastgele kanal erişiminde yaşanan sıkışıklık, dağınık şekilde cihazların kanala aynı anda çıkmaya çalışmaları durumunda çok ciddi bir boyuta varmaktadır. Bu tarz durumlar, kanala erişmeye çalışan tüm cihazların iletişiminde önemli seviyelerde gecikmeye neden olmaktadır. Bu nedenle, radyo erişim ağı dilimlenmesine olanak sağlanabilmesi ve kaynakların dinamik yönetilebilmesi için, rastgele kanal erişimi kaynakları, farklı servis tipleri için dağıtılabilir hale gelmelidir.

Rastgele kanal erişimi sırasında, cihazlar bir senkronizasyon sinyali seçerek yayınlarlar. Eğer birden fazla cihaz aynı sinyali yayınlarsa baz istasyonunda çakışma meydana gelir. Kanal erişimi sırasında bir servis sınıfını diğerinden izole edebilmek için uygulanan yöntemlerden biri de senkronizyon sinyallerini alt gruplara ayırdıktan sonra servislere bu alt grupları paylaştırmaktır. Statik paylaşım, servislerden gelen trafiğin dinamik olarak değiştiği durumlarda verimsiz hale gelmektedir. Bu yüzden dinamik paylaşım, gecikmeyi ve çakışmayı istenen seviyelerde tutabilmek için uygun bir yöntemdir.

Bu tez çalışması, senkronizasyon sinyallerinin alt gruplara dinamik olarak ayrılması için derin takviyeli öğrenme ve genetik algoritma temelli yöntemler önermektir. Önerilen yöntemler, kısıtlı sayıdaki senkronizasyon sinyallerini farklı servis sınıflarına, servislerin önceliklerine ve ağdaki trafiğin durumana göre dağıtarak servis sınıfları arasında sanal bir izolasyon yaratmaktadırlar. Çalışma kapsamındaki sonuçlar, önerilen yöntemlerin, değişen trafik ihtiyaçlarına uyum sağlayarak senkronizasyon sinyallerini gruplara dinamik şekilde dağıttığını göstermektedir.

Anahtar Kelimeler: Derin Takviyeli Öğrenme, Genetik Algoritma, Senkronizasyon Sinyali Paylaştırma, Ağ Dilimleme, 5G, Radyo Erişim Ağı, Makineler Arası İletişim

ABSTRACT

ARTIFICIAL INTELLIGENCE BASED FLEXIBLE PREAMBLE ALLOCATION FOR RADIO ACCESS NETWORK SLICING IN 5G NETWORKS

Ahmet Melih GEDİKLİ

Master of Science, Computer Engineering Department
Supervisor: Assoc. Prof. Dr. Sevil ŞEN
Co-Supervisor: Assoc. Prof. Dr. Mehmet KÖSEOĞLU
June 2021, 112 pages

One of the most difficult challenges in Radio Access Network (RAN) slicing occurs in the connection establishment phase where multiple devices use a common Random Access Channel (RACH) to gain access to the network. It is now very well known that RACH congestion is a serious issue in case of sporadic arrival of machine-to-machine (M2M) nodes and may result in significant delay for all nodes. Hence, RACH resources are also needed to be allocated to different services to enable RAN slicing so that the resources can be dynamically allocated.

In the RACH procedure, the nodes transmit a selected preamble from a predefined set of preambles. If multiple nodes transmit the same preamble at the same RACH opportunity, a collision occurs at the eNodeB. In order to isolate one service class from others during this phase, one approach is to allocate different preamble subsets to different service classes.

Static allocation of those subsets, however, may result in inefficiencies when the traffic generated by each service changes significantly over time. Hence, dynamic allocation is more suitable to be able to keep the delay and collision probabilities around the desired levels.

This work proposes adaptive preamble subset allocation methods using Deep Reinforcement Learning (DRL) and Genetic Algorithm (GA). The proposed methods can distribute preambles among different service classes according to their priority and the traffic in the network, providing a virtual isolation of service classes. The results indicate that the proposed mechanisms can quickly adapt the preamble allocation according to the changing traffic demands of service classes.

Keywords: Deep Reinforcement Learning, Genetic Algorithms, Preamble Allocation, Network Slicing, 5G, Radio Access Network, M2M

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisors Assoc. Prof. Sevil ŞEN and Assoc. Prof. Mehmet KÖSEOĞLU who have always encouraged me and guided me with their valuable contributions and criticisms at all stages of my dissertation.

Besides I would like to thank my thesis committee members for insightful comments for this thesis.

Finally, I thank my beloved wife and family for their continued support throughout my educational life.

CONTENTS

TABLES

FIGURES

SYMBOLS AND ABBREVIATIONS

Abbreviations

IoT	**I**nternet **o**f **T**hings
RL	**R**einforcement **L**earning
DRL	**D**einforcement **R**einforcement **L**earning
RAN	**R**adio **A**ccess **N**etwork
RAO	**R**andom **A**access **O**pportunity
CN	**C**ore **N**etwork
NS	**N**etwork **S**licing
M2M	**M**achine to **Machine**
H2H	**H**uman to **H**uman
PRACH	**P**hysical **R**andom **A**ccess **CH**annel
RACH	**R**andom **A**ccess **CH**annel
VNF	**V**irtual **N**etwork **F**unction
eMBB	**e**nhanced **M**obile **B**road**B**and
uRLCC	**u**ltra **R**eliable **L**ow **L**atency **C**ommunications
mMTC	**m**assive **M**achine **T**ype **C**ommunications
IMT	**I**nternational **M**obile **T**elecommunications
QoS	**Q**uality **o**f **S**ervice
LTE-A	**L**ong**T**erm **E**valuation - **A**dvanced
SDN	**S**oftware **D**efined **N**etworking
NFV	**N**etwork **F**unction**V**irtualization
BBU	**B**ase**b**and **U**nit
RRH	**R**emote **R**adio **H**ead
C-RAN	**C**loud - **R**adio **A**ccess **N**etworking
cap-ex	**cap**ital-**ex**penditure
op-ex	**op**erational-**ex**penditure
API	**A**pplication **P**rogramming **I**nterface

SBA	**S**ervice **B**ased **A**rchitecture
V2X	**V**ehicle to **E**verything
GA	**G**enetic **A**lgorithm
ACB	**A**ccess **C**lass **B**arring
NEAT	**NE**uroevolution of **A**ugmenting **T**opologies
ANN	**A**rtificial **N**eural **N**etworks
TRPO	**T**rust **R**egion **P**olicy **O**ptimization
PPO	**P**roximal **P**olicy **O**ptimization
BS	**B**ase **S**tation

1. INTRODUCTION

Internet of Things (IoT) are becoming more prevalent in a wide range of usage areas such as smart grids, personal communications, controlling traffic flow on roads, smart driving, and smart healthcare [6]. It was recently reported that a total of 75 billion IoT devices are expected to be in operation globally by 2025 [7]. Communication among these IoT devices are named as machine-to-machine (M2M) communication and the connections resulting from M2M communication are expected to be more than half of the global connected devices and connections by 2022 [8]. Hence, M2M communication is one of the main drivers of the evolution from 4G to 5G. While nearly half of these connections are resulted from home applications, the number of connections resulting from connected work and connected cities applications are showing an increasing trend in recent years.

M2M communication is the main driver of fully-automated production lines and connected cities. It has different characteristics than human-to-human communication (H2H) who has been main driver of Long-Term Evolution (LTE). In H2H, most of the traffic is carried on the downlink, sessions are longer and less frequent. On the other hand, M2M devices mostly utilize the uplink and sessions are shorter and more frequent. For example, a smart meter may wake up, send its data, and then immediately go back to sleep, thus, just for a short data transmission, an uplink physical resource allocation request has to be made. In short, these devices generally require low bandwidth, however, due to large number of them and their wake-up nature the up-link physical resource allocation for them are different from H2H [9].

5G has not only pledged to provide services for above described H2H and M2M types but pledged more, yet, up-to LTE it was sufficient. Therefore, LTE and 3G generally divided the service characteristics into two as M2M and H2H. However, with the 5G concept at least three but up to 5 different service characteristics are defined [10, 11]. The priorities of these service types may differ from each other. For example, while a remote surgery service should have the highest priority since it must be ultra reliable and provide ultra low latency [12], an IoT service which collects sensor data can have the lowest priority in a network. Thus, the increasing number of devices and services with changing characteristics cause us to review the bottlenecks of the communication.

One of the bottlenecks in Radio Access Network (RAN) communication is the Physical Random Access Channel (PRACH) allocation procedure [13]. This period is called Random Access Opportunity (RAO) and the whole procedure is maintained by base stations (BSs). The limited number of orthogonal distinguishable signals (preambles) arise the need of dynamic distribution of them to the services. In both LTE and 5G there are 54 PRACH preambles each one selected by devices randomly and transmitted for channel allocation requests. If multiple nodes transmit the same preamble at the same RAO, a collision occurs at the BS. To isolate one service class from others during this phase, one approach is to divide RACH resources to different service classes. Static allocation of those resources, however, may result in inefficiencies when the traffic generated by each service changes significantly over time. Hence, slicing the resource into dynamic groups is more suitable to be able to keep the delay and collision probabilities around the desired levels.

The Network Slicing concept arose from changing environment and diversified service requirements of devices ranging from delay-tolerant smart metering systems to mission critical smart driving applications [14]. The need for supporting applications with different service requirements on a single network has led to the development of the network slicing approach. It is an assignment of network resources to separate services such that a service can be run virtually independent of other applications in the network. RAN slicing is a challenging aspect of network slicing due to the shared nature of the wireless spectrum.

One of the most difficult challenges in RAN slicing occurs in the connection establishment phase where multiple devices use a common random access channel (RACH) to gain access to the network. It is now very well known that RACH congestion is a serious issue in the case of sporadic arrival of M2M nodes and may result in significant delay for all nodes. Hence, RACH resources are also needed to be allocated to different services to enable RAN slicing. These fundamental architectural need of changes and the increasing number of IoT devices with new requirements make LTE and 3G incompatible, since these devices need massive M2M communication [15].

The predecessor of 5G, LTE Advanced (LTE-A) standard is not designed to support various M2M communication services [16], rather it is designed for wide-band communication. Therefore, many problems arises with the current infrastructure for such types of communication services. While some well defined requirements such as 1ms latency for low latency services and 10Gbps for mobile broadband services etc. are implemented already, the service types can not utilize all the features brought by these requirements at the same time. Thus, in order to provide those requirements in a combined and dynamic manner a new approach for random access mechanism should be discussed, so that, each service type must have completely separated random access channel portion on the side of physical random access resources. These portions can be named as preamble allocation subsets. The resources of preamble allocation subsets must be dynamic in order to support real-world scenarios. For example, in power outage and restore case IoT devices may try to re-connect to network at the same time if no other prevention is conducted since all wake up at the same time. However, the devices of other services usually are not affected from power outage. As a result there may be overload in up-link resources of IoT service class in RACH. For a short amount of time, it is needed to increase the reserved resources of preamble allocation subset for IoT service.

In order to increase or decrease the resources we need to have flexible preamble subset allocation methods for changing environment like the scenario given above. Since immediate response is also a 5G requirement, allocating preamble subsets must be as fast as possible. The channel requirement for different services may change over time, so the preamble count for such services needs to be changed dynamically. However, the balance between such service types should be preserved and the channel should be fairly shared.

In this thesis, artificial intelligence based flexible preamble allocation methods are proposed for radio access network slicing in 5G. These methods are instantiable Virtual Network Functions (VNF) in any BS which use Network Slicing concept to slice PRACH preambles. The 5G concepts define the technical needs, limits and definitions to set the new boundaries and introduce new capabilities. Therefore, while H2H related aspects of network design may remain same, there should be fundamental changes and new concepts come with 5G to support M2M devices. In this manner first, 5G is explored step by step to explain how it enables network slicing in detail, than Reinforcement Learning (RL) and Genetic Algorithms (GA) are explored in order to divide channel using network slicing for any number of slices and any prioritization levels.

1.1. 5G Requirements

The global operators started to use 5G networks in late 2019. Its world-wide access is expected to be available in 2021 [17]. Meanwhile the mobile and streaming data are increasing every day. The mobile data traffic is 200 times more in 2020 than in 2010 and it is expected to be 100 times more in 2030 than in 2020 [18]. Moreover, with the increasing of data traffic, the new era requires higher number of connected devices with new service types for regular users or for industry. Therefore, 5G has to deliver us increased operational networking performance. The key factors in order to deliver such performance in 5G are listed as latency, connection density and throughput [19]. Combining the performance enhancement on those factors makes us one step closer to the new era. In the following, some use cases and enabler requirements of those use cases of mobile internet within the concept of 5G currently are given:

- Ultra HD and 3D video streaming

  Requirement: 1 Gbps of transmission rate

- Online gaming and game streaming

  Requirement: Imperceptible latency

Some other use cases and related requirements in IoT are listed below:

- Smart agriculture

  Requirement: Massive device communication

- Automotive driving/Internet of vehicles

  Requirement: %100 Reliability

Combining all above it is clear that 5G has to support variety of use cases, which needs different requirements. These combination of use cases and requirements formed the magic 5G triangle. Each corner of the triangle given in Fig. 1.1. represents the communication types which differs from other ones with their different use case scenarios. These communication types are Enhanced Mobile Broadband (eMBB), Ultra Reliable and Low Latency Communications (URLLC) and Massive Machine Type Communications (mMTC). In the use case scenarios of eMBB, a stable channel with large payload transmission is required in order to make possible to have capacity enhancement of using the existing bandwidth. On the other

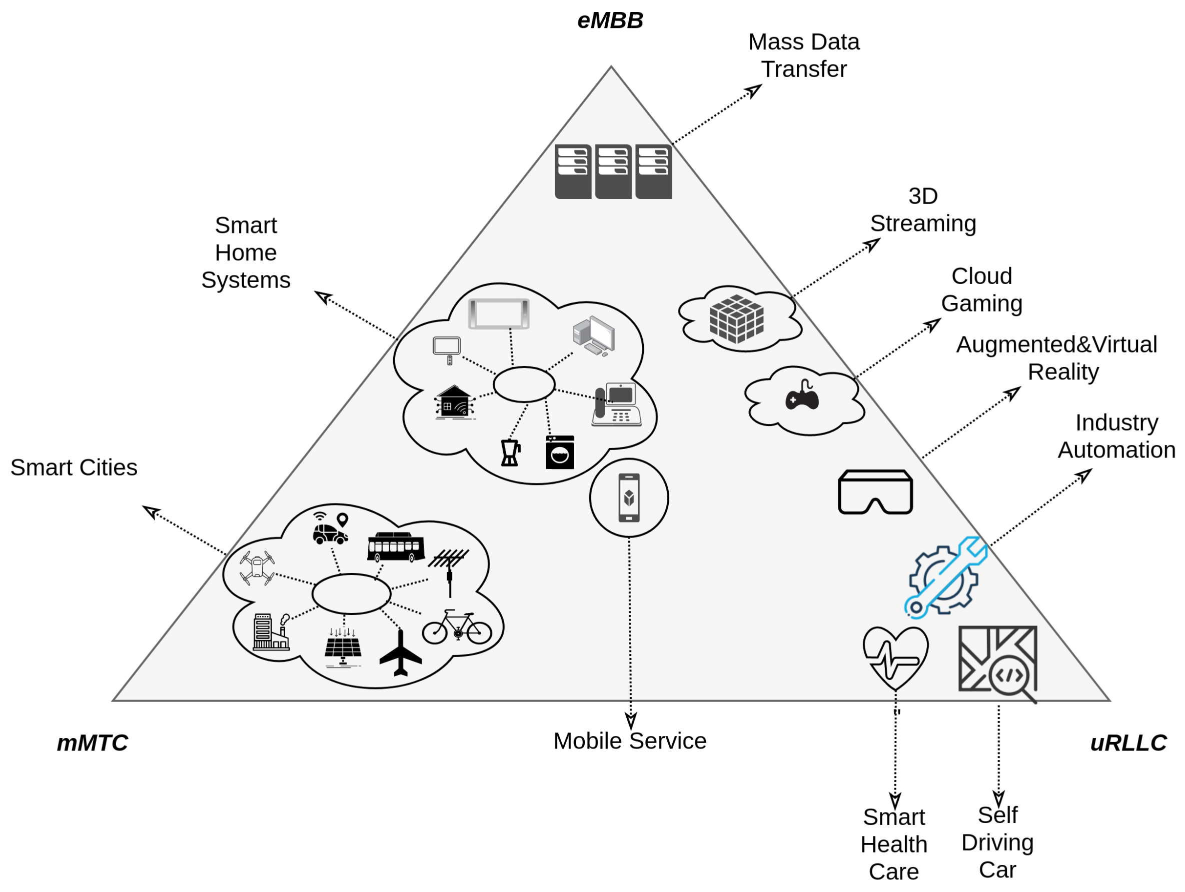


Figure 1.1. The 5G triangle [2]

hand, whereas in mMTC use case scenarios massive connectivity of devices is required, no-loss & no-latency small payload transmission on communication are required in URLLC [20].

5G aims to support these three generic communication types given in Fig. 1.1.. Whereas each representative corner in the triangle focuses on a set of use cases, the combination of different aspects of these types are usually needed in real life. For example, online gaming and game streaming requires a communication type that resides between eMBB and URLLC. The key capabilities with their different levels of need by three types can be seen in Fig. 1.2.. It is clear that the highest effect of connection density is on mMTC among all these three types. This connection density creates a bottleneck at the RAN side.

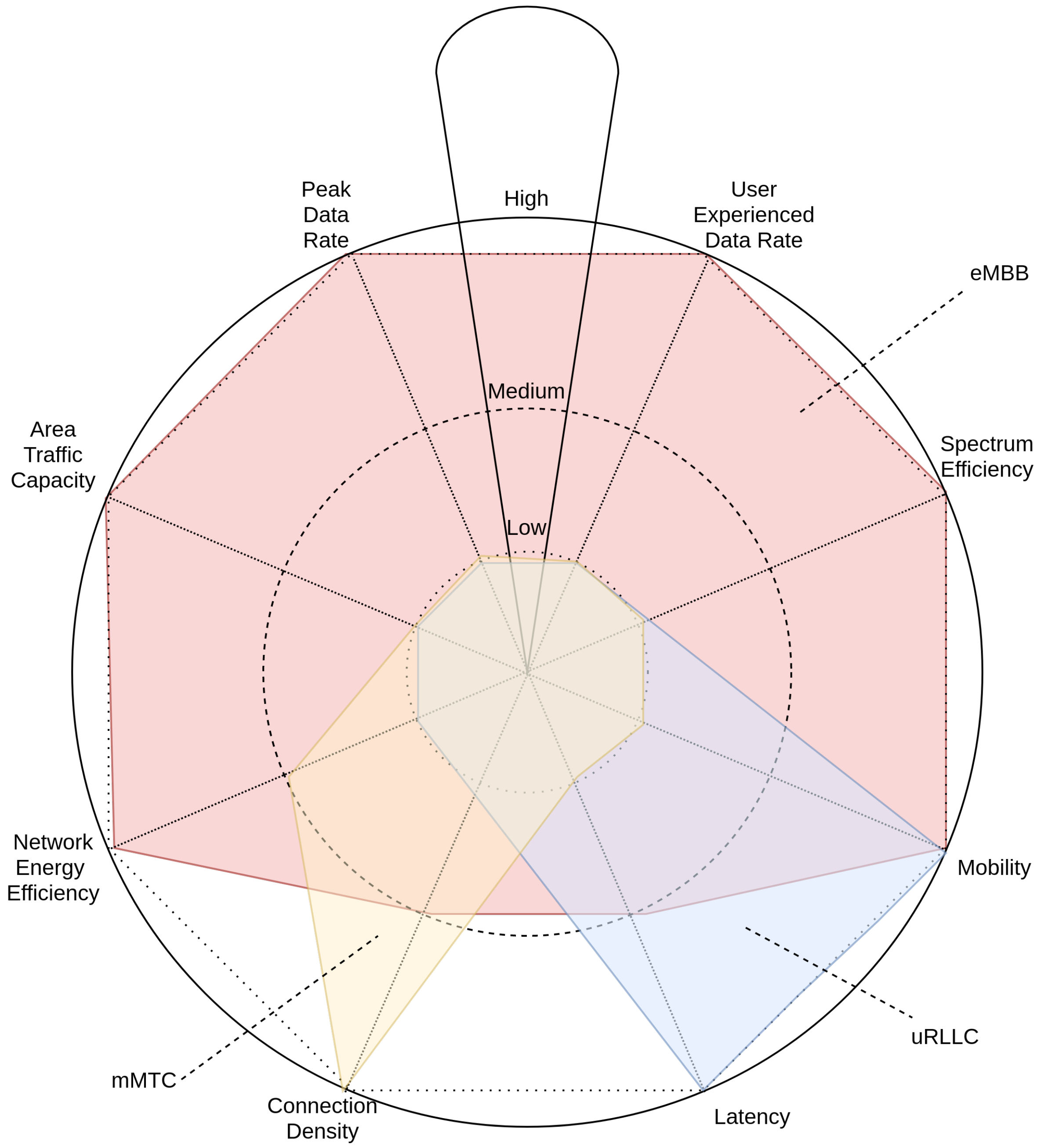


Figure 1.2. The 5G usage scenarios with the different level of need of key capabilities [2]

1.2. 5G RAN

RAN is an essential part of cellular network infrastructure which provides wide-area access network for the connection. The radio resource management, mobility management and data encryption are handled in RAN side. It has been in use since the beginning of cellular technology. Up to LTE, the RAN was providing business as usual approach wide-area access network to mobile phones mostly [21]. The arrival of LTE and especially the control plane enhancements paved the way for differentiated QoS [22]. With the extreme and diverse requirements in the new era, the demand for new business models is shown up.

Although 5G RAN is integrated with LTE-A, to support new demands and flexibility software defined networking (SDN) is proposed. To have this ability, the control and user plane of networking has been separated. The control and user plane of wireless networks define and specify network functionalities which orchestrate the whole network. While the user plane is responsible for forwarding the payload from source to destination, the control plane controls the user plane by defining new routing path, mobility actions, radio resource management etc. [23]. The benefits of separating the user plane from the control plane are listed briefly as below:

- The tight coupling of these planes brings difficulties of upgrading and changing one plane without changing the other. The separation paves the way of updating/upgrading the planes without affecting the other.
- The independent evolvement of the planes allows us to build new and target-specific network functions in a short amount of time.
- The idea is one of the important steps in order to enable the network slicing concept.

The Cloud Radio Access Network (C-RAN) is another important step for network slicing by introducing a paradigm shift using horizontal functional splitting idea [3, 23]. In the traditional cell architecture, each BS owns a dedicated Remote Radio Heads (RRH) and Base Band Unit (BBU). Generally, while RRHs perform all RF functionality like transmitting, receiving, filtering and amplification, the BBU process digitizes signals. Due to new requirements in the M2M and IoT era, the densification problem has introduced a scalability issue. This challenge is solved by requiring new BSs. On the other hand, increasing number of

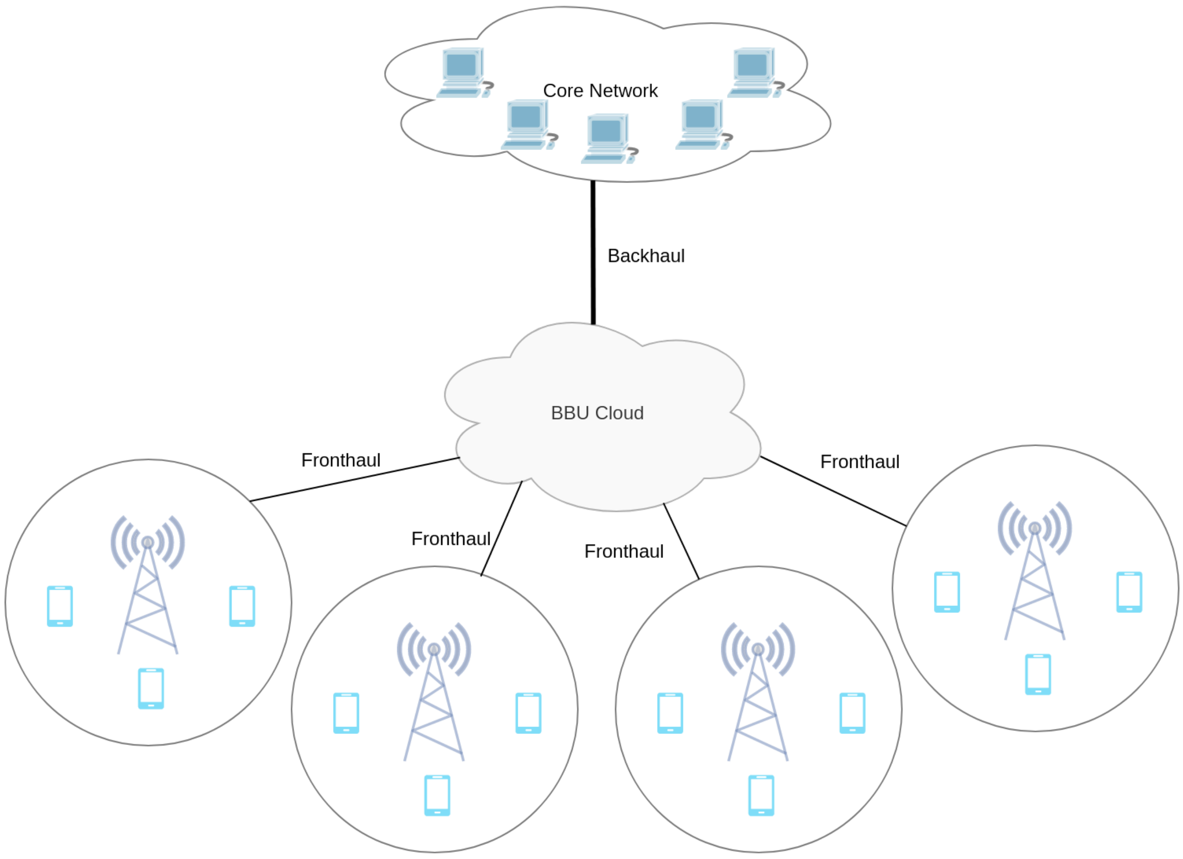


Figure 1.3. The architecture of C-RAN based cellular networks [3]

BSs will increase also the capital expenditure (cap-ex) and operating expenditure (op-ex). C-RAN solves this problem by horizontal functional splitting. Basically, it creates centralized virtual BBU pool which can be used by all BSs in the RAN as shown in Fig. 1.3.. The BBUs are virtually deployed and clustered into the pools. While op-ex decreases due to the centralization and BSs cap-ex decreases due to removing the process of deploying physical BBUs. Furthermore, the virtualization of BBUs facilitates network slicing concept by enabling virtualization of RAN functions [24, 25]. Thereby, the radio resources can be abstracted to create multiple independent slices. These independent multiple virtualized slices can be dynamically adjusted on demand of changing user requirements or environmental fluctuation.

1.3. 5G Core Networks (5G CN)

The major difference between the 5G Core (5GC) and the core of former cellular networks is that the functionality of Network Functions (NF) that is provided and accessed via Application Programming Interface (API) [26]. This results in Service Based Architecture (SBA) which provides loosely coupled set of services. Thanks to SBA, with NFs new combinable services can be dynamically and independently instantiated. The ability to orchestrate these core NFs allow us to instantiate NFs which can be very close to subscribers and service consumers. These capabilities led to emerge the Edge Cloud concept which is also needed for the requirements of mMTC and URLLC. For example, mMTC service which needs computational resources and have low battery capabilities can offload their computational needs to the edge cloud in order to reduce power consumption. URLLC service which needs ultra low latency communication can fetch the content immediately since the resources are moved to edge cloud [27]. Also, the burden of backhaul is decreased since the data is processed in the edge. Hereby, the service providers are able to offer mMTC and URLLC services with the invention of the Edge Cloud concept.

1.4. Network Slicing Based Network Architecture of 5G

5G aims to provide multi-tenant and service-tailored connection. Due to the very nature of services, constructing a physical infrastructure in order to support all service types is not feasible. Even if that kind of infrastructure is built up for all existing services up to now, 5G implies that the service consumers may need on demand new service types with different characteristics in the future [28]. Besides, users may expect to have privilege and access rights on a portion of network resources, meanwhile, they may want ultimate flexibility of using these resources with also scalable dynamic payment plans [29]. These specifications conclude on new and dynamic service capabilities which support wide range of requirements. The network slicing concept has emerged as a solution for these varying requirements and provides end-to-end, on-demand and tailored connection. As pointed out earlier in Section 1.1. and 1.2., the softwarization and virtualization of network components enable network slicing. In brief, with the separation of hardware and software, network functions became independent from underlying physical resource. The new virtualized network functions are interconnected with each other so that they form a network service. The new formed network is managed from a centralized point. This centralization is called SDN. In this context, network slicing can be defined as a unification of virtual network functions which are specific to a service or a customer over a common network infrastructure and also instantiated and chained using SDN. Network slices must be mutually isolated, independently controllable and on-demand [30].

Fig. 1.4. shows an example system architecture of 5G using network slicing. As shown in the figure, since eMBB requires both low-moderate latency and extreme data throughput, the eMBB service slice has cache and data unit VNFs on edge cloud On the other hand, for the IoT slice which serves vast amount of devices, application and data unit VNFs are placed on the core cloud to support external services demanded by different customers. Finally, for the uRLLC service slice autonomous driving use case is given as an example. In order to support ultra reliable and low latency communication requirements of vehicle to everything (V2X), this VNF is placed in both the edge and core cloud. Hence, the old-fashion traditional CN is evolved to core cloud in this new architecture. The core cloud is responsible for orchestrating the slicing, mobility and interference management etc. The architecture tries to separate the user and control plane as possible as it could be and moves content storage and data processing functions to edge cloud.

1.5. Major Contributions of the Thesis

The increasing significance of RAN resource allocation in 5G highlights flexible preamble allocation. This study aims to find a solution to the RAN slicing problem that takes into account services with varying prioritization. It explores the use deep reinforcement learning (DRL) in order to solve the optimum resource allocation problem in RAN slicing. The proposed methods could be integrated with other proposed methods like 3GPP proposed Access Class Barring (ACB). The main contribution of the thesis are listed below.

- DRL is proposed in order to solve the preamble allocation problem in 5G. As far as we know, this is the first study in the literature that explores the use of DRL for solving the preamble allocation problem for network slicing.
- The proposed method is evaluated thoroughly with 2, 3 and 5 slices using on simulated network scenarios where traffic could increase or decrease suddenly or constantly. The

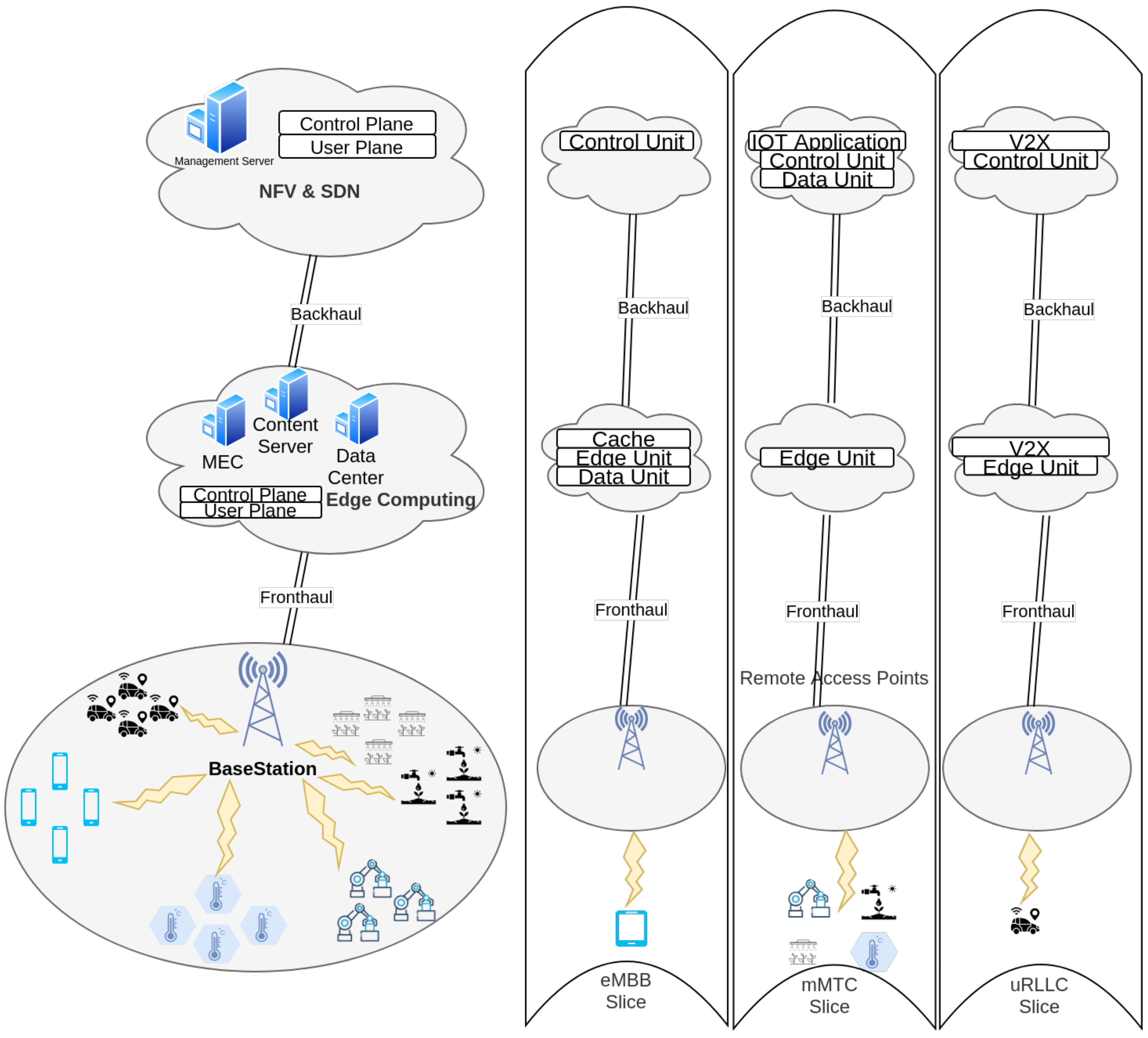


Figure 1.4. The network architecture based on network slicing in 5G [4]

experimental results show that the proposed DRL based method improves the overall performance of the slices significantly by carrying out the prioritization dynamically.

- The proposed method is compared with the approach proposed in [1] by using a scenario conducted in [1]. The results show that the proposed method generates close results to the approach proposed in [1] . Moreover, the *ideal algorithm* mentioned in [1] is also used as one of the comparison basis in order to show the success of the proposed method. The results show that the proposed method can generate more successful results than the *ideal algorithm* for some scenarios.

- Three reward functions, namely Successful Preambles Reward Function(SRF), Proportional Reward Function (PRF) and Collision Penalizing Reward (CRF) are proposed for the reinforcement learning formulation and mathematically analyzed. The most successful one is shown to be CRF, since it penalizes collisions and incentivize methods to prevent collision.

- A NEAT-based approach is also proposed as another promising solution. It is partially successful since it can prioritize slices successfully while only using CRF with fewer than 3 slices.

1.6. Organization of the Thesis

The organization of the thesis is given as follows:

Chapter 2 presents the essential background knowledge. It firstly presents the procedures of up-link channel allocation. Here, the main definitions used throughout the thesis such as preambles, RACH, RAO are given. Then, the DRL and GA methods used in the thesis are introduced. Finally, the main motivation behind using of DRL and GA for the preamble allocation problem in the study is given.

Chapter 3 presents the related studies in the literature. Especially the proposed approaches for RACH congestion control and RAN slicing are reviewed. In addition, the studies that use artificial intelligence for allocation of resources other than PRACH such as CPU and power are given. The discussion of related studies are also presented at the end of the chapter in order to highlight the difference of the proposed approach from the related studies. The study used for comparison [1] is also introduced in this chapter.

Chapter 4 describes the proposed approach based on DRL in order to dynamically allocate the preambles to the subsets of services. The DRL framework which uses Trust Region Policy Optimization (TRPO) and GA framework called Neuroevolution of Augmenting Topologies (NEAT) are presented. The reward and fitness functions and analysis of them are provided.

Chapter 5 introduces the training and testing experiments carried out in the thesis. The performance of the proposed methods using simulations is presented. Extensive simulations are conducted for 2,3 and 5 slices with different reward and fitness functions. The analysis of experimental results are discussed thoroughly.

Chapter 6 concludes the thesis with a brief summary of the proposed approach. The chapter ends with the limitations and possible future topics related to this study.

2. BACKGROUND

The introduction to the thesis is given in the previous chapter. After explaining the requirements of 5G and why these requirements need virtualization of network infrastructure in the first chapter, in this chapter, the background information is given in order to base upon a comprehensible foundation for the work in this thesis. First of all, the concept of preamble allocation is given in Section 2.1., following the RAO procedure is given in Section 2.1.3.. Finally, the general information about the methods used in this thesis, namely DRL and GA, are given in Section 2.2. and 2.3. respectively.

2.1. PRACH Preamble

The cellular network synchronizes BSs with UEs while they communicate. The synchronization in down-link (from BS to UEs) is easy to handle since it is managed from a single source and the sync signal can be broadcast from BS device to everybody periodically [31]. There is no interference in this type of signal since the BS is established in this manner. However, the case for the up-link synchronization is not the same, since, there can be many UEs in the environment, the synchronization process cannot be dedicated to only one UE. Therefore, the up-link synchronization needs additional care.

The PRACH procedure is used in cellular networks in order to allocate radio resources to users. The up-link in PRACH is used by users to initiate access requests. A PRACH preamble is the generated signal from an UE to BS to conduct RACH procedure. The signal carries a “signature” which is recognized by BS. There are in total 64 distinctive preamble signals available for UEs in 5G as were in LTE-A. UEs randomly select one of these preambles to generate signals.

The signals have mainly two purposes. The first one is to have up-link synchronization between UE and BS and the second is obtaining the radio resource to communicate. In LTE-A, not all 64 preambles are reserved for contention-based RACH, the 10 preambles are reserved for contention-free access by BS. In this type, BS specifically informs a UE to use a reserved preamble so that no collision occurs. As a result, there are left 54 preambles reserved for contention-based. Since, UEs randomly select a preamble from 54 preambles, there is a possibility that multiple UEs may select the same preamble. The possibility of collision is the reason why this procedure is called contention-based RACH. In this approach, immediately after the contention process, there is also contention resolution process. The contention free and contention-based RACH are explained more in the subsequent sections.

2.1.1. Contention-free RACH

Even though the focus of this thesis study is not contention-free RACH, yet, in order to fully understand the contention-based RACH the steps of contention-free RACH is also given in this section. Fig. 2.1. shows the steps of both contention-free and contention-based RACH. There are 3 messages in contention-free RACH and the first message is from BS to UE. With the first message BS assigns a specific preamble to the UE, following that, UE sends the assigned preamble. In RA response, the connection is granted.

2.1.2. Contention-based RACH

In contention-based RACH, there are 4 steps as seen in Fig. 2.1.. In the first step, UE randomly selects a preamble and sends it. For example, if more than one UE sends the same preamble in this step, BS device sends the same random access response to these devices. In this scenario, these UEs get the same resource allocation (time/frequency domain). As a result, they transmit the exact same information on the same time/frequency domain on the 3rd step (RRC connection request). The first and most probable possibility is the interference of these messages so that BS is not able to decode them. Since, the 4th step is not reached anymore, the UEs get back to the first step and try the RACH procedure from the beginning. As a result, none UEs are successful allocating the up-link resource also a limited resource, preamble, is wasted. The focus of this work is to propose a solution for the contention-based RACH procedure. Rather than changing the mechanisms of contention-based RACH, the aim is to make use of network slicing and network virtualization and create a dynamic model.

2.1.3. Random Access Opportunity

In time domain, the transmissions in LTE are divided into 10ms length radio frames each of which is also divided into ten sub-frames of 1ms length. The PRACH procedure is an interval

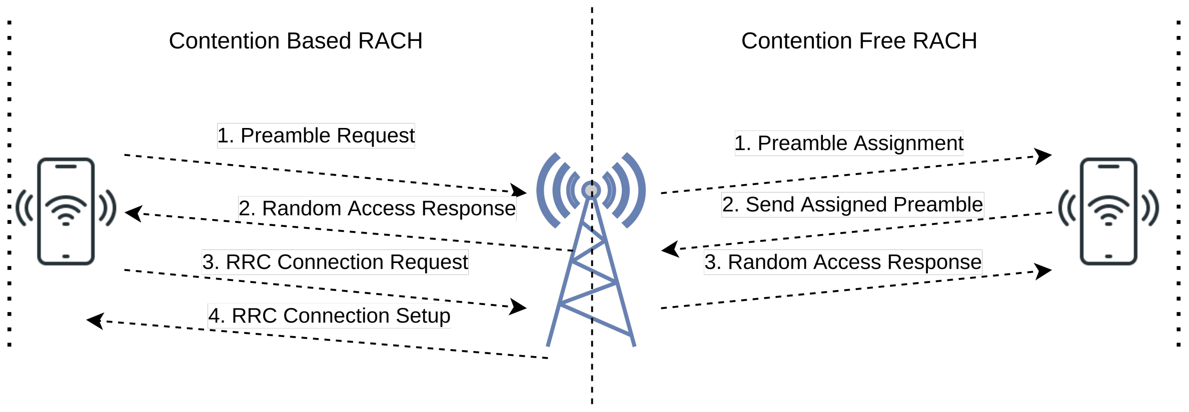


Figure 2.1. The steps of contention-based and contention-free RACH [5].

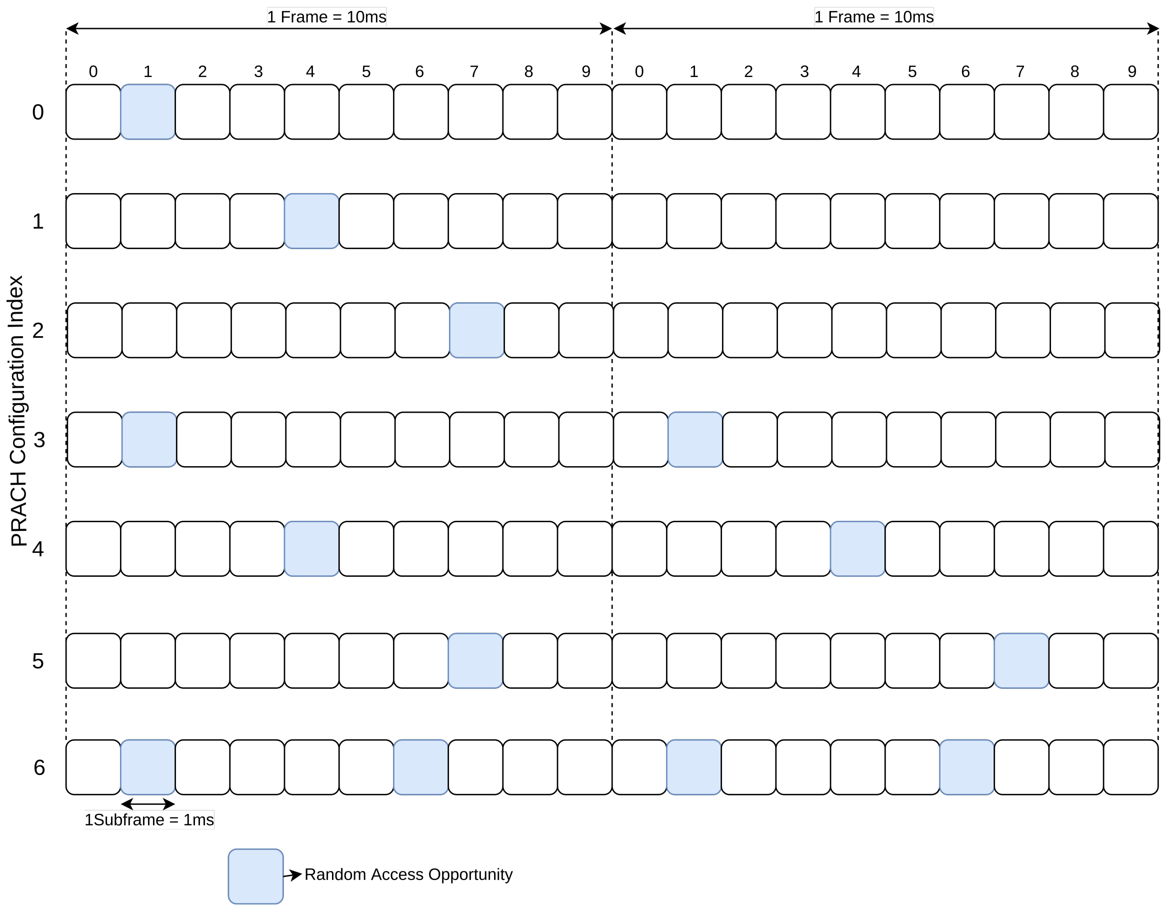


Figure 2.2. The PRACH configuration index and RAO sub-frame relation.

which takes one sub-frame. The PRACH configuration index specifies in which sub-frame UEs can send PRACH preamble. This configuration is broadcast by BS and the devices around the BS sends their preambles with respect to this information. The time interval in which the devices can send their preambles is called RAO. Fig. 2.2. shows on which sub-frame(s) RAO is held with respect to configuration index in LTE/LTE-A. For example, when the PRACH Configuration Index is equal 3 the RAO is held every 10ms, when the index is equal to 6 it is held every 5ms. In Fig. 2.2. the first 6 configuration index relation is shown but there are a lot more than 6 configurations in both LTE and 5G. The PRACH preamble format and PRACH configuration index properties for 5G is nearly same with LTE-A. In this thesis, the configuration index 3 is employed to have a single RAO in 10ms interval.

2.2. Deep Reinforcement Learning

In this thesis study, DRL based approach is proposed in order to solve the preablem allocation problem in 5G. Since DRL is based on Reinforcement Learning (RL), firstly the main components of reinforcement algorithm is explained below.

2.2.1. Reinforcement Learning

Reinforcement Learning (RL) is a field of machine learning where the machines can learn from their actions by using the results of their actions. In the standard reinforcement learning model there exists an agent connected to the environment surrounding it. There is/are input(s) to the agent which represent the current state of environment. The response to these states that generated by an agent is called actions. The iterative actions and states evolve environment in time.

In each iteration, the action of agent is evaluated and some reward/punishment signal is generated. The agent should choose the actions such that in time the sum of reward signals must increase and the sum of punishment signals must be decreased [32]. Fig. 2.3. shows the interaction of the entities of the standard model which consists a discrete set of environment states(s_t), a discrete set of actions(a_t) and a set of reinforcement signals(R_t). As it is called Markov Decision Process (MDP), so that, the aim of agent is to develop a policy π mapping the states to actions so that the overall reinforcement signals will be maximized in long run.

Figure 2.3. Conceptual schema of the reinforcement learning model.

There are two main approaches while solving problems using RL. There exist methods based on value function and methods based on policy search. Another third approach is the hybrid of value function and policy search methods and is called the actor-critic approach.

2.2.1..1 Value Function: The majority of reinforcement learning algorithms try to estimate the value functions. In detail, the value is expected return in a given state(s_t). Therefore, the value function is a function of states which measures the level of quality to be in given state. Recalling π is a mapping from states to actions, the state value function $V^{\pi}(s)$ is the expected return. E denotes the expected value when agent follows the policy π.

$$V^{\pi}(s) = E[R|s, \pi] \tag{1}$$

If the optimal policy π^* has value function $V^*(s)$ than the optimal state-value function $V^*(s) = max(V^{\pi}(s))$ for every state. To retrieve the optimal state-value return in state s_t an action must be selected that maximizes the next expected return. Since the underlying transition dynamics can not be known due to very high dimensional observations of the environment, another state-action or quality function $Q(s, a)$ defines the value of taking action a in the state s under the policy π. The quality function is provided with an initial action unlike the state-value function. The policy proceeds from only the current succeeded state [33].

$$Q^{\pi}(s, a) = E[R|s, a, \pi] \tag{2}$$

To learn $Q^{\pi}(s, a)$, MDP is used and the function is defined as Bellman equation. The equation has a recursive form which emphasizes the value of the current state is related with successive states so that Q^{π} can be used to improve the future estimates. This leads us to well known Q-learning and State Action Reward State Action (SARSA) algorithms. While

SARSA is an on-policy algorithm where the behavioral policy and target policy are same, Q-learning is an off-policy algorithm in which those two are different.

2.2.1..2 Policy Search: These types of methods do not need to evaluate a value function model. It starts with random policy and it tries to find the value function of that policy at the policy evaluation step. In the policy improvement state, it will find a new policy using the previous value function. These two steps repeat until the convergence. With the value function and policy search, value based and policy based reinforcement learning show up. The key difference is while value based methods do not need to explicitly build a policy function, policy based methods store representation of policy and keeps it while learning.

Policy based algorithms use policy gradients to get learning signal along with REINFORCE algorithm given in Algorithm 1. A basic trajectory $\tau = (s_1, a_1, s_2, a_2, s_3, a_3, ..., s_t, a_t)$ shows state/action pairs. To compute the expected return, an average plausible trajectories retrieved by the current policy parametrization is needed to be calculated. This procedure requires stochastic approximation in model-free approach using Monte Carlo roll out to compute the rewards. However, since the gradient based learning has difficulties under the stochastic policy due to one small change can completely alter the results, the REINFORCE algorithm is used as an estimator of gradient. Formally, REINFORCE algorithm can be used to compute the expectation over all the possible paths that agent can take and using the computation it estimates of action values to evaluate the policy. The REINFORCE algorithm performs trajectory roll-out using with the current policy. Then log probabilities with rewards are stored. Following, discounted future reward is estimated. Finally, the policy gradient is computed and the policy parameter is updated. These steps are followed in a loop.

Algorithm 1 REINFORCE algorithm.

```
1: initialize policy paramater $\theta$ arbitrarily
2: for each episode $\{s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
3:     for $t = 1$ to $t = T - 1$ do
4:         $\theta \leftarrow \theta + \alpha\Delta_\theta log\pi_\theta(s_t, a_t)v_t$
5: return $\theta$
```

Value based methods require total coverage of state space and function approximation to the value function. However, in high-dimensional state space a small change in policy may cause a major change in the value function. This results in unstable behaviour. On the other hand, policy search methods generally find local optima and work well with continuous

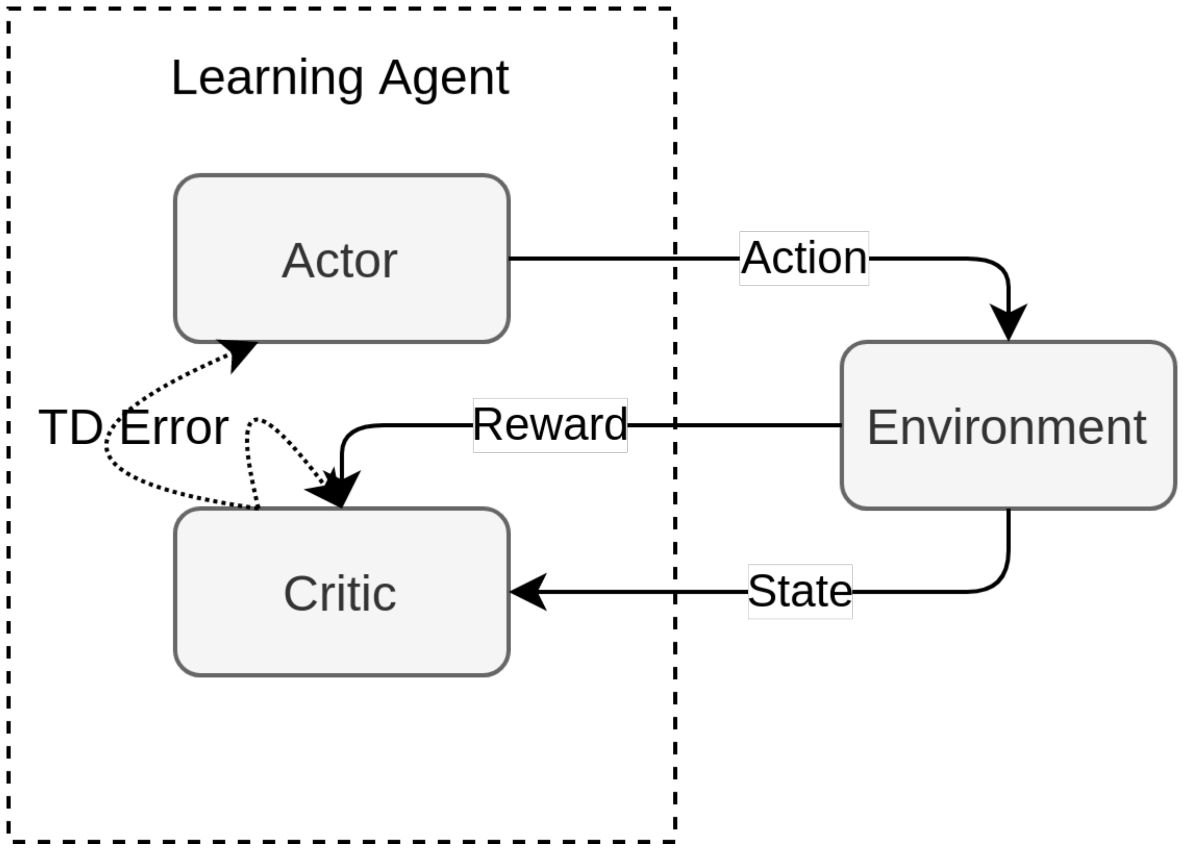


Figure 2.4. Actor-Critic reinforcement learning model.

features. Since it improves the current policy, it gently improves the performance. While value iteration based methods are called critic-only methods generally, policy search methods are called actor-only methods [34].

2.2.1..3 Actor-Critics: In actor-critic methods an explicit policy is evaluated. At the same time, a value function is also used to not to select actions but to decide when the policy needs to be updated. The value function observes the consequences of the actor and then, the policy evaluation decision is made. Fig. 2.4. shows the structure of the actor-critic reinforcement learning model. Temporal Difference (TD) error represent how different the new value from the old prediction. Using this feedback, the policy is updated and the learning process is accelerated. The policy gradient based actor-critic algorithms are more common and they have generally good convergence properties [35].

2.2.1..4 Natural Policy Gradients: Policy gradients constitute the largest part of the actor-critic algorithms. The policy gradients are divided into standard and natural policy gradients [35]. Standard policy gradient fits to the cost function with a single local minima and whose gradients are isotropic in magnitude with respect to any direction away from this minimum [36]. However, real life problems involves multiple local minimums and in those the cost function is not relied on the choice of coordinate system instead they may have curved state space. Standard policy gradient fails in the curved state space so that the natural policy gradient comes into prominence [37]. The natural policy gradients guarantee performance improvement unlike greedy policy iteration [38].

2.2.1..5 Trust Region Policy Optimization(TRPO): Trust Region Policy Optimization is an on-policy model-free actor-critic scalable algorithm for optimizing the policy by using natural policy gradient. It works for both continuous and discrete action space. It updates the policy by taking largest step possible to improve the performance calculating how the distance between the old and new policies using Kullback–Leibler divergence which is measure of distance between probability distributions. Thus, it guarantees monotonic improvement [39]. In this work, TRPO method is used in DRL model, how it is used will be mentioned later in Chapter 4..

2.2.2. Deep Reinforcement Learning

Deep learning combined with RL is used for scaling decision making problems that were unable to be solved due to high dimensions of state and action space [40]. The pioneer DRL method was used to play Atari from directly the pixels of the game [41]. Later, the very famous AlphaGo system based on DRL has defeated a human world champion at Go [42]. Following, DRL has been applied to so many problems from finance to robotics [43, 44].

The key difference DRL from RL is that deep neural networks are used to approximate to components of RL in DRL [45]. Fig. 2.5. shows the symbolic representation of a DRL schematic. The deep neural networks in the agent may be in any component of RL. For example, it can be used in value function, policy or even in the state to action transition model.

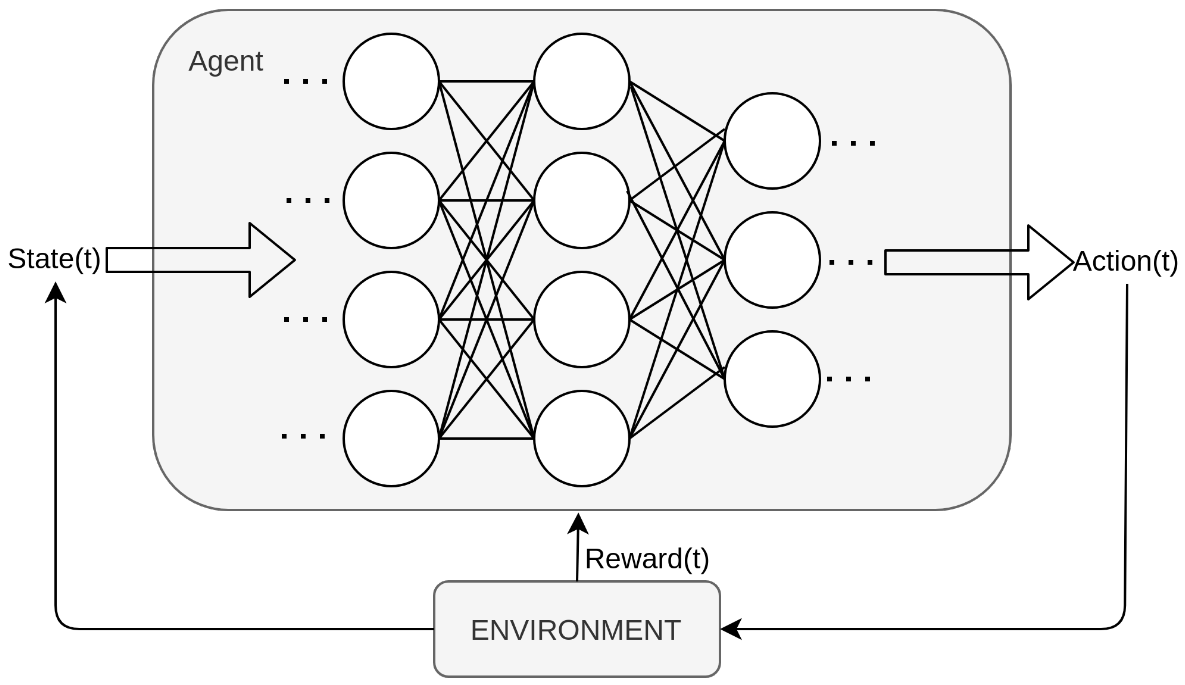


Figure 2.5. Deep reinforcement learning structure.

2.3. Genetic Algorithm

Genetic Algorithms (GA) are a subset of Evolutionary Computation algorithms which are basically the family of all algorithms based on natural evolution. Other than GA, there are ant colony optimization, genetic programming, particle swarm optimization algorithms and etc. under the class of Evolutionary Computation algorithms. GA is specifically a search based optimization technique which uses the natural selection mechanism in the evolution [46]. Some definitions related to GA are given below:

- **Gene**s are the basic parameters that form individuals (chromosomes). The individuals are represented by strings. Generally binary strings that consists of 0s and 1s are employed.
- **Individual or Chromosome** represents a candidate solution for the problem at hand.
- **Population** represents group of all individuals.
- **Fitness function** determines how fit an individual for the problem.
- **Crossover** is the process of exchanging the genes between the chromosomes randomly. After offsprings are created and added to the new population.

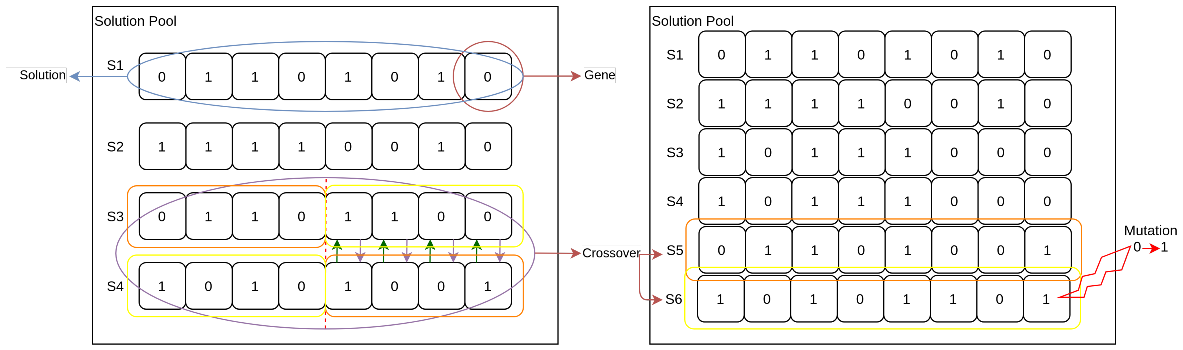


Figure 2.6. The relationship between the operators and individuals in GA.

- **Mutation** is the process of randomly mutating some of genes in the chromosomes.

Algorithm 2 shows the pseudo code of the classical GA. The input to the GA is the candidate solutions for the problem at hand. These solutions, which form the first population are usually initialized randomly at the beginning of the algorithm. The possible solutions are represented as binary strings as shown in Fig 2.6.. The solution should be represented by this kind of fixed size strings in the GA algorithm. They are called artificial chromosome whose genes are associated with each of variables of the problem [47]. Then, the fitness function is employed to evaluate each solution in the population. The algorithm selects the parent solutions based on their fitness values in order apply genetic operators such as selection, crossover and mutation, and so new offsprings are created. For a given problem, having a well defined fitness criteria is very important. Since the natural selection is based on fitness function, if it is poorly designed the individuals that may generate higher fitness in a well constructed architecture may be eliminated. The convergence may take a long time or even GA may not find global optimum. Fig. 2.6. shows how operators work on individuals in generations. Newly created individuals are included into the new population by using replacement strategies for the next generations. The algorithm continues until it reaches to the maximum number of generations or until it reaches to a satisfied solution. As shown in Algorithm 2, the parameters such as population size, maximum iteration number, mutation or crossover probability are set at the beginning of the algorithm.

The main genetic operators in GA, which are selection, crossover and mutation, are explained in details below.

Algorithm 2 Genetic algorithm.

```
1: START
2: Set initial parameters
3: Choose an encode method for individuals
4: Generate the initial population
5: while Iteration $< MAX\_ITERATION$ and Best Fitness $< MAX\_FITNESS$ do
6:     Compute fitness
7:     Selection
8:     Crossover
9:     Mutation
10: return Best Individual
```

2.3.1. Selection

Selection is the process of selecting parents which reproduce and create offsprings for the next generation. It is also a crucial part of the GA since better parents drive the individuals to generate higher fitness results. There are several different methods to select the parents like elitist, roulette wheel, tournament etc. For instance, in one of the most used methods, in the roulette wheel selection, a circular wheel is created and divided proportionally to the fitness results of each individual. Since the individual with highest fitness gets larger pie on the wheel it is more likely to be selected for the reproduction.

2.3.2. Crossover

Once the parents have been selected, they are used to produce the offsprings in the crossover phase. There are too many different crossover methods in GA. The most common one is single point crossover in which a crossover point is selected, then the right or left side of the point is swapped between the parents. In Fig. 2.6. the single point crossover is applied. In the next generation, if those individuals are kept in the new population, both S5 and S6 offsprings have genetic information from their parents.

2.3.3. Mutation

Following the selection and crossover operators, a new population including new members is created. Since the population size must to be fixed, some replacement strategies are employed. Depending on the strategy used, new population might include some copied individuals from the previous population along with new offsprings. To provide additional genetic diversity and prevent to get stuck in a local maximum mutation operation is applied on genes with a probability. Basically, the string representing the gene is replaced with a newly generated string.

2.3.4. NEAT Algorithm

In this study, Neuro-Evolution of Augmenting Topologies (NEAT) algorithm is applied in order to integrate GA with DRL. NEAT is a GA used in evolving and creating neural networks. There are numerous works that use this algorithm such as flappy bird, cartpole balancing games [48, 49]. So that, NEAT has proved itself by training ANN models for dynamic environments. In this work, as a second approach NEAT based model is used. How it is used for the problem at hand will be explained in details later in Chapter 4.. Here, only an introduction to NEAT is given.

NEAT is a GA for evolving and altering ANN that is introduced in [50]. The main aim of the algorithm is to find effective connection weights, biases and even sometimes ANN topology for the problem. Each genome consists of two types of genes which defines ANN. Each neuron is represented with node gene and each connection is represented with connection gene [51]. To evolve the model user provides a fitness function which generates a single real number. NEAT evolves ANNs over generations and creates more complex and better ANN architectures for the problem in time.

Since the genome itself is an ANN in NEAT, reproduction and mutation operations are a bit more complex. For example, the results of these operations may add or delete new ANN nodes and connections to architecture. The crossover is applied on two networks. NEAT keeps track of each origin of node with an identifier. The nodes having common ancestors are matched up for crossover or the connections are matched up for crossover if the nodes connected by that connection have same ancestor.

3. RELATED WORK

In this section, studies on RACH congestion control, RL in wireless networks and RAN slicing in the literature are reviewed.

3.1. RACH Congestion Control

The most well known proposed congestion control technique, ACB, prevents UE to access RACH resources when there is congestion in the network. The BS periodically distributes a probability factor and barring time. Then, the UEs select a random number. If the selected random number is lower than the probability factor, the UE is permitted to access RACH resources. Otherwise, UE waits a random amount of time based on the barring time distributed by the BS.

Although some ACB methods separate the service classes into two as delay-tolerant and delay-constrained for service prioritization, it doesn't fulfil 5G network requirements which may define more than 2 RAN service types [52]. For instance, [11] exemplifies 5 different service types for 5G RAN. To support multiple services, Extended Access Barring (EAB) is proposed as an extension of ACB for service differentiation among different M2M devices. In this scheme, the main aim is to reduce the number of collisions among delay-tolerant M2M devices, so generally the delay is higher than ACB [53]. There are also few studies to incorporate service differentiation into random access channel by distributing multiple barring factors to different service classes in combination with different techniques [54–56].

Back-off adjustment schemes proposed by 3GPP works well in low congestion circumstances [57]. There are two main approaches in back-off adjustment schemes, the first one assigns different back-off timers to different M2M devices and prioritizes them [58, 59] and the second one increases the back-off time interval each time a collision occurs [60]. While the first method aims to achieve different QoS for different service types, the second method aims to prevent collisions in RACH. It is reported that back-off adjustment schemes cannot prevent high average access delay for peak congestion scenarios [61]. Authors in [62] propose a dynamic backoff collision resolution method for massive M2M communication by adjusting backoff indicator which is a parameter indicating the time delay between a PRACH and the next PRACH. In another dynamic backoff collision method, collaborative distributed

Q-learning is used to utilize RACH congestion level for IoT devices [63]. In the work, each device has Q-values corresponding to RA slots and these values are updated based on transmission results.

Slotted random access is another mechanism which allows M2M devices to access RACH in a reserved time slots [64]. The reserved random access slot can be used by only same classified M2M devices. However, if the class has multiple devices, since the access procedure is random, collisions in these scheme reduces the overall performance of the mechanism. Therefore, pre-allocation of resources to each device increases the success rate dramatically. Also, estimating the number of arrivals and optimizing the time slots with respect to estimation is quite successful in peak congestion scenario [65].

Clustering the M2M devices and assigning a head to these groups who interacts with the BS is another suggested mechanism. These groups can be formed with respect to different concerns like geographic proximity, functionality, intended purpose of use. M2M devices don't send their preamble allocation requests directly to BS instead they send their data to the head M2M node. The head node send the buffered information through the BS. The environment changes and information sent by BS is broadcast by the head node to other M2M devices [66].

There are also a limited number of studies proposing to prioritize RACH access through preamble separation [56, 67–69]. Most of these studies propose a fixed preamble separation configuration which divides the preambles into two groups. For example, the total set of preambles are divided into two groups in [67, 68], one group is exclusively reserved for H2H devices and the other group is either reserved for M2M or can be used by both M2M and H2H devices. The authors in [69] proposed to partition the preambles into 3 groups by introducing a higher priority traffic type for smart grid but the groupings are not adaptive. Another static approach where PRACH (Physical Random Access Channel) slots are assigned to different service classes is also proposed in [56]. The issue with these works is that the preamble groupings are static which may be inadequate to respond to changes in the traffic of different service classes.

To overcome this problem, the use of dynamic preamble subset allocation is proposed in [11]. The main difference between the approach in [11] and this work is that the authors use heuristic algorithms for subset allocation whereas we use reinforcement learning. Besides they consider a setup where nodes transmit multiple consecutive preambles whereas we consider single preamble transmission as in the LTE standard. Another dynamic preamble separation method which is combined with binary exponential back-off with respect to three different priority classes is proposed in [70]. A load adaptive dynamic preamble allocation method called LATMAPA [71] is proposed for prioritizing in the context of 5G Random Access. However, it proposes a solution for only 2 slices which are delay tolerant and delay intolerant.

Very recently, an online control method for dynamic preamble distribution over prioritized preamble groups is proposed in [1]. At first, the number of active devices in each priority is estimated in a recursive Bayesian way. Then, together with this estimation, a heuristic novel algorithm that distributes the preambles over the service groups with respect to their priority levels in a dynamic manner is applied. Finally, they extend their approach with ACB. In terms of separating the preambles and prioritizing the services, [1] shows similarity to the proposed approach in this current study. For example, while they classify the devices into different priorities by assigning priority weights, priority coefficients are employed in this study. Furthermore, both study support any number of services. Therefore, the proposed approach is compared with this approach by using a scenario given in [1]. Moreover, the *ideal algorithm* given as the baseline in [1] is used in the experiments of this thesis. Another adaptive method proposes to allocate a dynamic number of preambles to contention-based RACH [72]. In this study, the number of preambles is varied instead of using a fixed number of preambles but different service classes are not taken into consideration.

Code-word expanded random access scheme is a distinctive collision decreasing method without affecting the physical layer [73]. The study uses more than one sub-frames to send preambles and by combining these multi preambles, a distinguishing code-word is created. Doing so, the contention space has been expanded to code domain and the probability for collision drastically decreases. However, it is denoted in the research during the low load conditions the method affects badly to performance because of code ambiguity.

3.2. RL in Wireless Networks

RL-based approaches have recently been proposed for several problems in RAN. In general, these works are focused on optimizing the allocation of resources and increasing the QoS. One relevant study on RACH proposes ACB barring rate adaptation using Q-learning to increase the success probability of M2M communications with low impact on H2H communication [74]. Another study employs RL to optimize the joint allocation of fiber and radio resources for Cloud RANs. The authors note that RL improves performance with respect to genetic and tabu search algorithms [75].

There are also several studies about mobile edge computing (MEC) to optimize the allocation of network and computing resources [76–78]. Another application of RL is to improve the energy efficiency of heterogeneous networks through user scheduling and resource allocation [79]. A Q-learning based DRL is used for dynamic resource optimization for both communication and computational down-link resources like CPU usage [80]. A multi-agent RL method is used to assign distributed cooperative sub-channels and control power to improve transmission success probability in massive access scenario[81].

A RL-based slice admission controller which makes its decisions based on resource availability is also proposed in [82]. A DRL-based method, specifically a DQL-based radio resource management and priority based CN slicing method is proposed in [83]. The authors specify that rather than using demand-prediction methods, demand-aware DRL methods could implicitly incorporate more deep relationship with demand and supply of limited resources. Similarly, there are several DRL based resource allocation methods using network slicing concept in very recent [84–86].

A dynamic resource scheduling method uses DRL to extract user experience related data while user interacting with network environment to understand the user behavior [87]. Using extracted information, dynamic adjustment of the resource to the slices are enabled. CPU usage, storage, bandwidth, and the like are named as network resources and quantified with resource units. These units are assigned to VNFs in the experiments. Authors in [88] first introduce the problem of capacity management in heterogeneous satellite network which is composed of LEO, MEO and GEO satellites, then they construct a network model and a low-complexity method for calculating the capacity between the satellites in this model. Finally, based on Q-learning, they propose a long-term capacity optimal allocation method to optimize long-term utility.

To the best of my knowledge, RL has not been applied to the problem of preamble allocation for network slicing. The advantage of using RL with respect to methods based on heuristic methods is that there is no assumption on the traffic model in RL. RL can learn to maximize channel performance regardless of the traffic distribution.

3.3. RAN Slicing

Network slicing is also gaining popularity as a key enabler technology for the 5G vision and RAN slicing is one of its main components. The main aim of RAN slicing is to enable dynamic on-demand allocation of radio resources among multiple services. A run time slicing method that isolates RAN slices and a set of algorithms in order to partition inter-slice resources are proposed in [89]. Another RAN slicing method allocates radio resources between enhanced mobile broadband and vehicle-to-everything services using RL combined with a heuristic algorithm to maximize utilization [90]. A two-layer scheduler approach is proposed in [91] to manage the isolation and efficiency balance where the first layer is used to allocate resources to each slice and the second one is used to allocate resources for each user. A dynamic up-link resource allocation scheme for RAN slicing is proposed in [92] in order to improve service quality of eMBB service and to reduce power consumption of URLLC service using Lyapunov optimization.

Authors in [93] proposes service provisioning in RAN slicing using a unified framework for access control and bandwidth allocation with the aim of maximized resource utilization and guaranteed QoS. Another study focuses on spectral efficiency on RAN slicing [94]. The authors propose Powell–Hestenes–Rockafellar method in order to allocate resources for two slices (URLLC and eMBB). In [95], a RAN slice selection and optimization method based on transmission rate, blocking rate and delay is proposed. The optimization problem is proved to be a 0-1 knapsack problem.

DRL is also widely used in RAN slicing optimization. One example formulates RAN slicing as joint optimization problem with content caching and mode selection [96]. The dilemma between limited resources and user demands drives authors to use DRL in cloud server in order to make decisions on content caching and mode selection to maximize the reward under the dynamic environment. Another RAN slicing optimization method proposed in [97] uses DRL for intelligent hand-off policy since authors formulates hand-off problem as the Markov decision process. An efficient device association for the RAN slicing method uses hybrid federated DRL to improve network throughput while reducing hand-off cost [98]. The individual devices use federated DRL since transferring data or models from an agent to another agent is not allowed due to privacy of data [99].

A Q-learning method proposes a dynamic resource optimization solution for RAN slicing in [100]. Each slicing request remarks latency, rate, and CPU usage requirements. The method assigns both communication (frequency-time blocks) and computational resources (CPU usage) to these slicing requests for down-link communications. Authors in [101] proposes a DRL approach for multi-tenant cross-slice resource orchestration for RAN slicing. They formulates the problem as stochastic game between service providers (tenants), who are trying to maximize long-term pay-off from the competition with other service providers. The service providers offer their users communication and computation slices over same RAN infrastructure. The DRL is proposed to find optimal abstract policies between the service providers.

Fog-RAN has been proposed to assure the requirements of URLLC. Since the URLLC devices can not accommodate large delays, fog node is invented to be deployed to physically near places to these device and it is equipped with computing, processing and storage capabilities for immediate need. The authors in [102] formulates Fog-RAN resource allocation as the Markov decision process and employ various RL methods. They claim that RL methods always achieve the best results.

Another Q-learning based method proposes resource allocation scheme for C-RAN slicing [103]. The authors create a two-layer framework which is divided into upper and lower layer. In the upper layer, the mapping of virtual protocol stack functions is applied; in the lower layer, RRH association, subchannel and power is managed. The DRL is also proved itself in real-time resource allocation of RAN slicing in [104].

3.4. GA in Wireless Networks

There are a few proposed GA based resource optimization and allocation methods under the concept of network slicing. One study combines Q-learning with GA [105]. While they use Q-learning in order to control cross-slice congestion, for admission strategies they use GA as complementary method to Q-learning [105]. While admission refers issuing new slice requests and handling the resource allocations to the new slice, cross-slice control refers the maintenance of newly created slice until it is released. Radio resource isolation is one of the most important QoS property that must be maintained when any change occurs in the slices. Inter-cell interference is a dominating factor that breaks the isolation rule.

A novel approach used GA for finding dynamic spectrum allocation for slices in order to perform radio resource isolation and utilization [106]. They represent the resource allocation schemes of the cells as genes. A very similar online GA slicing method [107] proposes a strategy optimizer in order to maximize the network utility in long-term. The method encodes slicing strategies into binary sequences for GA to handle request-decision mechanism. Authors of [108] proposes a virtual network resource allocation method named GSO-RBFDM which is based on dynamic resource pricing idea. They chain Group Search Optimization, Radial Basis Function and GA algorithms in order to form a dynamic solution. Instead of the classical Dijkstra Algorithm, GA is used to find low-cost network mapping.

3.5. Discussion

Generally, optimization studies other than allocation resources for RACH focused on initiating and maintaining the network resources for network slices. On the other hand, unlike down-link resource allocation, dynamic allocation of up-link resources for the RACH preamble is a more challenging problem as the BS has limited information about the number of nodes waiting for different services. Previous studies on service differentiation in the RACH context either focused on heuristics methods or analytical models. This study focus on DRL and GA based approaches for this complex and dynamic problem.

DRL methods have the advantage of being able to operate without exact system models. Moreover, they have the ability to adapt to changes in an environment and to give a suitable action automatically with the help of their reward mechanism. These features make DRL very attractive for dynamic preamble allocation and suitable for changing service requirements over time. GA based model is used in order to compare the results with DRL and baseline. Therefore, in this study, the use of DRL and GA for dynamic RAN slicing is investigated.

4. MODEL

This chapter presents the proposed adaptive preamble subset allocation methods based on DRL and GA in details. The both methods can distribute preambles among different service classes according to their priority providing a virtual isolation of service classes. Besides, several reward functions for the DRL algorithm are proposed and the behavior of these functions are mathematically analyzed. This chapter also introduces these reward functions.

In a typical LTE-A and 5G configuration, 64 preambles are used in the random access procedure, however, while 10 of them are reserved for contention-free access, 54 preambles are allocated for contention-based access [109]. Hence, it is assumed that the total number of available preambles is 54. A system is considered as there are N slices with different service requirements with an ideal channel where the preamble losses only occur due to collisions. Therefore, 54 preambles used in the random access procedure are divided into N different groups. Each preamble group is assigned to a slice and a UE is only allowed to select a preamble from the group of its service class. Hence each slice is isolated from the rest of the slices in the sense that the preamble transmitted by a UE can only collide with the other UEs in the same slice.

In the next RAO, the nodes which have been collided will remain backlogged in the system and will attempt to transmit a new preamble in the next round. The preamble groupings are adaptive and assumed to be announced by the BS before each RAO. Hence, the backlogged nodes will follow the new preamble groupings announced by the BS in the next round. As well as the backlogged nodes, newly arriving nodes will also be attempting to transmit in the next slot. A sample RAO is shown in Fig. 4.1.. In the figure, the BS has announced that 9 preambles reserved for $1st$ slice, 12 preambles are reserved for $2nd$ slice, etc. There were 22 UEs transmitted a preamble from $1st$ slice and four of them became successful as they are the ones that did not experience a preamble collision. Hence, the number of backlogged UEs are reduced to 18 after the RACH opportunity. In this thesis, the maximum number of preamble retransmissions (W) by a single node is set to 10. The notations used throughout in this thesis are listed in Table 4.1..

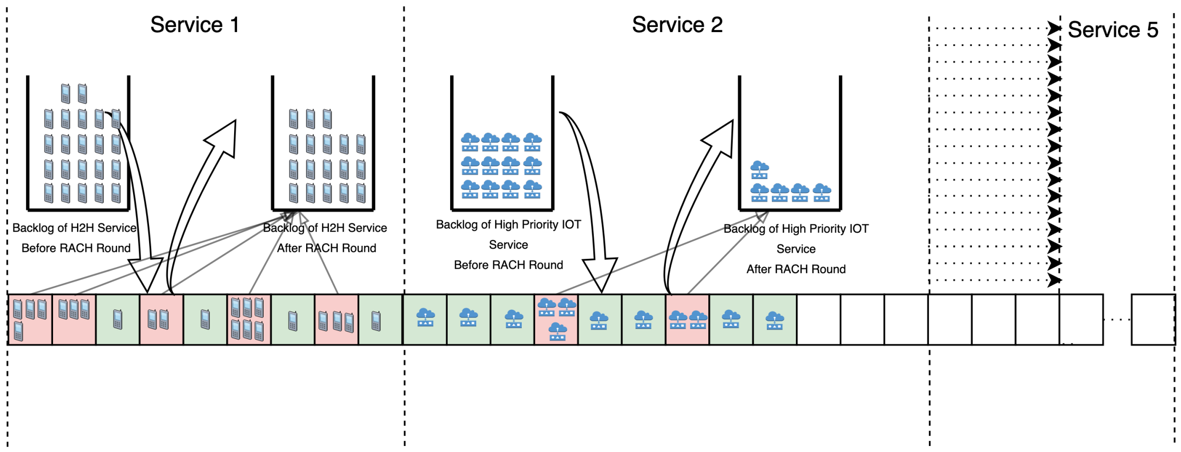

Service 1
Service 2
Service 5
Backlog of H2H Service Before RACH Round
Backlog of H2H Service After RACH Round
Backlog of High Priority IOT Service Before RACH Round
Backlog of High Priority IOT Service After RACH Round

Figure 4.1. RACH process of BS simulator.

Table 4.1. System model notations.

M	Total number of available preambles for contention-based RACH (54)
W	Maximum number of allowed preamble transmission attempt (10)
J	Number of slices(services)
m_j	Number of preambles reserved to slice j
c_j	Number of collided preambles at a RACH opportunity for slice j
s_j	Number of successful preambles at a RACH opportunity for slice j
u_j	Number of unused preambles at a RACH opportunity for slice j
n_j	Number of new arrivals at a RACH opportunity for slice j
λ_j	Normalized arrival rate for slice j
r_j	Collected reward in a RACH period for slice j
k_j	Reward or priority coefficient for successful transmission of preamble for slice j

4.1. DRL-based Model

In recent years, there has been a tremendous increase in the use of DRL in order to solve highly-complex problems. Here, it is used for the adaptive selection of the preamble groups. In this section, the proposed model is explained.

4.1.1. The Model Layout

In RL, an agent learns suitable actions to take in an environment in order to maximize the cumulative reward. One of the main advantages of RL with respect to supervised learning is that it does not require labeled input-output pairs. Hence, the agent learns only by interacting with the environment. In this problem, the learning agent is the BS and the action in RL corresponds to the preamble groupings for each slice. More specifically, the BS decides on the number of preambles for each slice such that their sum will be 54. Before each RAO, the BS distributes the groupings by publishing them.

The environment is the RACH where a number of UEs are waiting to transmit a preamble. Based on the announced preamble groupings, each UE transmits one of the preambles from their respective preamble group. Each node randomly selects its preamble and the number of UEs transmitting is not known to the BS. Moreover, the number of UEs waiting to transmit a preamble changes as new UEs arrive at the channel over time. Hence, the environment is stochastic.

In a classic policy gradient RL implementation, the policy of the agent maps the state to actions and the REINFORCE algorithm optimizes the policy. Here, Trust Region Policy Optimization (TRPO) which has recently achieved state-of-the-art results in various AI tasks is employed in RL [110]. In the TRPO algorithm, the two dependent neural networks shown in Fig. 4.2. are used for approximating the value function and the policy function. From the TRPO perspective, the value network is used to calculate the expected future reward from the current policy. The policy network is used for approximating the policy function. This network gets the current state of the environment as input and produces the probability distributions of each action using the value function, then, the algorithm selects the most probable action [111].

In this thesis, an open implementation of TRPO [112] is used as the DRL framework which uses ADAM optimizer for both neural networks due to its fast convergence [113, 114]. Fig. 4.2. illustrates the interaction between the agent and the simulation environment. Specifically for this problem, the policy network plays the actor role and produces the action space to the environment. Meanwhile, the value network produces state values by assigning each state a score calculated using the sum of rewards and the state of the previous round so that the states with higher rewards have more value in the network. Actions which results in a better state

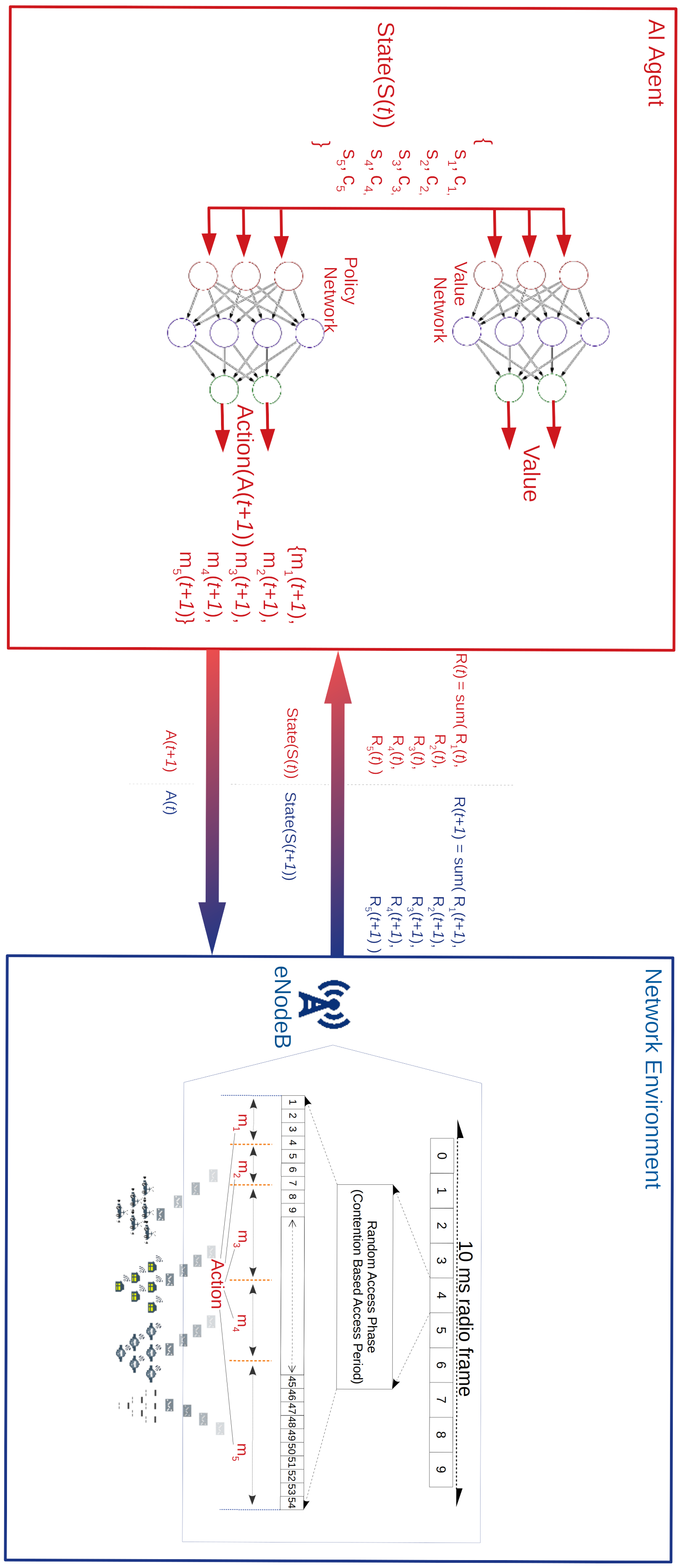

AI Agent
State(S(t))
{ $S_1, C_1,$ $S_2, C_2,$ $S_3, C_3,$ $S_4, C_4,$ S_5, C_5 }
Value Network
Policy Network
Value
Action(A(t+1))
{$m_1(t+1)$, $m_2(t+1)$, $m_3(t+1)$, $m_4(t+1)$, $m_5(t+1)$}
R(t) = sum($R_1(t)$, $R_2(t)$, $R_3(t)$, $R_4(t)$, $R_5(t)$)
State(S(t))
A(t+1)
R(t+1) = sum($R_1(t+1)$, $R_2(t+1)$, $R_3(t+1)$, $R_4(t+1)$, $R_5(t+1)$)
State(S(t+1))
A(t)
Network Environment
eNodeB
10 ms radio frame
0 1 2 3 4 5 6 7 8 9
Random Access Phase
(Contention Based Access Period)
1 2 3 4 5 6 7 8 9
45 46 47 48 49 50 51 52 53 54
m_1
m_2
m_3
m_4
m_5
Action

Figure 4.2. Interaction between DRL agent and network environment when $J = 5$.

is preferred since it produces a higher reward. Using neural networks for the approximations make TRPO a DRL algorithm.

4.1.2. The RL States, Actions and Reward Function

The most crucial aspect in the RL framework is the reward function. The reward function defines the objective of the problem. In this study, the reward is defined as a function of the number of successful preambles and collided preambles for each service class. For each service class j, a reward function is calculated as follows:

$$R_j(t) = s_j * k_j - c_j \tag{3}$$

where s_j and c_j are the number of successful and collided preambles from class j, respectively. k_j is defined as the reward or priority coefficient for class j which is used to prioritize among different slices. Higher the priority of the slice, the higher its reward coefficient should be. Then, the total reward is calculated as the sum of rewards for all service classes.

$$R(t) = \sum_j R_j(t). \tag{4}$$

The selection of this (CRF) reward function is discussed in detail in Sec. 4.3.. The state of the environment is the number of UEs from each service class waiting to transmit a preamble. However, this information is not available to the BS, rather, it uses the most recent observation about the channel in terms of successful and collided preambles. In detail, BS can only know if a preamble is successful or not without knowing how many UEs have sent the same preamble in failure. Than, the state is defined as:

$$S(t) = \bigcup_j S_j(t) \tag{5}$$

where

$$S_j(t) = \{c_j, s_j\}. \tag{6}$$

4.2. NEAT-based Model

In this model, NEAT is used for the adaptive selection of the preamble groups. In this section, the proposed technique is explained.

4.2.1. The Model Layout

An individual corresponds to a candidate ANN model based solution in this model. The ANN model used in this work has as number of input nodes as twice number of slices J. These inputs are number of collided preambles c_j and number of successful preambles s_j for each slice. These were the states $S(t)$ of the DRL agent given in Equation 6. Similarly, the current state of the environment is also given as ANN inputs in this model. Two hidden layer is defined in the configuration since it achieves better performance compared to a single layer [115, 116]. There are as number of slices as output nodes to determine number of preambles reserved to each slice. The detailed information of configurations used for training the NEAT-based model is given in Appendix A. The default configuration file of the cart-pole game [117] is used as the configuration file in this thesis.

Fig. 4.3. illustrates the interaction between the NEAT individual and the simulation environment while training or testing. In the figure, there exists 5 slices; $J = 5$. The NEAT individual provides preamble distribution actions to the network environment, in return, it gets the state of that round as inputs. In each generation, genetic operators are applied to ANN solutions. Firstly, two ANN solutions are selected as parents. By applying crossover, new ANNs which are completely or partially different from their parents are generated. In crossover, the connections, weights or nodes may be transferred between the parents. Mutation operator also works on the same parameters and changes them with a given probability.

4.2.2. The Fitness Function

As the reward function in RL, a well defined fitness function in GA is the most important part of the problem. In GA, the fitness function of an individual measures how well any black box model works in an environment relative to other individuals in a solution pool while each individual aiming the same predetermined objective. It is used by GA as an indicator for survival of that individual. Basically, the individual with a higher fitness value has a higher chance to propagate in the next round. In RL, the reward function is a direct signal for the model to learn since the policy is preserved until the end of training. Still, both fitness function and reward function indicate how well a model performs in an environment [118]. Therefore, the fitness function of GA in this study is same with the reward function of RL. For each service class j, fitness function is calculated as follows:

$$F_j(I,t) = s_j * k_j - c_j \tag{7}$$

where s_j and c_j are the number of successful and collided preambles from class j respectively. k_j is defined as reward coefficient for class j who gives the priority level of slice. Than, the computed fitness for the given individual I at time t for j slices is equal to sum fitnesses of all slices.

$$F(I,t) = \sum_j F_j(I,t) \tag{8}$$

4.3. Reward Function Analysis

In this section, the problem in RL context is formulated and, then, the RL algorithm that is used to solve the problem at hand is described. The reward functions explained in this section are used in GA model as well. Each formulation here also constructs the fitness function used in NEAT.

Reward function is the objective function used for evaluating the model that has to be maximized over successive steps. The choice of the reward function is crucial, since a poor selection may result in suboptimal allocation of resources. In this part, analytical expressions are derived for several possible reward functions for a two-slice scenario, $J = 2$. Then, the behavior of these functions is evaluated. Finally, these analytical results are compared with the experimental results presented in Sec. 5..

Let m_1 and m_2 are the number of preambles assigned to each slice such that $M = m_1 + m_2$. Let n_1 and n_2 are the average number of arrivals per RACH opportunity for each slice. First the case where $n_1 < m_1/e$ and $n_2 < m_2/e$ is considered. In this case, all incoming traffic can be supported because the capacity of the RACH with m preambles is approximated as m/e [119].

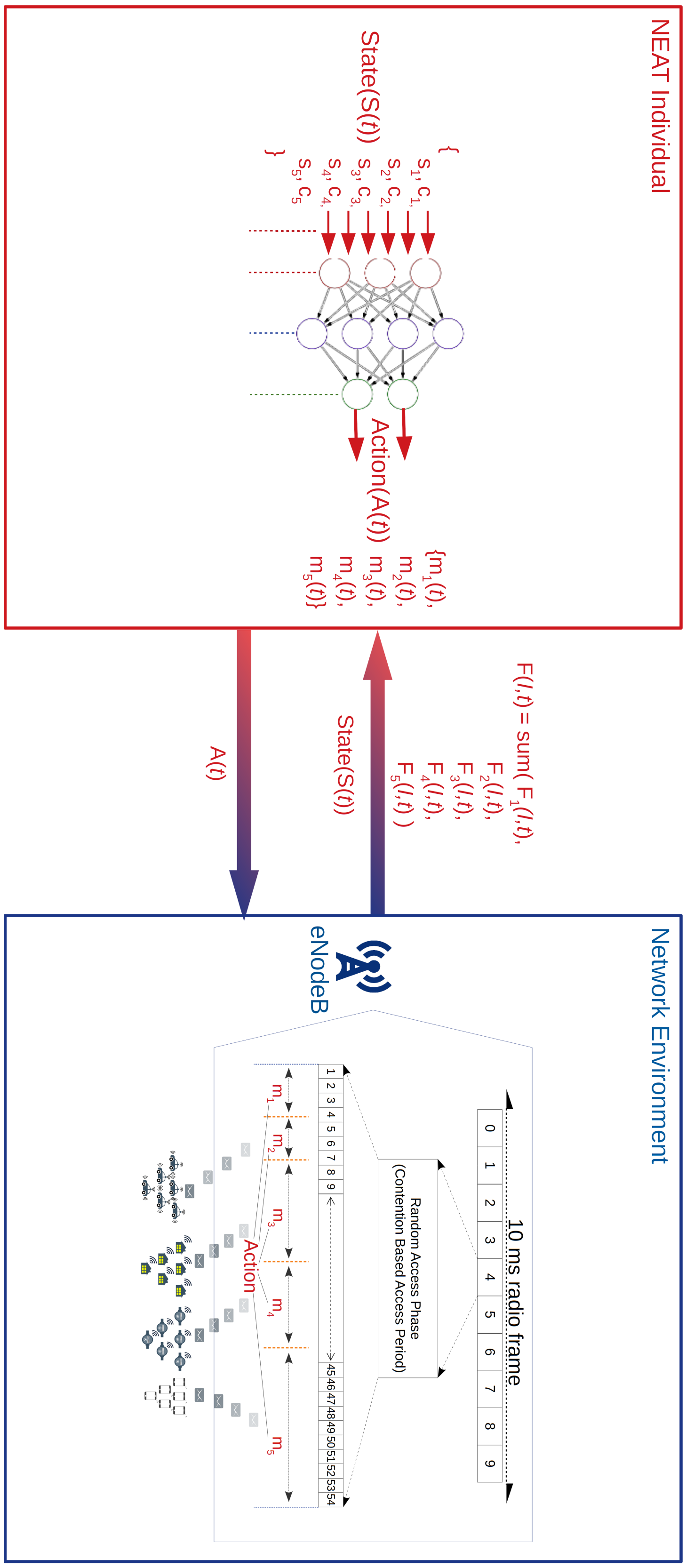

NEAT Individual
State(S(t))
{ s1, c1, s2, c2, s3, c3, s4, c4, s5, c5 }
Action(A(t))
{m1(t), m2(t), m3(t), m4(t), m5(t)}
A(t)
State(S(t))
F(l,t) = sum(F1(l,t), F2(l,t), F3(l,t), F4(l,t), F5(l,t))
Network Environment
eNodeB
10 ms radio frame
0 1 2 3 4 5 6 7 8 9
Random Access Phase (Contention Based Access Period)
1 2 3 4 5 6 7 8 9
45 46 47 48 49 50 51 52 53 54
m1
m2
m3
m4
m5
Action

Figure 4.3. Interaction between NEAT individual and network environment when $J = 5$.

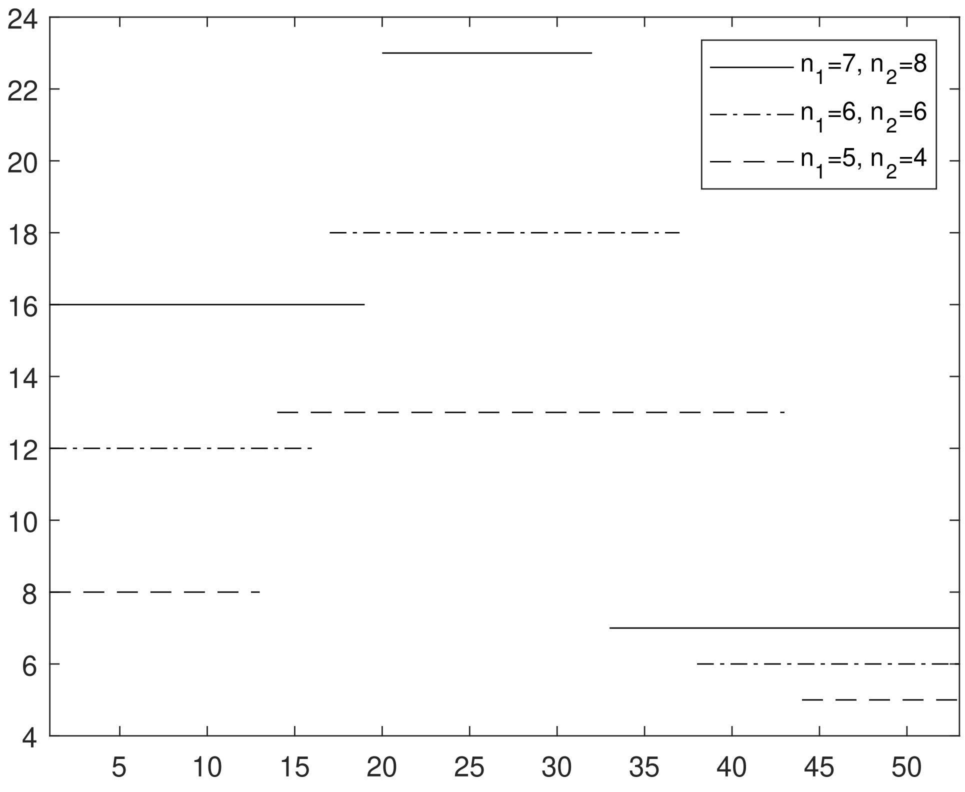


Figure 4.4. Change in the SRF as m_1 changes for $k_1 = 1$ and $k_2 = 2$ for different arrival rates.

When there are b_i nodes randomly selecting preambles from a set with size m_i, the expected number of successful preambles is given by [120]:

$$s_i = b_i(1 - m_i^{-1})^{b_i - 1}. \tag{9}$$

Similarly, the expected number of unused preambles can be found as [120]:

$$u_i = m_i(1 - m_i^{-1})^{b_i}. \tag{10}$$

The remaining preambles are collided:

$$c_i = m_i - s_i - u_i. \tag{11}$$

When the RACH is stable, the expected number of successful preambles must be equal to the average number of arrivals in the long run, i.e., $s_i = n_i$. To satisfy this equality, the number

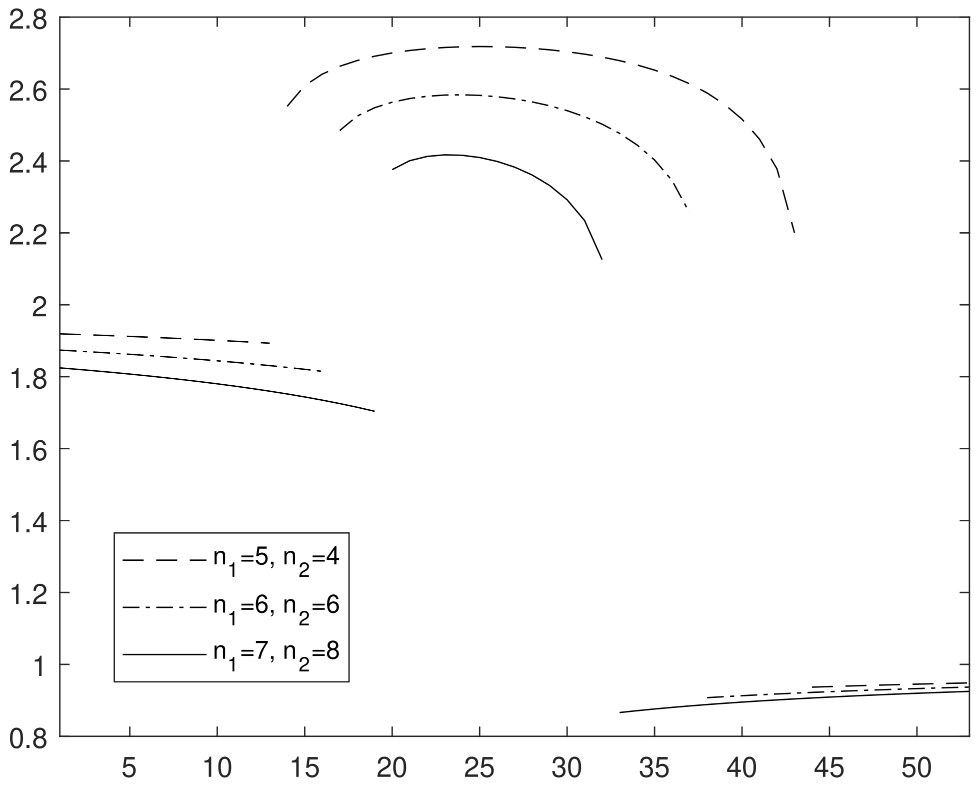


Figure 4.5. Change in the PRF as m_1 changes for $k_1 = 1$ and $k_2 = 2$ for different arrival rates.

of attempting nodes at each slot needs to increase to a level $b_i > n_i$. This value can be found by solving (9) which gives $b_i = -m_i W(-n_i/m_i)$ where $W()$ is the principal branch of the Lambert W function [119]. Using the approximation $(1 - 1/x)^n \approx e^{-n/x}$, u and c can be further simplified as

$$u_i \approx m_i e^{-b_i/m_i} = m_i e^{W(-n_i/m_i)} \tag{12}$$

and

$$c_i \approx m_i(1 - e^{W(-n_i/m_i)}) - n_i. \tag{13}$$

4.3.0..1 Successful preambles reward function *(SRF)* The most intuitive and simple reward function is the number of successfully transmitted preambles at each RACH opportunity. In this case, the reward for a slice is $r_i = s_i$ and the total reward can be defined as the weighted sum of rewards of each slice:

$$r_{succ} = k_1 r_1 + k_2 r_2 = k_1 s_1 + k_2 s_2 \tag{14}$$

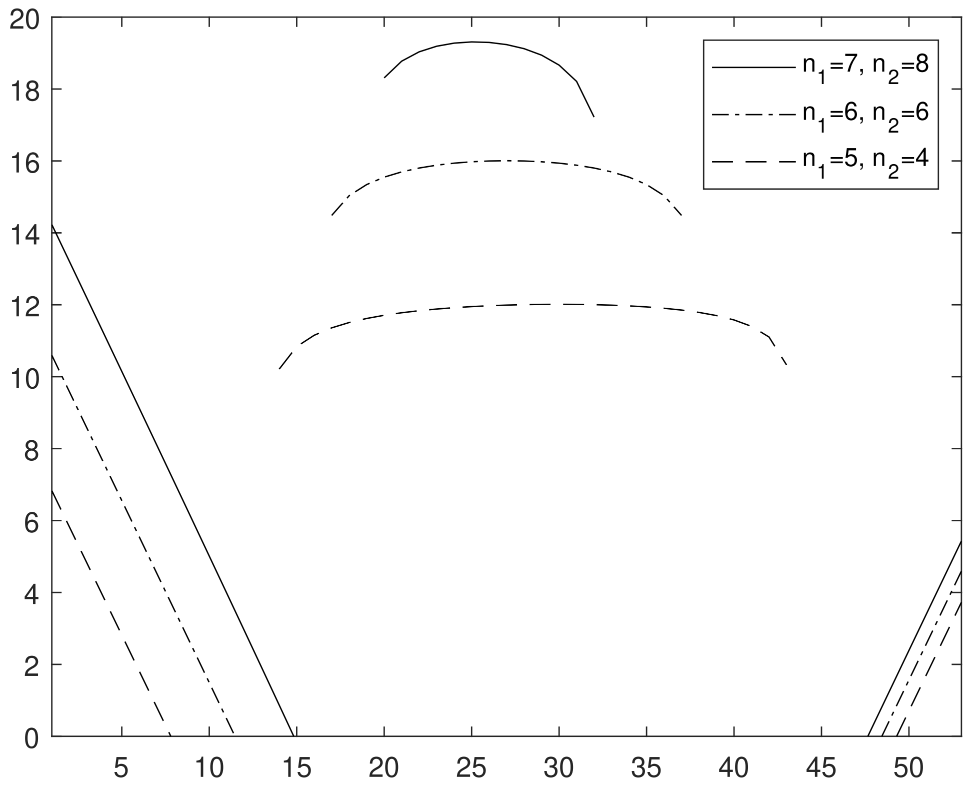


Figure 4.6. Change in the CRF as m_1 changes for $k_1 = 1$ and $k_2 = 2$ for different arrival rates.

where k_1 and k_2 are used to prioritize different slices. This reward function, however, has some undesirable properties. For any choice of m_1 and m_2 which satisfies $n_1 < m_1/e$ and $n_2 < m_2/e$, RACH is stable and the number of successful preambles equal to the number of arrivals, $s_i = n_i$. Hence, the reward function is flat in this region. For $n_1 > m_1/e$ and $n_2 < m_2/e$, all preambles will be unsuccessful for $1st$ slice and all traffic can be supported in $2nd$ slice, again leading to a flat reward function. In general, the reward function can be written as:

$$r_{succ} = \begin{cases} k_1 n_1 + k_2 n_2 & \text{if } n_1 < m_1/e, n_2 < m_2/e \\ k_1 n_1 & \text{if } n_1 < m_1/e, n_2 > m_2/e \\ k_2 n_2 & \text{if } n_1 > m_1/e, n_2 < m_2/e \\ 0 & \text{if } n_1 > m_1/e, n_2 > m_2/e \end{cases} \tag{15}$$

The behavior of this reward function is illustrated in Fig. 4.4. for $n_1 = n_2 = 5$. As the reward function is flat for the stable region, the RL agent does not have any incentive to change the groupings as long as the RACH is stable. This behaviour may bring the system very close to instability when the number of preambles reserved for one slice is barely sufficient for the traffic of that slice. In that case, a slight increase in traffic may result in serious congestion which could have been easily avoided if the agent used a more robust allocation of preambles. Hence, a reward function which would "steer" the system towards a better operating point is needed.

4.3.0..2 Proportional reward function *(PRF)* Another intuitive reward function for the RACH channel is the ratio of successful preambles among the number of transmitted preambles. Weighting each ratio corresponding to each slice, it is possible to differentiate preambles among slices according to their priority. Let k_1 and k_2 denote the priority coefficients of each slice. Then, for $n_1 < m_1/e$ and $n_2 < m_2/e$, total reward for a single RACH opportunity is given by

$$r_{pr} = r_1 + r_2 = k_1 \frac{s_1}{m_1 - u_1} + k_2 \frac{s_2}{m_2 - u_2} \tag{16}$$

For $n_1 > m_1/e$, there are not enough preambles for $1st$ slice which will result in a growing backlog and all preambles will collide resulting in $r_1 = 0$. Similarly, for $n_2 > m_2/e$, $r_2 = 0$. Hence,

$$r_{pr} = \begin{cases} \sum_{i=1}^{2} \frac{k_i n_i}{m_i (1 - e^{W(-n_i/m_i)})} & \text{if } n_1 < m_1/e, n_2 < m_2/e \\ \frac{k_1 n_1}{m_1 (1 - e^{W(-n_1/m_1)})} & \text{if } n_1 < m_1/e, n_2 > m_2/e \\ \frac{k_2 n_2}{m_2 (1 - e^{W(-n_2/m_2)})} & \text{if } n_1 > m_1/e, n_2 < m_2/e \\ 0 & \text{if } n_1 > m_1/e, n_2 > m_2/e \end{cases} \tag{17}$$

For a given n_1, n_2, k_1 and k_2, this function can be numerically maximized to find the values of m_1 and m_2 such that $m_1 + m_2 = m$. The behavior of the *PRF* is shown in Fig. 4.5. for $n_1 = n_2 = 5$. This reward function does not suffer from the problem mentioned in the previous part. The optimum values of m_1 for different priority coefficients is 25, 24 and 23 for $(n_1, n_2) = (5, 4)$, $(6, 6)$ and $(7, 8)$, respectively. Even the traffic of lower priority traffic is higher than the traffic of higher priority traffic, this reward function allocates a lower number of preambles to the lower-priority traffic.

4.3.0..3 Collision-penalizing reward function *(CRF)* Another reward function can be defined as $r_i = s_i k_i - c_i$ where k_i is the reward coefficient. Hence, the total reward for $n_1 < m_1/e$ and $n_2 < m_2/e$ can be written as:

$$
\begin{aligned}
r_{cp} =& r_1 + r_2 && (18)\\
=& k_1 s_1 + k_2 s_2 - c_1 - c_2 && (19)\\
=& (1 + k_1) n_1 - m_1 (1 - e^{W(-n_1/m_1)}) + && (20)\\
& (1 + k_2) n_2 - m_2 (1 - e^{W(-n_2/m_2)}) && (21)
\end{aligned}
$$

For $n_1 > m_1/e$ and $n_2 < m_1/e$, all preambles reserved for $1st$ slice will experience collisions, $c_1 = m_1$ and $s_1 = 0$. Hence, the reward will be

$$
\begin{aligned}
r_{cp} =& k_2 s_2 - m_1 - c_2 && (22)\\
=& (1 + k_2) n_2 - m_1 - && (23)\\
& m_2 (1 - e^{W(-n_2/m_2)}). && (24)
\end{aligned}
$$

Similarly, for $n_2 < m_2/e$:

$$
\begin{aligned}
r_{cp} =& k_1 s_1 - m_2 - c_1 && (25)\\
=& (1 + k_1) n_1 - m_2 - && (26)\\
& m_1 (1 - e^{W(-n_1/m_1)}) && (27)
\end{aligned}
$$

For the case $n_1 > m_1/e$ and $n_2 > m_1/e$, all preambles will be collided:

$$
r_{cp} = -m_1 - m_2. \qquad (28)
$$

The behavior of the *CRF* is shown in Fig. 4.6.. For $k_1 = 1$ and $k_2 = 2$, the optimum values of m_1 for different priority coefficients is 30, 27 and 25 for $(n_1, n_2) = (5, 4)$, $(6, 6)$ and $(7, 8)$, respectively.

5. EXPERIMENTS AND RESULTS

In this chapter, the both proposed techniques are evaluated using simulations. Different models are trained and tested for each reward and fitness function for different number of slices. The models are trained by using only *CRF* and *PRF* since the agent cannot distribute preambles successfully and cannot handle the load of the higher priority slices effectively when *SRF* is used. Therefore, a higher number of dropped preamble requests and longer waiting times are observed in comparison to *CRF* and *PRF*. Hence, only the results of the models trained by using *CRF* and *PRF* are demonstrated in the results. For the NEAT the same reward functions are used as fitness functions as explained in Chapter 4..

In the previous Chapter 4., three different reward functions are analyzed for the 2-slice scenario since the complete state space is relatively small. However, with the context of 5G there are defined at least 3 service types: extreme-mobile-broad-band (xMBB), massive machine-type-communications (mMTC) and ultra-reliable machine-type communications (uMTC) [10]. This number, however, can increase up to 5 as delay-tolerant IoT, emergency, high priority IoT, human-to-human and mobile broadband [11]. On the other hand, increasing the number of slices also increases the state space drastically and finding the optimum policy through analysis gets intractable. In order to prove that both methods propose a valid solution to the problem even in the high number of slices, the simulations are conducted for 3 and 5-slice scenarios besides the 2-slice scenario in this thesis study.

In the simulations, slices are prioritized using their reward coefficients, k_j, also called priority coefficient. Slice numbers are used as the prioritization factor such that $k_j = j$. In other words, assuming there are n slices, the n^{th} slice has the highest priority. This prioritization scheme is employed in all simulations through this work. Therefore, the slice numbers in all figures also indicate their priority. In addition, the first and lowest priority slice will be named as low-priority slice and the n^{th} and highest priority slice will be named as high-priority slice in the rest of the study.

5.1. Traffic Distribution

5.1.1. The Distribution on Increase of Arrival Rate of Only High-priority Slice Scenario

In these simulations, n_j preamble requests are generated every 10 ms and these requests are added to the testing backlogs of each slice. While low priority slices have a constant arrival rate during the whole simulation, the rate of the high-priority slice is increased every 200 ms. In the 2-slice case, low-priority slice has a constant normalized arrival rate of $\lambda_1 \approx 0.09$ where the normalized arrival rate of slice j is defined as $\lambda_j = n_j/M$. The normalized arrival rate for the high-priority slice starts from $\lambda_2 \approx 0.04$ and increases up to $\lambda_2 \approx 0.39$. In the 3-slice case, the low-priority slice has a constant rate of $\lambda_1 \approx 0.07$ and medium-priority slice also has a constant rate of $\lambda_2 \approx 0.09$. The normalized arrival rate of the high-priority slice starts from $\lambda_3 \approx 0.04$ and increases up to $\lambda_3 \approx 0.39$. The x-axis of the Figs. 5.3.-5.33. demonstrate this increase. Since the rate of increase of the high-priority slice is constant, the x-axis is proportional to the simulation time. Each simulation runs 7 seconds.

5.1.2. The Distribution on Dynamic Environment Scenario

In these simulations, n_j preamble requests are generated every 10 ms and these requests are added to the testing backlogs of each slice like above. Differently from the previous distribution, in order to test the temporal behaviour, here not only the requests of the high-priority slice change, all preamble requests to slices may vary in time. The general approach is having sudden increase of the arrival rate from low-priority to high-priority and the sudden decrease from high-priority to low-priority respectively in time.

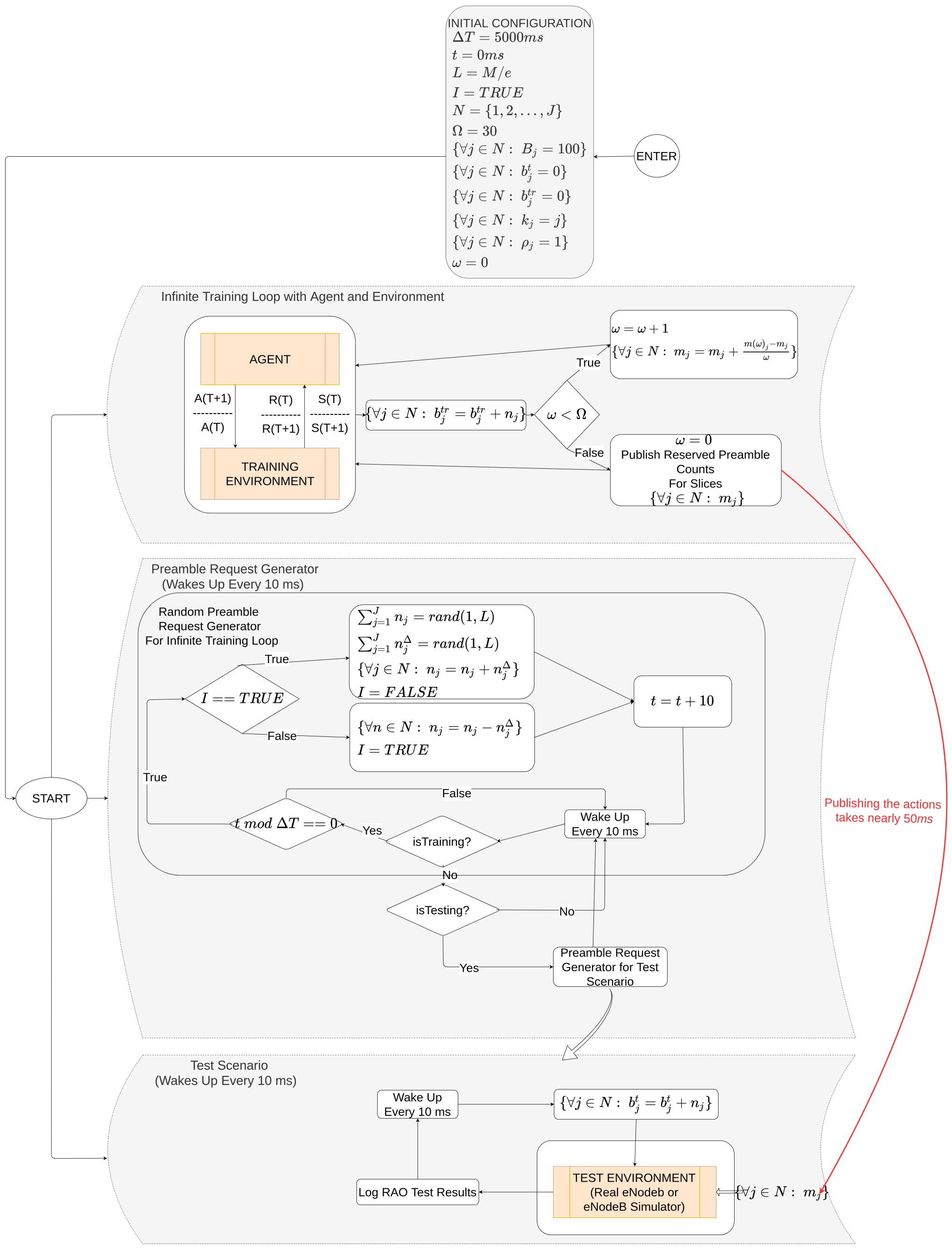
5.2. DRL Training

Here, a scenario where the BS policy is trained online but tested offline with actual traffic after deployment is assumed. RL algorithms start by exploring the action space. At the initial state, the first actions are mostly random and their initial behavior could be very far from optimum. Hence, RL algorithms take a considerable amount of time in order to converge, especially if the training process starts from scratch.

Over-fitting is a crucial problem to avoid in all types of machine learning approaches. RL algorithms are also prone to over-fitting to the environment and the trained policies may not generalize very well to newer environments [121]. In the scenario, the system could over-fit to the traffic arrival distributions used in training. In order to avoid over-fitting, the policy is trained by using a randomly generated traffic model. n_j values are randomly generated in each random access traffic generator period. After the training, the performance of the optimized policy is evaluated on different traffic arrival distributions which is more realistic than random arrivals. Thus, the trained policy comes across first time with the actual traffic pattern in the evaluation phase.

In the experiments, a random number of preamble request generator for slices is created. First, the objective behind this approach should be revealed. In general, number of successful transmissions in a round yields to M/e if the number of arrival of preamble requests is greater than or equal to M [122]. As a result, if the arrival request count is greater than M/e constantly, the number of backlogged preamble request increases in time. At some point, if no other mechanism is not implemented, the preamble allocation process becomes unstable irreversibly.

Considering the above objective, the random preamble request generator generates nearly M/e preambles requests on average. Fig. 5.1. shows the training flow. In detail, the generator wakes up in every 10ms (RAO period) and changes n_j values on every 5th seconds. On each 5th second, the number of arrivals per round for each slice is recalculated with respect to alternating two sub-algorithms. Considering $J = 5$, in the first sub-algorithm, a random number is generated between 1 and M/e and the value of the number is distributed over number of new arrivals such that $\sum_{j=1}^{J} n_j = rand(1, M/e)$. The distribution of that value over those 5 numbers is also random. Later, a second random number is generated again between 1 and M/e that is distributed over change amount values of number of new arrivals, $\sum_{j=1}^{J} n_j^{\Delta} = rand(1, M/e)$. The distribution is again random. Finally, the change amounts are added to the number of arrivals for each slice, $\{\forall j \in \{1..J\} :\ n_j = n_j + n_j^{\Delta}\}$. For the upcoming 5 seconds, generator will generate preamble requests every 10ms and adds these requests to training backlog of each slice. On the next 5th second, the second sub-algorithm steps in, the change amounts from previous 5th second are subtracted from the number of arrivals $\{\forall j \in \{1..J\} :\ n_j = n_j - n_j^{\Delta}\}$. For the upcoming 5 seconds in every 10ms, those subtracted number of arrivals will be added to training backlogs as in before.



INITIAL CONFIGURATION
$\Delta T = 5000ms$
$t = 0ms$
$L = M/e$
$I = TRUE$
$N = \{1, 2, \dots, J\}$
$\Omega = 30$
$\{\forall j \in N: \ B_j = 100\}$
$\{\forall j \in N: \ b_j^t = 0\}$
$\{\forall j \in N: \ b_j^{tr} = 0\}$
$\{\forall j \in N: \ k_j = j\}$
$\{\forall j \in N: \ \rho_j = 1\}$
$\omega = 0$
ENTER
START
Infinite Training Loop with Agent and Environment
AGENT
A(T+1)
A(T)
R(T)
R(T+1)
S(T)
S(T+1)
TRAINING ENVIRONMENT
$\{\forall j \in N: \ b_j^{tr} = b_j^{tr} + n_j\}$
$\omega < \Omega$
True
False
$\omega = \omega + 1$
$\{\forall j \in N: \ m_j = m_j + \frac{m(\omega)_j - m_j}{\omega}\}$
$\omega = 0$
Publish Reserved Preamble Counts For Slices
$\{\forall j \in N: \ m_j\}$
Publishing the actions takes nearly $50ms$
Preamble Request Generator
(Wakes Up Every 10 ms)
Random Preamble Request Generator For Infinite Training Loop
$I == TRUE$
True
False
$\sum_{j=1}^{J} n_j = rand(1, L)$
$\sum_{j=1}^{J} n_j^{\Delta} = rand(1, L)$
$\{\forall j \in N: \ n_j = n_j + n_j^{\Delta}\}$
$I = FALSE$
$\{\forall n \in N: \ n_j = n_j - n_j^{\Delta}\}$
$I = TRUE$
$t = t + 10$
True
$t \ mod \ \Delta T == 0$
False
Yes
isTraining?
Wake Up Every 10 ms
No
isTesting?
No
Yes
Preamble Request Generator for Test Scenario
Test Scenario
(Wakes Up Every 10 ms)
Wake Up Every 10 ms
$\{\forall j \in N: \ b_j^t = b_j^t + n_j\}$
TEST ENVIRONMENT (Real eNodeb or eNodeB Simulator)
$\{\forall j \in N: \ m_j\}$
Log RAO Test Results

Figure 5.1. Training scenario of the DRL-based model.

In each RAO period, these backlogged preamble requests are evaluated by the BS simulator. The request which ends up with success or the same request fails consecutive W times is removed from backlog. This loop continues forever. The infinite training loop with agent and training environment is shown in Fig. 5.1.. In this figure, the notations given in Table 5.1. are used. The arrival request counts of slices are calculated in the preamble request generator thread then added to the training backlogs of slices. On the other hand, in testing environment, these arrival request counts are calculated using preamble request generator and then are added to the testing backlogs in every 10ms, since the testing environment simulates the random access procedure in every 10ms as in the real BS device. In the training environment, these arrival requests are added to the training backlogs in an infinite loop without waiting 10ms and the training environment simulates random access procedure also in an infinite loop without waiting 10ms. By doing so, when Ω is 30 the publishing of action space takes 50ms. Using a lower Ω value would result a shortened publishing action space duration. However, selecting a lower value may induce a lower accuracy also.

The DRL agent and the training environment is interacting constantly. While the agent feeds from the state space and sum reward of the previous round, it produces the action space for the training environment. However, the number of the preambles reserved to slices inside the action space can fluctuate due to very nature of the DRL algorithms. To avoid from that unpredictable behaviour, rather than using an action space generated by the agent at an arbitrary point in the time, moving averages of the number of the preambles reserved to slice are used to smooth out the fluctuations in the action space. Therefore, while using lower Ω values has an advantage of shorter period for publishing the action space, higher Ω values has an advantage of smooth and accurate values in the action space.

The training procedure can be denotable as online-learning since the training data is produced during the training. The training and testing environment is different as shown in Fig. 5.1.. Generally, the testing data is not random and produced/collected with respect to a real life scenario. As a result, although the model can be used for testing or for in real life after training is stopped if the state space is provided by BS, it can also be used for testing while the model is being trained.

Table 5.1. Training model parameters.

ΔT	The time interval to change the training parameters in milliseconds
t	Current time in milliseconds
Ω	Maximum step count to update the policy and publish the action space
ω	Current step count which shouldn't exceed Ω
L	Maximum limit for number of arrivals of preamble requests
b_j^{tr}	Number of backlogged preamble request for slice j used in the training
b_j^{t}	Number of backlogged preamble request for slice j used in the testing
B_j	Number of maximum preamble requests backlogged for slice j
n_j^{Δ}	The change amount of the arrival numbers of preamble requests in every ΔT for slice j

5.3. NEAT Training

On the contrary to DRL, the testing can be done after the training is finished in NEAT. The finite training loop in NEAT is shown in Fig. 5.2.. The random preamble generator part of training is same with the DRL training part. Each NEAT individual is an ANN from the solution pool. The GA operations can be seen in Fig. 5.2. of the NEAT. In each generation, genetic operators are applied to ANN solutions. Firstly, two ANN solutions are selected as parents. By applying crossover, new ANNs which are completely or partially different from their parents are generated. In crossover, the connections, weights or nodes may be transferred between the parents. Mutation operator also works on the same parameters and changes them with a given probability. The algorithm runs up to 2000 number of generations.

After the training is finished, an evolved solution pool containing survivors have left for testing. Using the random number preamble request generator and simulation environment, all possible individuals are tried in each RAO using same environment and producing fitnesses for that environment for 1000 RAO interval. Finally, the best individual which gives highest cumulative fitness value is selected among them.

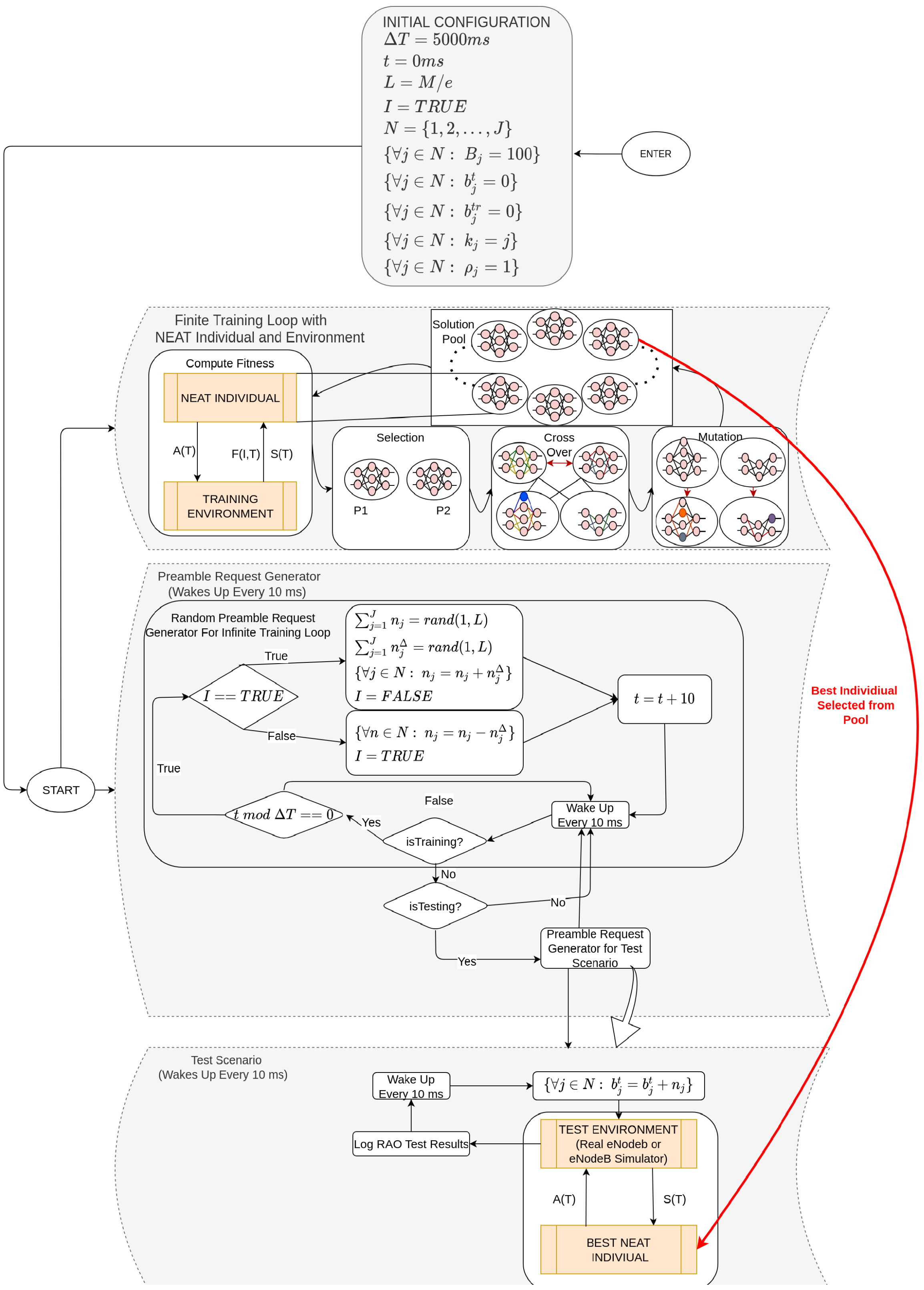

INITIAL CONFIGURATION
$\Delta T = 5000ms$
$t = 0ms$
$L = M/e$
$I = TRUE$
$N = \{1, 2, \ldots, J\}$
$\{\forall j \in N : \ B_j = 100\}$
$\{\forall j \in N : \ b_j^t = 0\}$
$\{\forall j \in N : \ b_j^{tr} = 0\}$
$\{\forall j \in N : \ k_j = j\}$
$\{\forall j \in N : \ \rho_j = 1\}$
ENTER
Finite Training Loop with NEAT Individual and Environment
Solution Pool
Compute Fitness
NEAT INDIVIDUAL
A(T)
F(I,T)
S(T)
TRAINING ENVIRONMENT
Selection
P1
P2
Cross Over
Mutation
Best Individiual Selected from Pool
START
Preamble Request Generator (Wakes Up Every 10 ms)
Random Preamble Request Generator For Infinite Training Loop
$\sum_{j=1}^{J} n_j = rand(1, L)$
$\sum_{j=1}^{J} n_j^{\Delta} = rand(1, L)$
$\{\forall j \in N : \ n_j = n_j + n_j^{\Delta}\}$
$I = FALSE$
True
$I == TRUE$
False
$\{\forall n \in N : \ n_j = n_j - n_j^{\Delta}\}$
$I = TRUE$
$t = t + 10$
True
False
$t \ mod \ \Delta T == 0$
Yes
isTraining?
Wake Up Every 10 ms
No
isTesting?
No
Preamble Request Generator for Test Scenario
Yes
Test Scenario (Wakes Up Every 10 ms)
Wake Up Every 10 ms
$\{\forall j \in N : \ b_j^t = b_j^t + n_j\}$
Log RAO Test Results
TEST ENVIRONMENT (Real eNodeb or eNodeB Simulator)
A(T)
S(T)
BEST NEAT INDIVIUAL

Figure 5.2. Training scenario of the NEAT-based model.

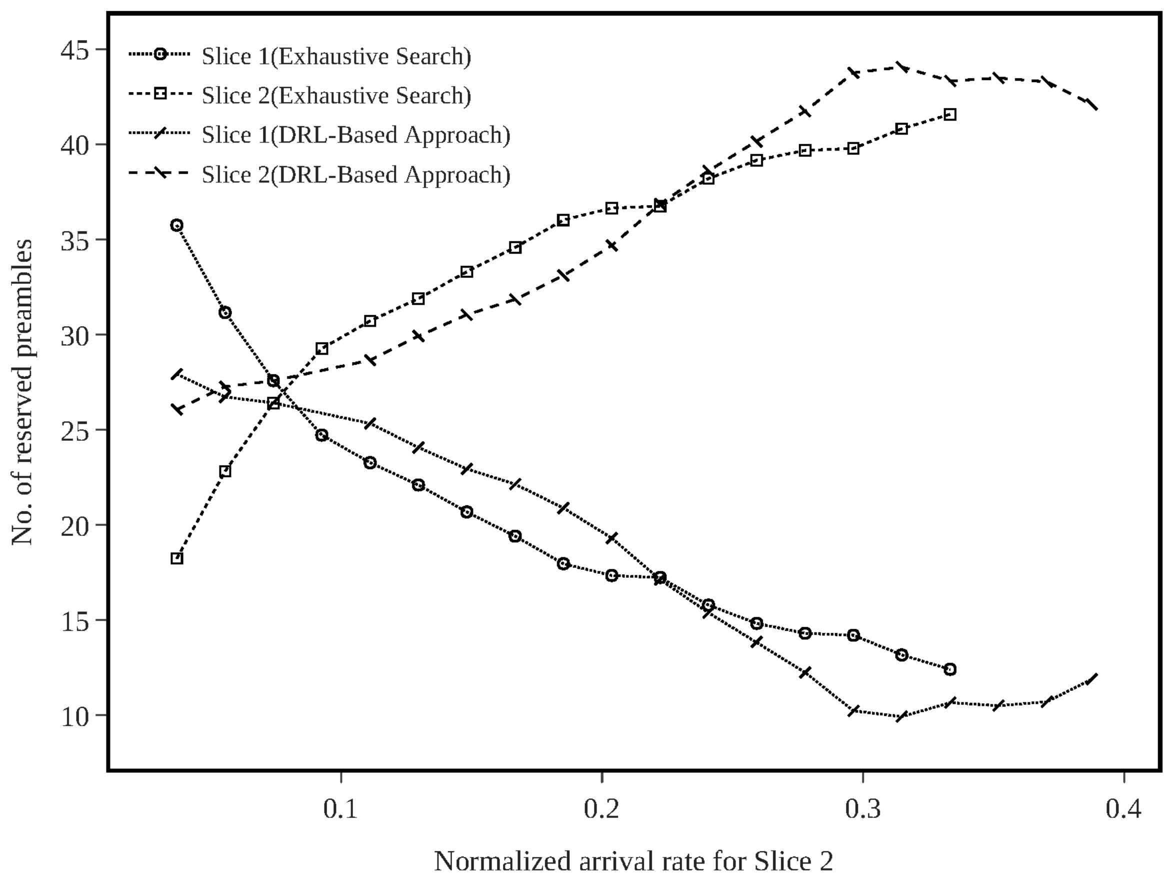


Figure 5.3. Preamble allocations computed by the exhaustive search against the DRL-based approach in 2-slice scenario for the PRF.

5.4. Benchmarks for Simulations

In order to evaluate how the proposed techniques based on DRL and GA get close to the optimal solution, it is compared with an exhaustive search technique in addition to the analyses presented in Sec. 4.3.. In exhaustive search, all possible preamble allocations for a given traffic load are searched through and the best preamble allocation is chosen. The same system parameters are used in the exhaustive search.

The performance of the proposed methods are also compared with an unsliced scenario. In this scenario, there is only one slice in the system and all preamble requests belong to that slice. In such a case, $n_{unsliced}$ is the sum of the number of new arrival of preamble requests to all slices. As a result, the total number of new arrival requests does not change between the unsliced and sliced scenarios. That gives us a comparison basis between the performances of slices of our proposed model and the unsliced scenario.

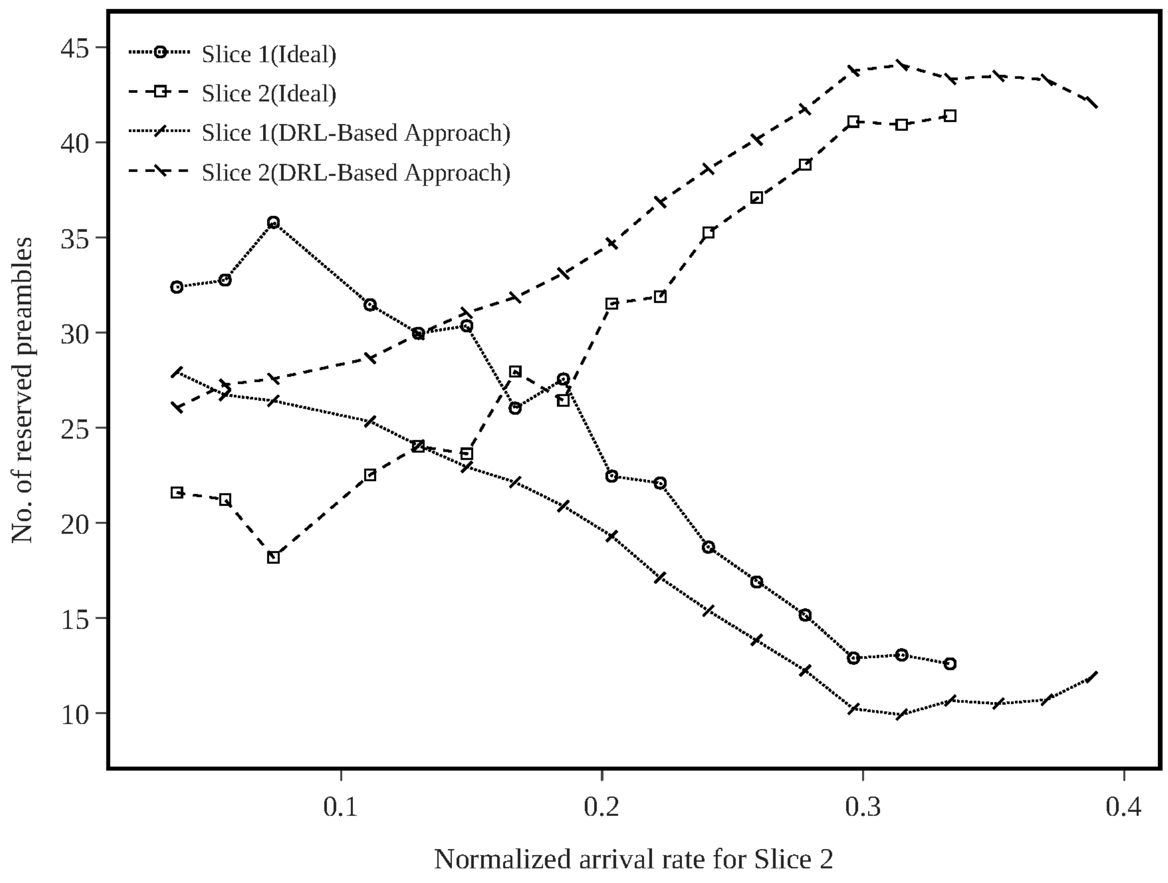


Figure 5.4. Preamble allocations computed by the *ideal algorithm* against the DRL-based approach in 2-slice scenario for the PRF.

Finally, the *ideal algorithm* given as a baseline in [1] is used for comparison. The ideal algorithm assumes that the exact number of preamble request counts for a given RAO (b_j) is known for each priority. It is noted that the preamble request count for the next RAO for slice j, b_j, is found by adding the number of new arrivals (n_j) to the number of nodes which has not transmitted W times yet. Then, by using the Equation 29 in [1], *the number of reserved preambles* to slice j (m_j) in each RAO can be found.

$$M = \sum_{j=1}^{N} m_j = \sum_{j=1}^{N} \frac{b_j}{-\ln(\frac{k_j}{\sum_{i=1}^{N} k_i}) - \ln(x)} \tag{29}$$

Here, x denotes the proportionality factor which satisfies the above equation. Assuming b_j values are known, x is also can be found by solving the equation and hence, m_j values are obtained.

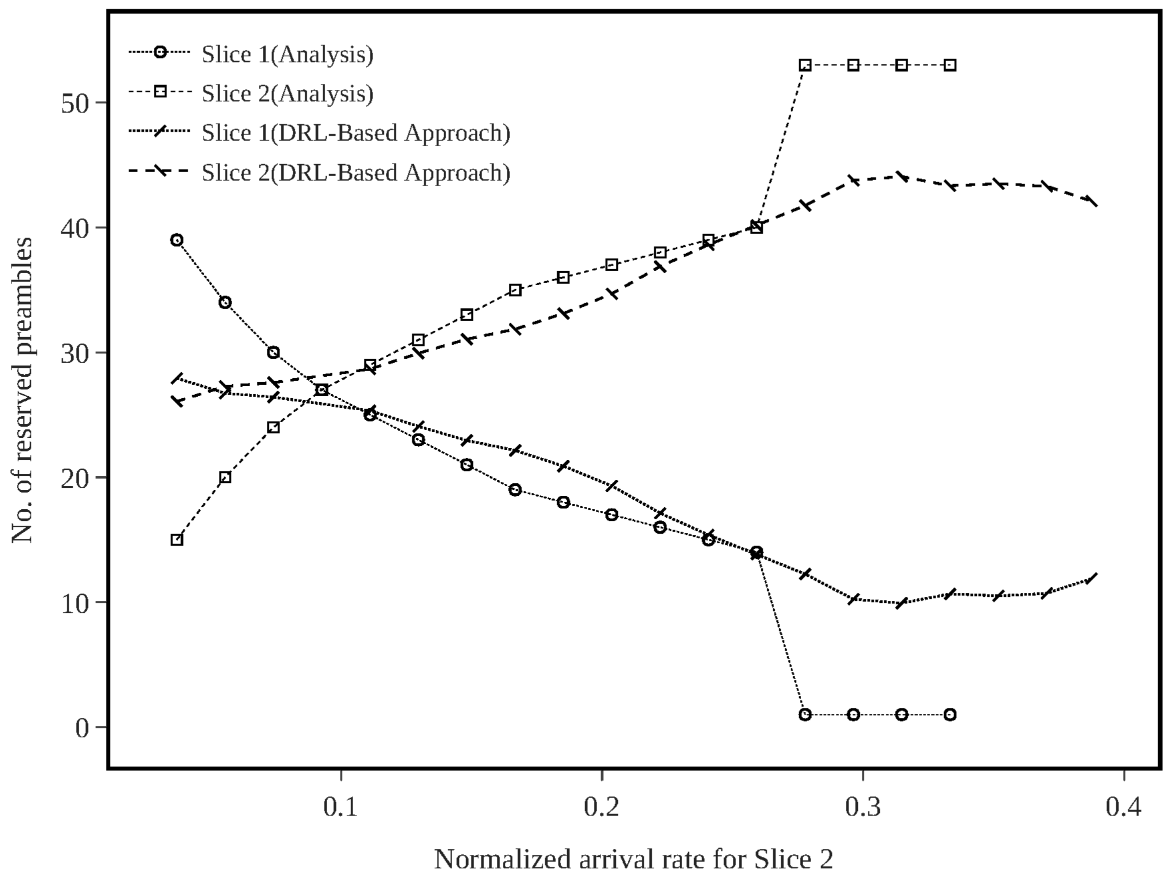


Figure 5.5. Preamble allocations computed by mathematical analysis against the DRL-based approach in 2-slice scenario for the PRF.

5.5. The Performance of the Methods Using Traffic Distribution in Sec. 5.1.1.

In this section the simulation results for 2-slice scenario are discussed using Figs. 5.3.-5.24.. All the figures show the same scenario with the different comparison metrics. For 2-slice scenario both CRF and DRF are used. Likewise, Figs. 5.25.-5.33. are used to discuss the simulation results for 3-slice scenario. The simulations with 3-slice scenario differs from 2-slice in that they only use CRF, after it is made sure that PRF performs worse than CRF both in DRL-based and NEAT-based proposals using 2-slice.

5.5.1. 2-slice Scenario

5.5.1..1 *The Number of Reserved Preambles*

Firstly, the proposed DRL-based approach is compared with the mathematical analysis given

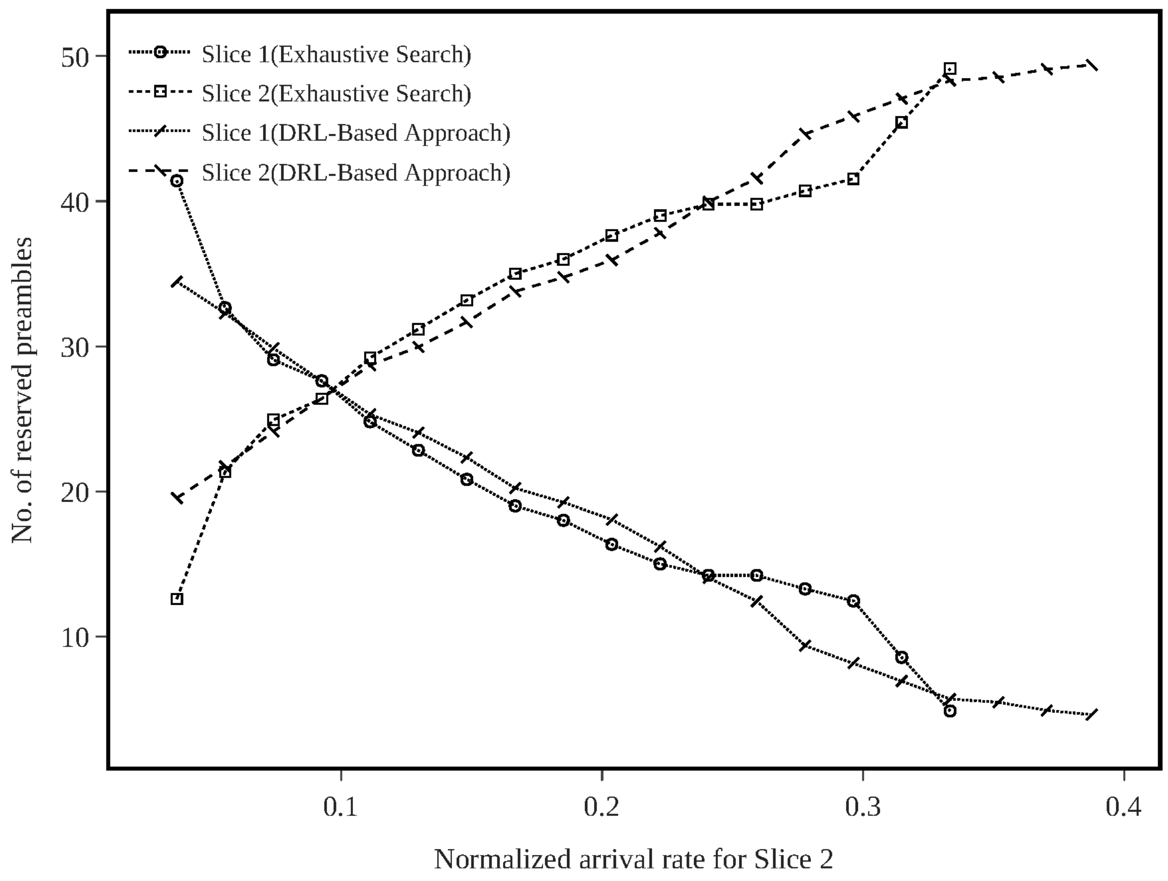


Figure 5.6. Preamble allocations computed by the exhaustive search against the DRL-based approach in 2-slice scenario for the CRF.

in Sec. 4.3., the exhaustive search and the *ideal algorithm* for the scenario with 2 slices. Since the 2-slice scenario has a smaller state space, it is preferred for a baseline comparison. Figs. 5.3.-5.5. and 5.6.-5.8. plot the optimum number of reserved preambles for each slice that maximizes the reward functions. The results show that the preamble allocations which yield maximum rewards for each technique behave very similarly especially when CRF is used. Also Figs. 5.3.-5.5. and 5.6.-5.8. denote DRL-based approach using CRF behaves more aggressive than PRF since it allocates almost all preambles to higher priority slice. While CRF plots a sharper line, PRF follows a smoother path in DRL-based approach compared to CRF. It is also denoted that, in the beginning of the simulation low-priority slice has more preamble allocated than high-priority one in mathematical analysis, exhaustive search and CRF. However, in PRF-based approach high-priority slice gets the majority of the preambles even in the very beginning the simulation. When λ_2 exceeds 0.08 in Figs. 5.6.-5.8., DRL

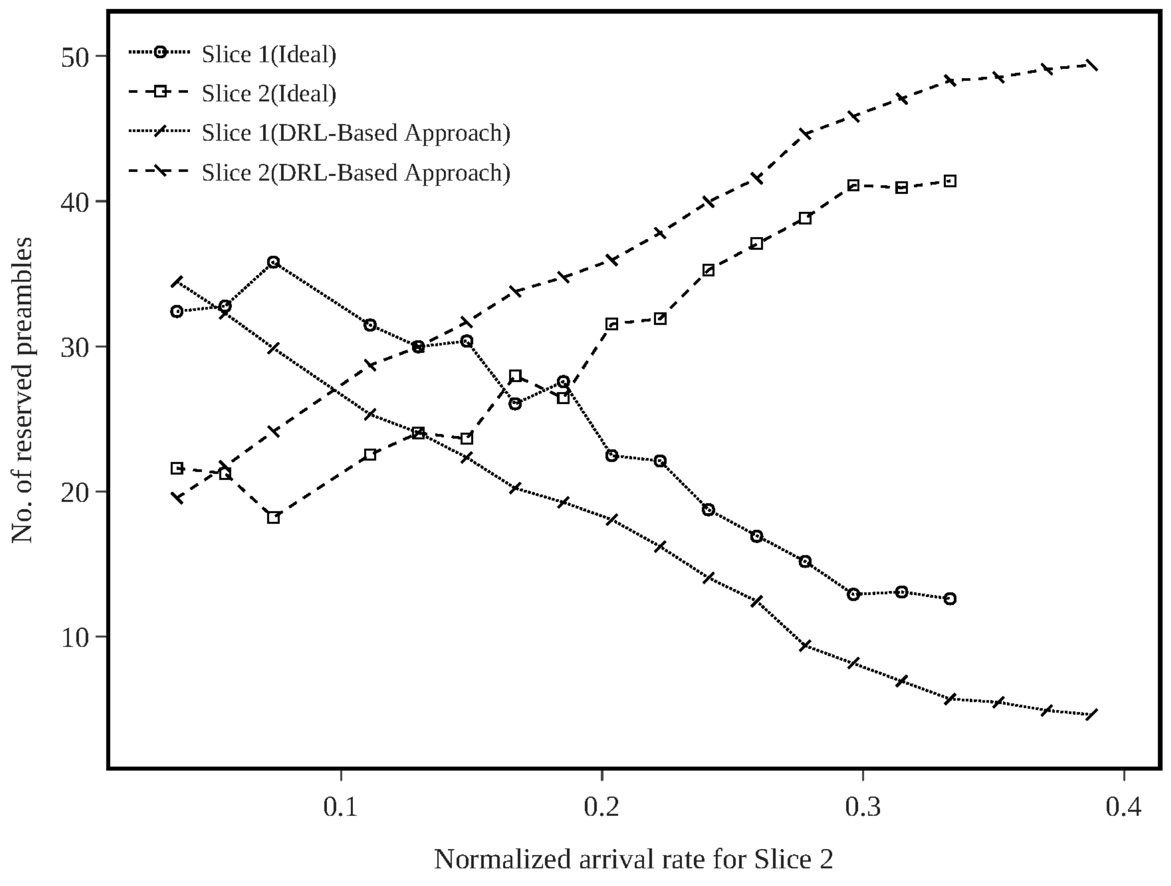


Figure 5.7. Preamble allocations computed by the *ideal algorithm* against the DRL-based approach in 2-slice scenario for the CRF.

starts to give majority of the preambles for high-priority slice meanwhile, the traffic to low-priority one $\lambda_1 \approx 0.09$ is slightly higher. That is another indicator that CRF take precautions on right moment and is more chary compared to PRF.

Following, the proposed NEAT-based approach is also compared with the mathematical analysis given in Sec. 4.3., the exhaustive search and the *ideal algorithm* for the scenario with 2 slices. The reserved preamble counts for slices using NEAT-based PRF are plotted in Figs. 5.9.-5.11.. On the contrary to DRL, using NEAT-based PRF approach there exists prioritization error between the slices in plots. Although the traffic to the high-priority slice increases in time, NEAT-based PRF allocates the preambles equally to the high-priority and low-priority slice. Even, small majority of the preambles are allocated for low-priority one. Also, it is obvious that NEAT-based PRF behaves different from mathematical analysis and exhaustive search. There is significant performance degradation using it compared to DRL.

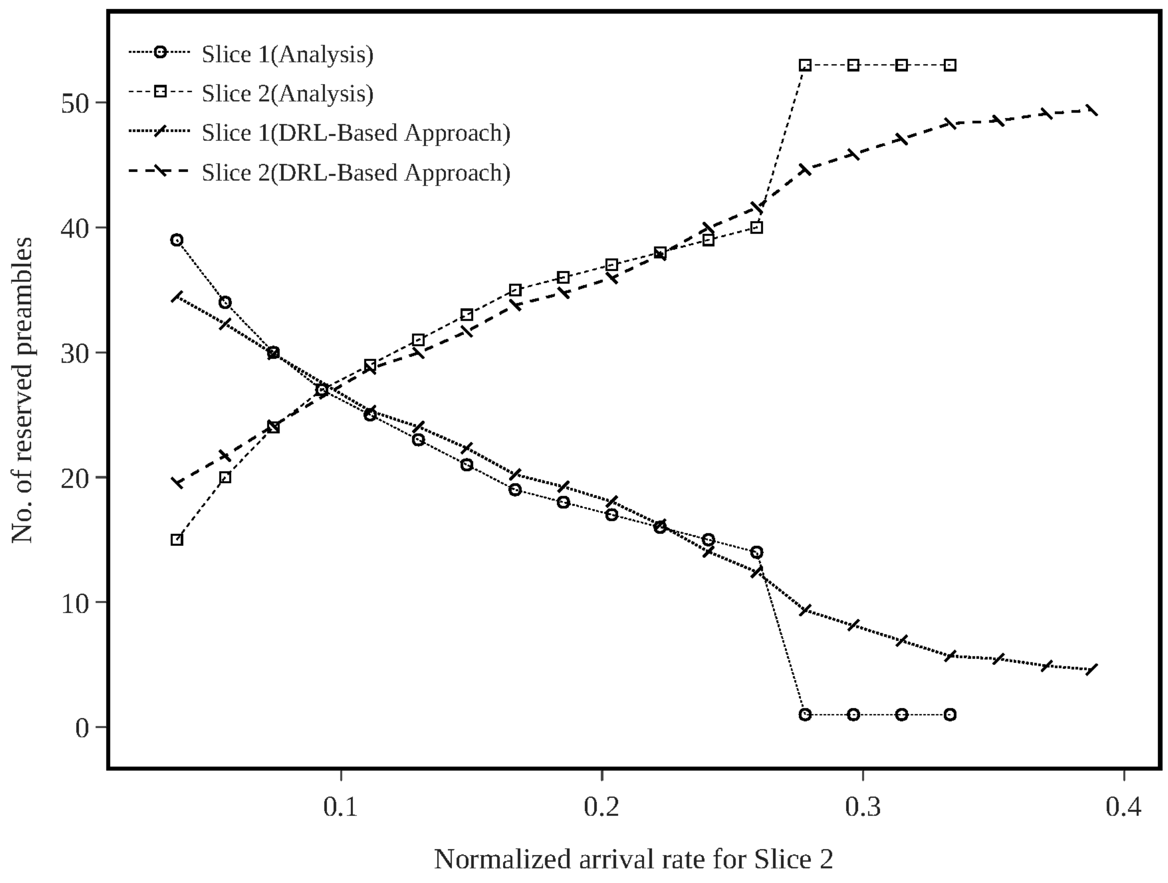


Figure 5.8. Preamble allocations computed by the mathematical analysis against the DRL-based approach in 2-slice scenario for the CRF.

On the other hand, the result of NEAT-based CRF approach in Figs. 5.12.-5.14. show the the analysis, exhaustive search and CRF plots overlap. When λ_2 exceeds 0.12 in Figs. 5.12.-5.14., NEAT starts to give the majority of the preambles for high-priority slice. However, the dynamic behaviour is belated comparing to CRF in DRL. Nevertheless, NEAT-based CRF successfully allocates the majority of the preambles dynamically for the high-priority slice.

5.5.1..2 ***The Ratio of Dropped Messages to All Transmitted Messages***

Figs. 5.15.-5.19. demonstrate *the ratio of dropped messages to all transmitted messages* in each increase of normalized arrival rate of $2nd$ slice for 2-slice scenario. The results for the reward and fitness functions *CRF* and *PRF* are given for DRL in Fig. 5.15., Fig. 5.16. and for NEAT in Fig. 5.17., Fig. 5.18. respectively. Fig. 5.19. presents the behaviour of *ideal algorithm* for the same scenario.

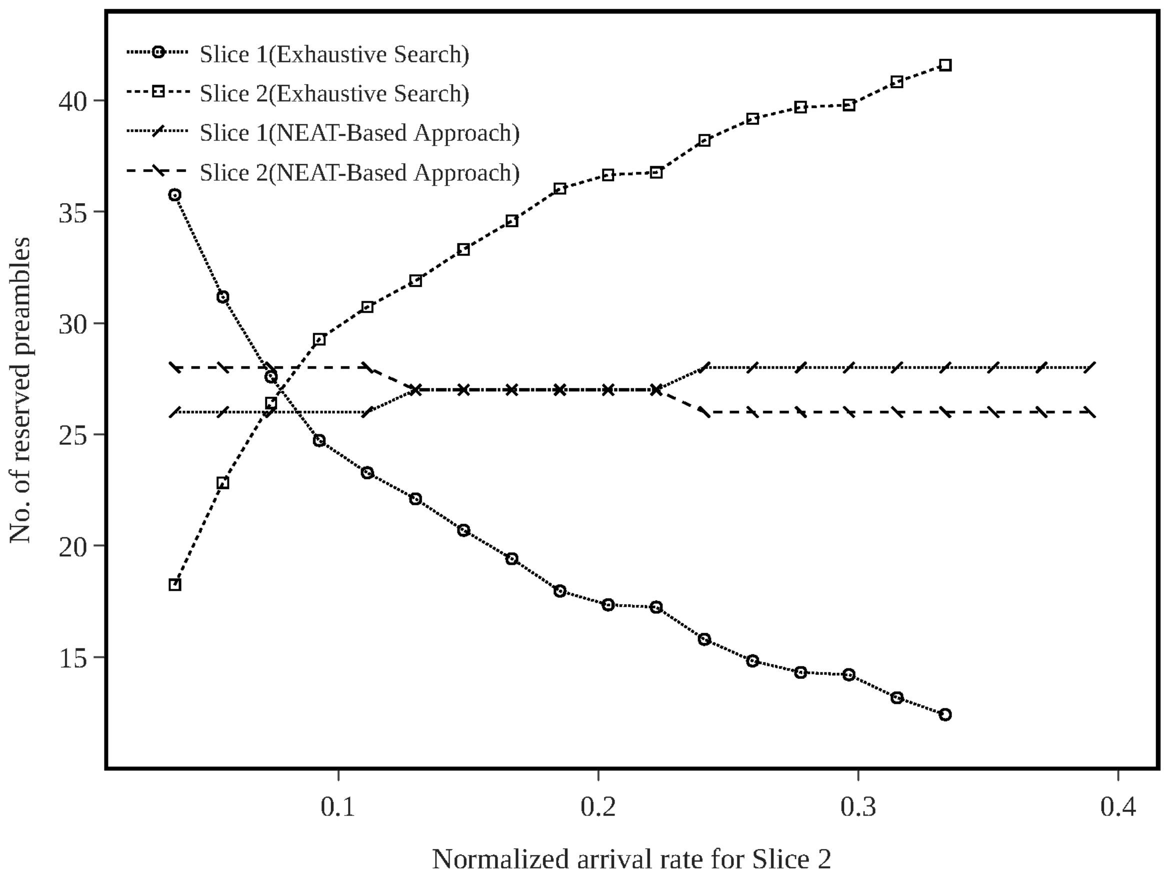


Figure 5.9. Preamble allocations computed by the exhaustive search against the NEAT approach in 2-slice scenario for the PRF.

For DRL case, in Figs. 5.15. and 5.16., while the high-priority slice outperforms the unsliced case for both reward functions, the low-priority slice also gets a considerable amount of reserved preambles. Fig. 5.15. points out that the high-priority slice using *CRF* can satisfy almost all preamble requests so that preamble request messages are nearly not dropped even when the total load passes $(n_1+n_2)/M = 25/54 \approx 0.46$. On the other hand, the unsliced plot has a sharp increase after passing the maximum normalized traffic load limit that the random access channel can handle $(1/e \approx 0.37)$. Fig. 5.16. shows using PRF-based DRL, even if high-priority slice outperforms unsliced one, it can not succeed not to drop any preamble requests.

For NEAT case, in Fig. 5.17., high-priority slice overcomes the unsliced one from $\lambda_2 \approx 0.33$ to $\lambda_2 \approx 0.38$ using CRF. However, the method can not manage to reduce the ratio after λ_2 exceeds 0.38 rather it prefers to reduce the ratio of low-priority slice. The moment when

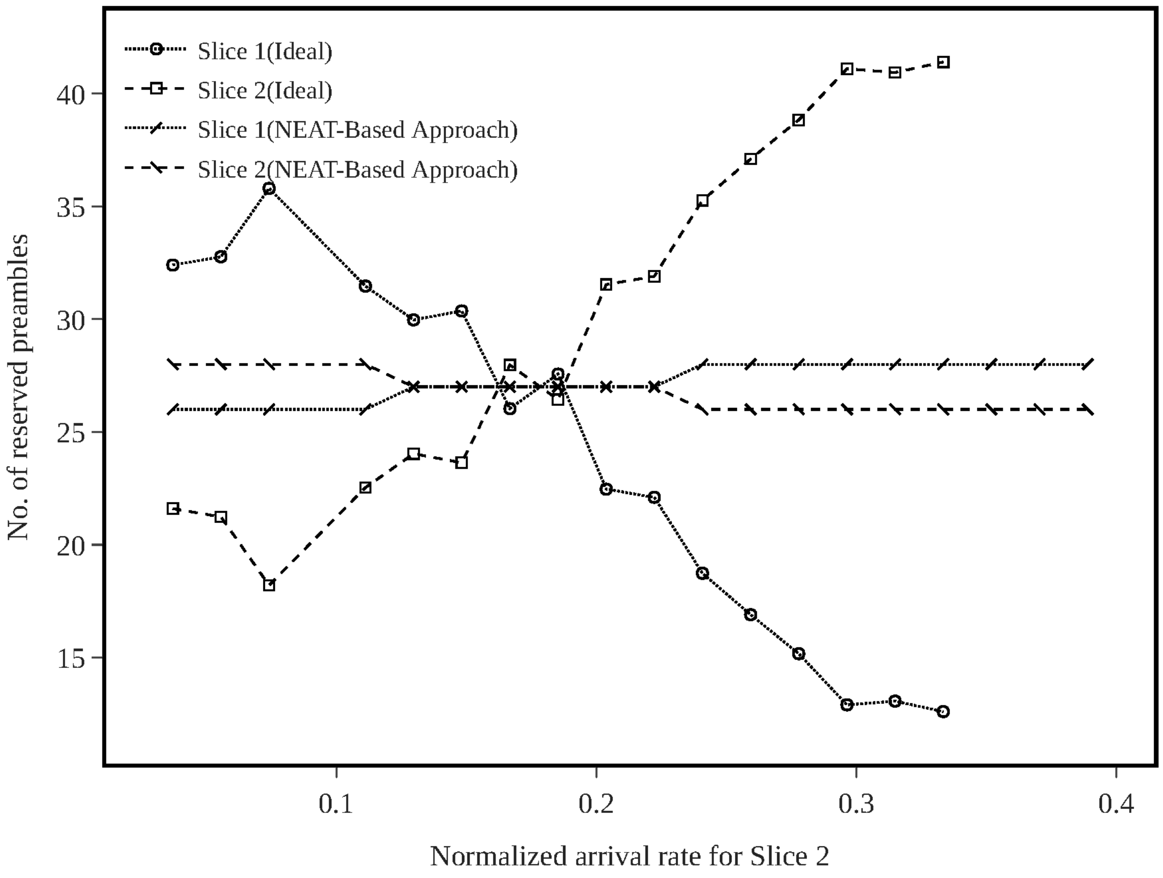


Figure 5.10. Preamble allocations computed by the *ideal algorithm* against the NEAT approach in 2-slice scenario for the PRF.

$\lambda_2 \approx 0.34$ in Figs. 5.12.-5.14. is the point where number of reserved preambles for high-priority slice is maximized, $m_2 \approx 43$. After that point, even if λ_2 increases, *the number of reserved preambles* for high-priority slice doesn't increase contrary it drops from ≈ 42 to ≈ 40. This is the reason why method fails reducing the ratio for high-priority slice.

Fig. 5.18. shows the ratio using PRF-based NEAT. Likewise in Figs. 5.12.-5.14., PRF-based NEAT method performs worse comparing both DRL and CRF-based NEAT. Although there is no dropped message observed for low-priority slice, when λ_2 exceeds 0.2 *the ratio of dropped messages to all transmitted messages* experiences sharp increase up to 1.

The behaviour of the *ideal algorithm* is demonstrated in Fig. The algorithm slightly prioritizes the slices in a way that plots of each slice follows the unsliced plot with small differences in prioritization. For example, when $\lambda_2 \approx 0.35$, both slices have nearly 0.8 ratio of dropped messages in the *ideal algorithm*, while these ratios are 1 and 0 respectively for the 1st and

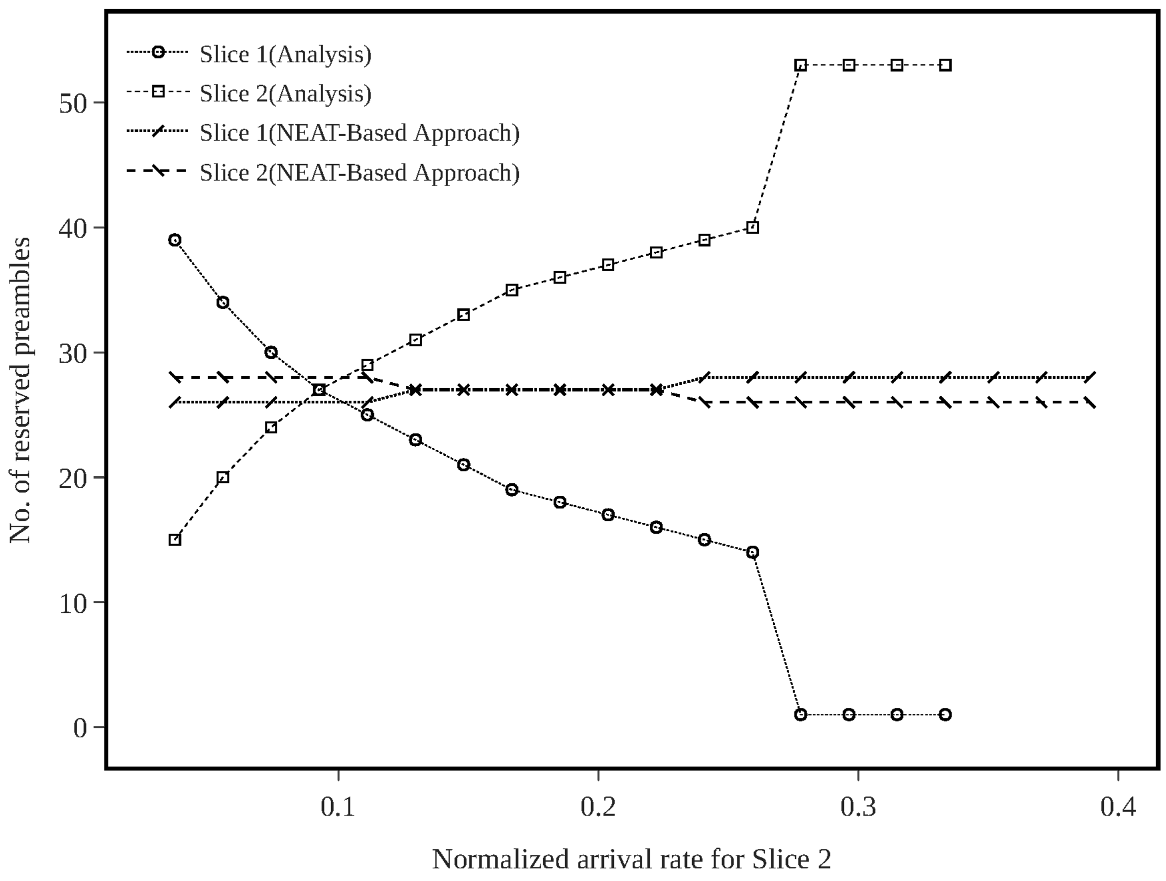


Figure 5.11. Preamble allocations computed by mathematical analysis against the NEAT approach in 2-slice scenario for the PRF.

2nd slices for CRF in Fig. 5.15.. Therefore, the *ideal algorithm* remarks a fair approach to the all slices, hence the difference between prioritizing slices is minor.

5.5.1..3 ***The average waiting time* in ms**

Figs. 5.20.-5.24. present the average waiting time, which denotes how much time successful preamble requests waited in the message backlog on average. The results for the reward and fitness functions *CRF* and *PRF* are given for DRL in Fig. 5.20., Fig. 5.21. and for NEAT in Fig. 5.22., Fig. 5.23. respectively. Also, Fig. 5.24. presents the behaviour of the *ideal algorithm* for the same scenario. If there is no successful preamble request in the RAO, the average waiting time can not be calculated for that RAO. Nevertheless, to show the performance, the value is set to the maximum average waiting time which is $10 \times W$ ms. Also, the dropped messages are not taken into the account. For example, assuming that 10 preamble requests are issued from the backlog in a RAO with 9 of them dropped and 1

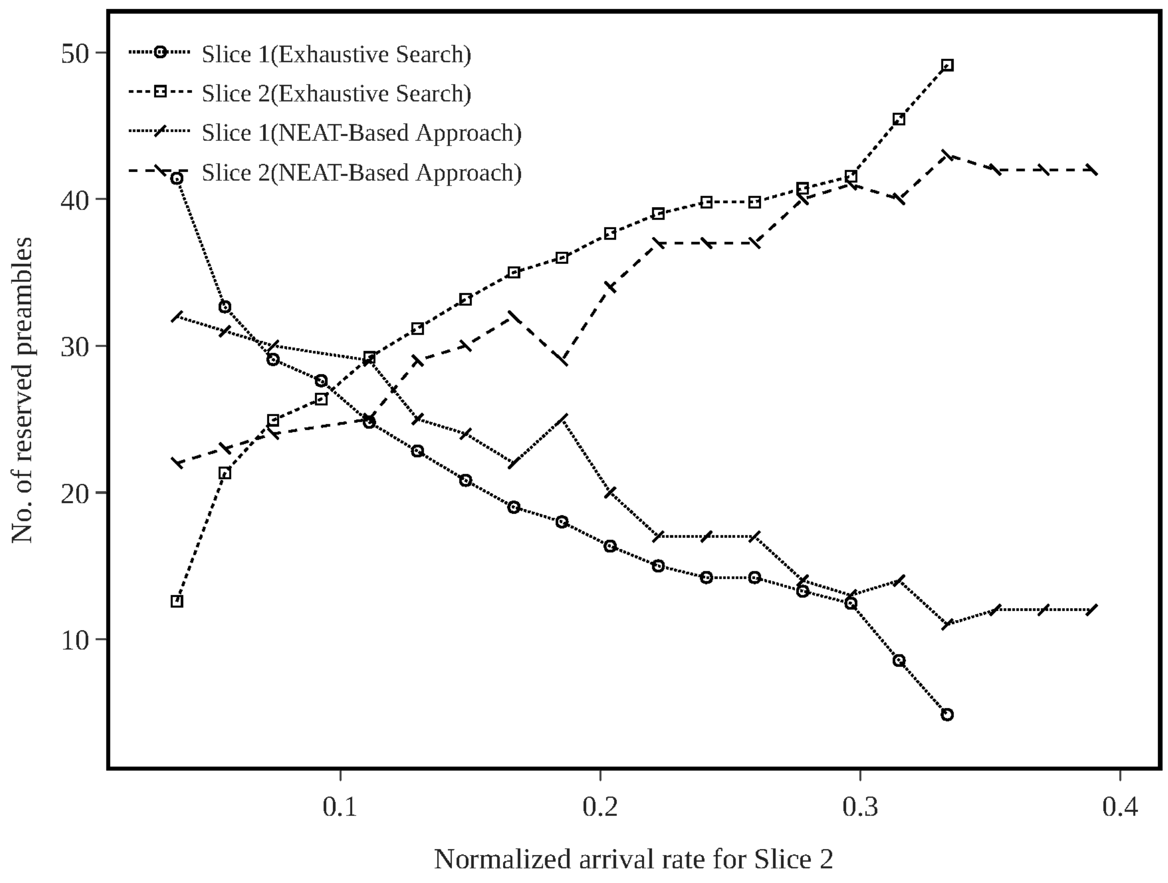


Figure 5.12. Preamble allocations computed by the exhaustive search against the NEAT approach in 2-slice scenario for the CRF.

request on 4th try succeeded, than the average waiting time is calculated as 40ms. Therefore, the average waiting time performance must be evaluated with *the ratio of dropped messages to all transmitted messages*.

For the DRL model, the average waiting time for the high-priority slice almost stays constant despite the increasing traffic rate. Still, the average waiting time for the high-priority slice for *PRF* is higher in comparison to the waiting time of the same slice for *CRF* in DRL. Moreover, although *PRF* scheme distributes the preambles more fairly to two slices compared to CRF as seen in Figs. 5.3.-5.5., when the total load passes $(n_1 + n_2)/M = 21/54 \approx 0.39$, dropped preamble requests are observed.

For the NEAT model, the average waiting time for the high-priority slice using CRF stays constant up to $\lambda_2 \approx 0.34$ in Fig. 5.22.. However, it seems CRF-based DRL keep the average waiting time under control up to higher λ_2 values compared to CRF-based NEAT. In

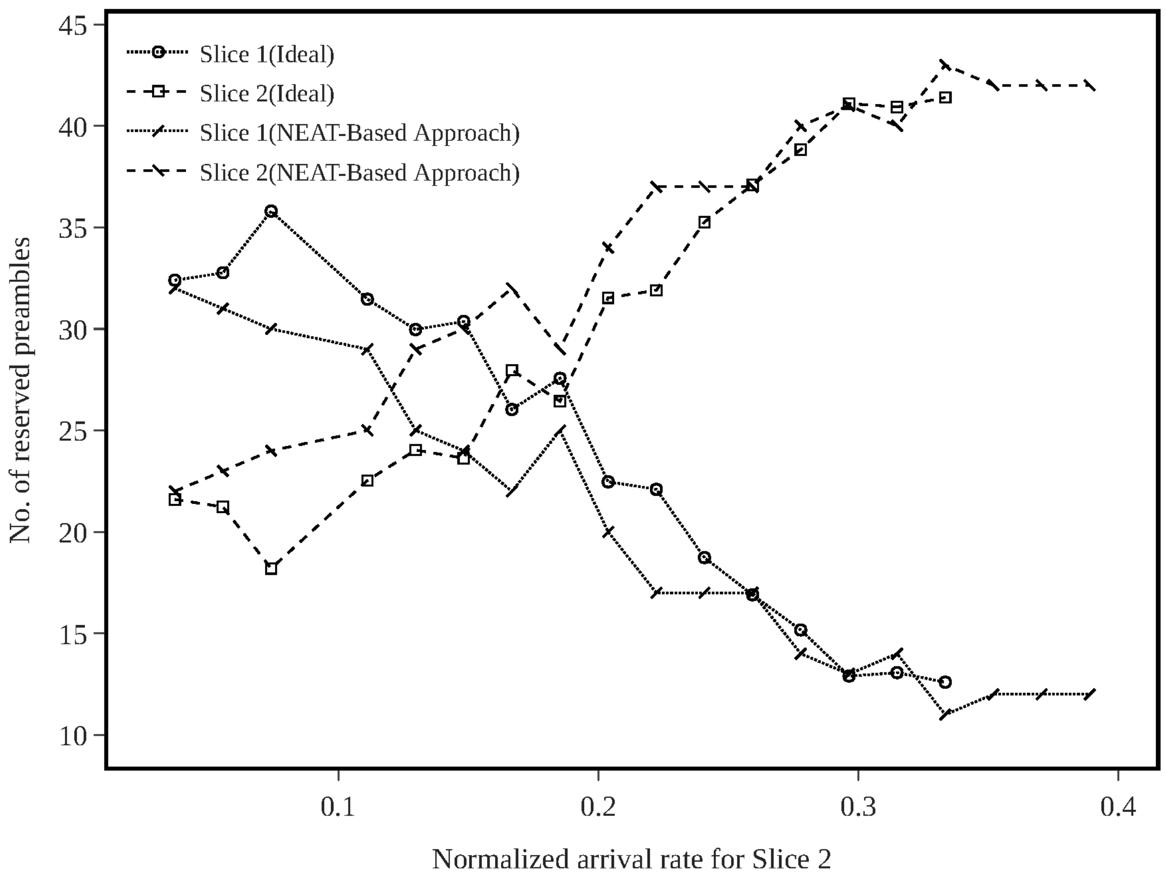


Figure 5.13. Preamble allocations computed by the *ideal algorithm* against the NEAT approach in 2-slice scenario for the CRF.

Fig. 5.23., the performance of PRF-based NEAT is given. It shows that, using PRF-based NEAT both low-priority slice and unsliced have lower average waiting time compared to high-priority one. Since, the PRF performs worse in all metrics in both DRL and NEAT comparing to CRF, the results which use only *CRF* are presented in the rest of the experiments.

The behaviour of the ideal algorithm is demonstrated in Fig. 5.24.. Similar to the results given in the previous section, Each slice follows the unsliced plot with small differences.

5.5.2. 3-slice Scenario

The simulation results for the 3-slice scenario is given in Figs. 5.25.-5.33.. As noted above, here the following performance metrics are evaluated for only the *CRF*. At time $t = 0$, $n_1 = 5, n_2 = 4, n_3 = 2$ and the relation between *the number of reserved preambles* is

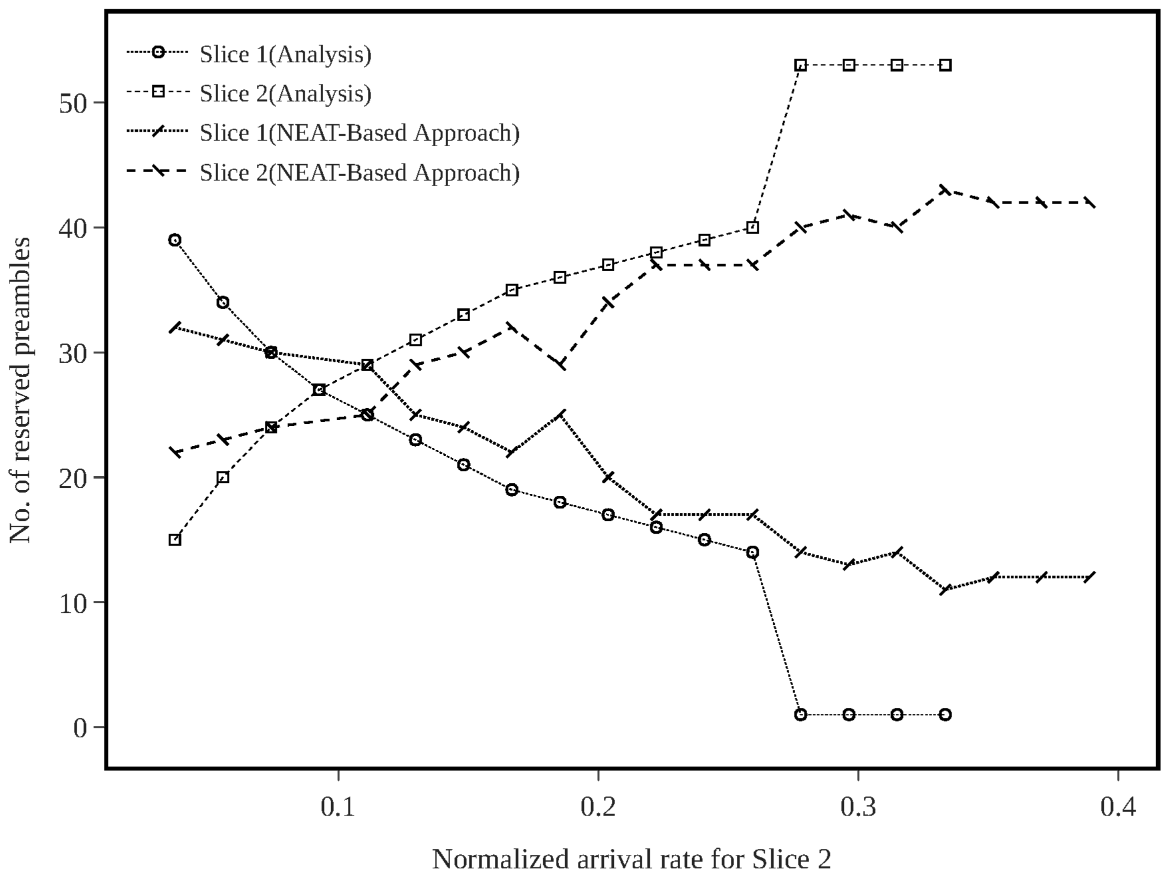


Figure 5.14. Preamble allocations computed by the mathematical analysis against the NEAT approach in 2-slice scenario for the CRF.

$m_1 > m_2 > m_3$. Since the total load is $((n_1 + n_2 + n_3)/54 \approx 0.20) < (1/e \approx 0.37)$, the proposed method allocates more preambles to $1st$ and $2nd$ slices as shown for DRL in Fig. 5.25. and for NEAT in Fig. 5.31. in order to relieve traffic to the low priority slices when the high-priority slice easily handles the traffic. As n_3 increases, both NEAT and DRL methods aggressively increase the reserved preambles count for the $3rd$ slice in order to avoid dropping messages and long average waiting.

While DRL manages not to drop any preamble allocation requests from the $3rd$ slice even the total load passes $(n_1 + n_2 + n_3)/M = 26/54 \approx 0.48$ in Fig. 5.26., NEAT can manage this around $(n_1 + n_2 + n_3)/M = 19/54 \approx 0.36$ in Fig. 5.32.. *The average waiting time* for the $3rd$ slice follows a horizontal line up to the normalized arrival rate of 0.3 for DRL. After that, it slightly increases due to high load on the RACH; yet, the method is able to prevent a sharp increase as seen in Fig. 5.27.. For the NEAT-based model, the average waiting time for the $3rd$ slice follows a similar path to the DRL up to the normalized arrival rate of 0.2 as

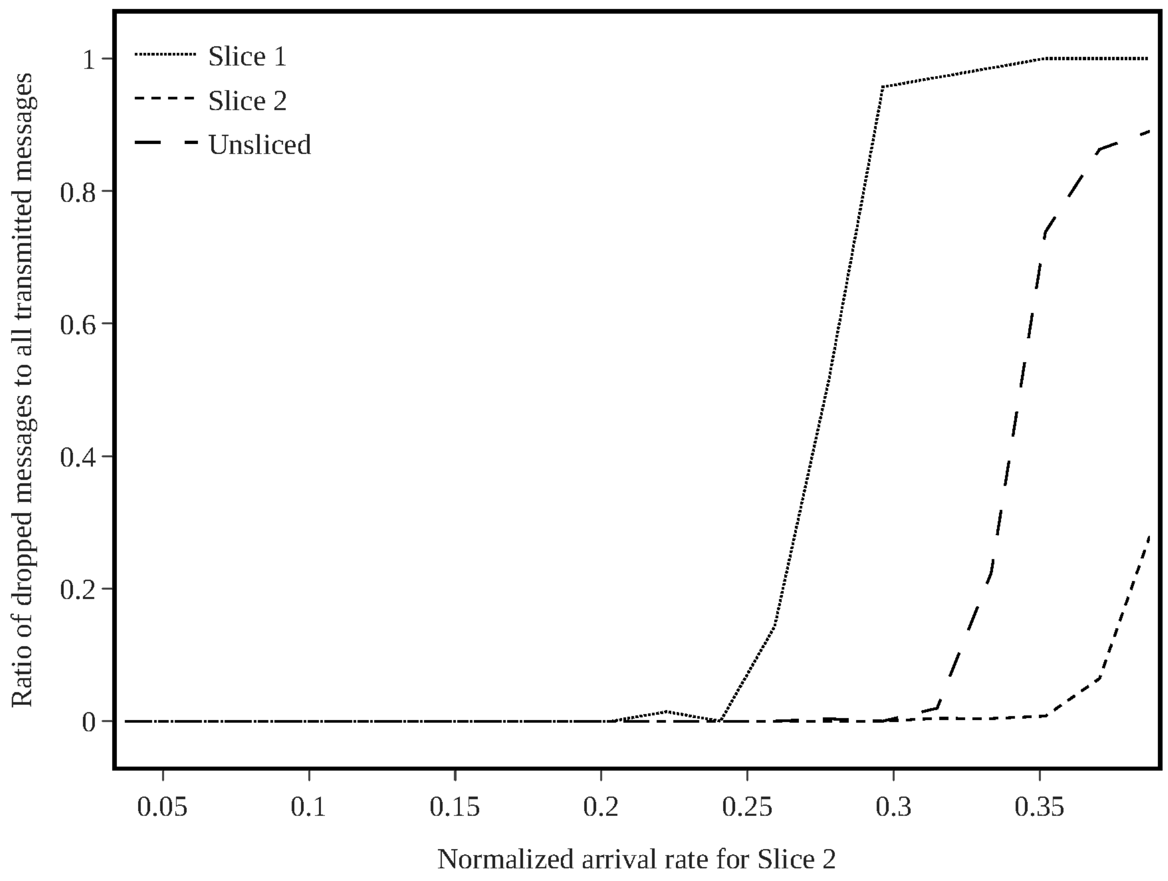


Figure 5.15. *The ratio of dropped messages to all transmitted messages* for 2-slice scenario while the arrival rate of second slice consistently increased using CRF-based DRL.

shown in Fig. 5.33.. After that, it sharply increases due to high load on the RACH contrary to DRL when comparing Fig. 5.27. and Fig. 5.33.. On the other hand, the unsliced curve also shows a sharp increase after passing the maximum normalized traffic load limit that the RACH can handle ($1/e \approx 0.37$), as in the 2-slice scenario.

There is a major difference between the NEAT and DRL-based models for the 3-slice scenario when comparing Fig. 5.25. and Fig. 5.31.. While DRL model reserves more of the preambles to $2nd$ than $1st$, NEAT have done the opposite. Since $n_1 > n_2$, the prioritization interpretation of NEAT may seem more successful than DRL at first. However, Fig. 5.32. simply shows the $2nd$ slice experience increase of the *ratio of dropped messages to all transmitted messages* before the $1st$ slice, although the prioritization scheme propose slices should experience increase of *ratio of dropped messages to all transmitted messages* in order of priority levels. In Fig. 5.26., as the total normalized arrival rate increases in time, $1st$ slice experience increase in *ratio of dropped messages to all transmitted messages* first, following

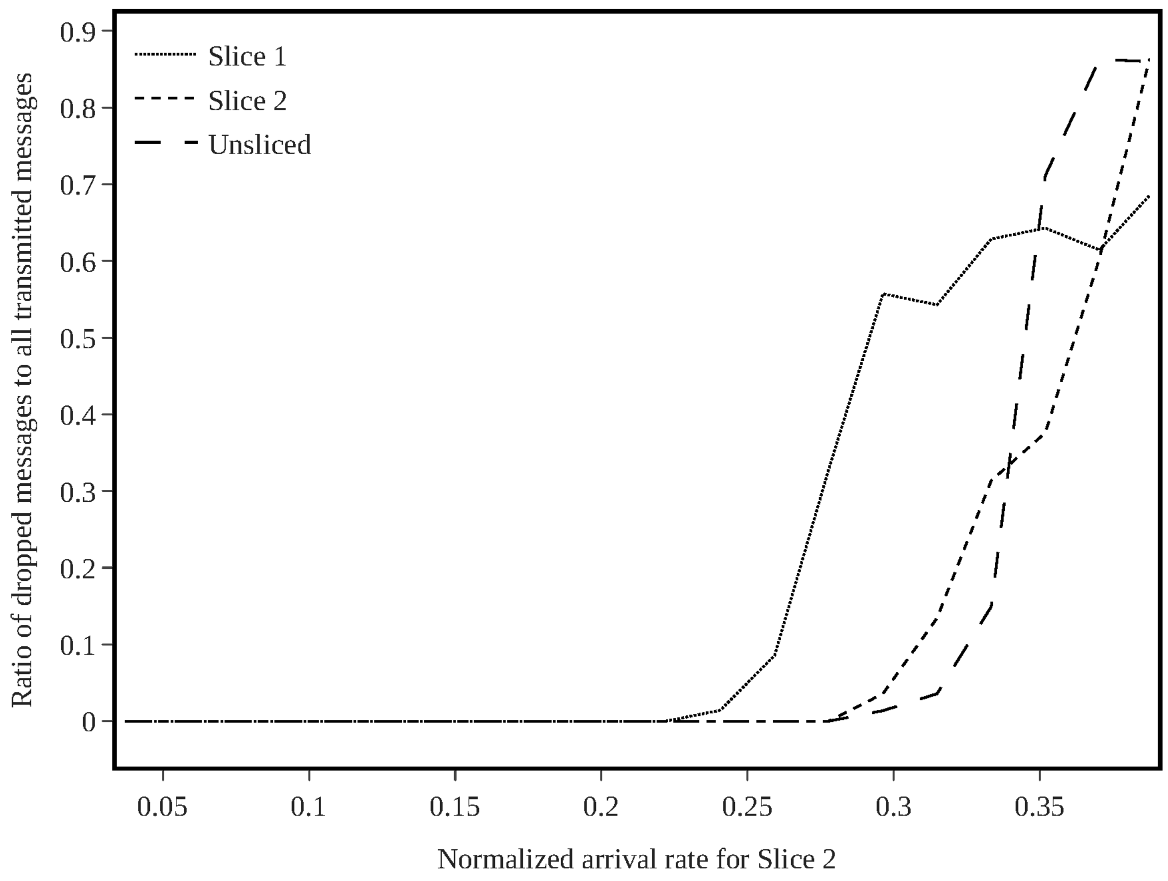


Figure 5.16. *The ratio of dropped messages to all transmitted messages* for 2-slice scenario while the arrival rate of second slice consistently increased using PRF-based DRL.

$2nd$ slice experience increase and finally $3rd$ slice experience increase, as it is supposed to be.

For the *ideal algorithm*, Fig. 5.28. shows that for the 3-slice scenario the algorithm prefers to allocate more number of preambles to the 3rd slice even in the beginning of the scenario when the total load is much less than $1/e \approx 0.37$ and normalized arrival rate for the 1st slice is highest. As a result of using the *ideal algorithm*, there are dropped preamble requests observed in the 1st slice even at the early stage of the scenario when the arrival rate of the 3rd is slice between 0.05 and 0.1 as and the total load is 0.24 shown as in Fig. 5.29..

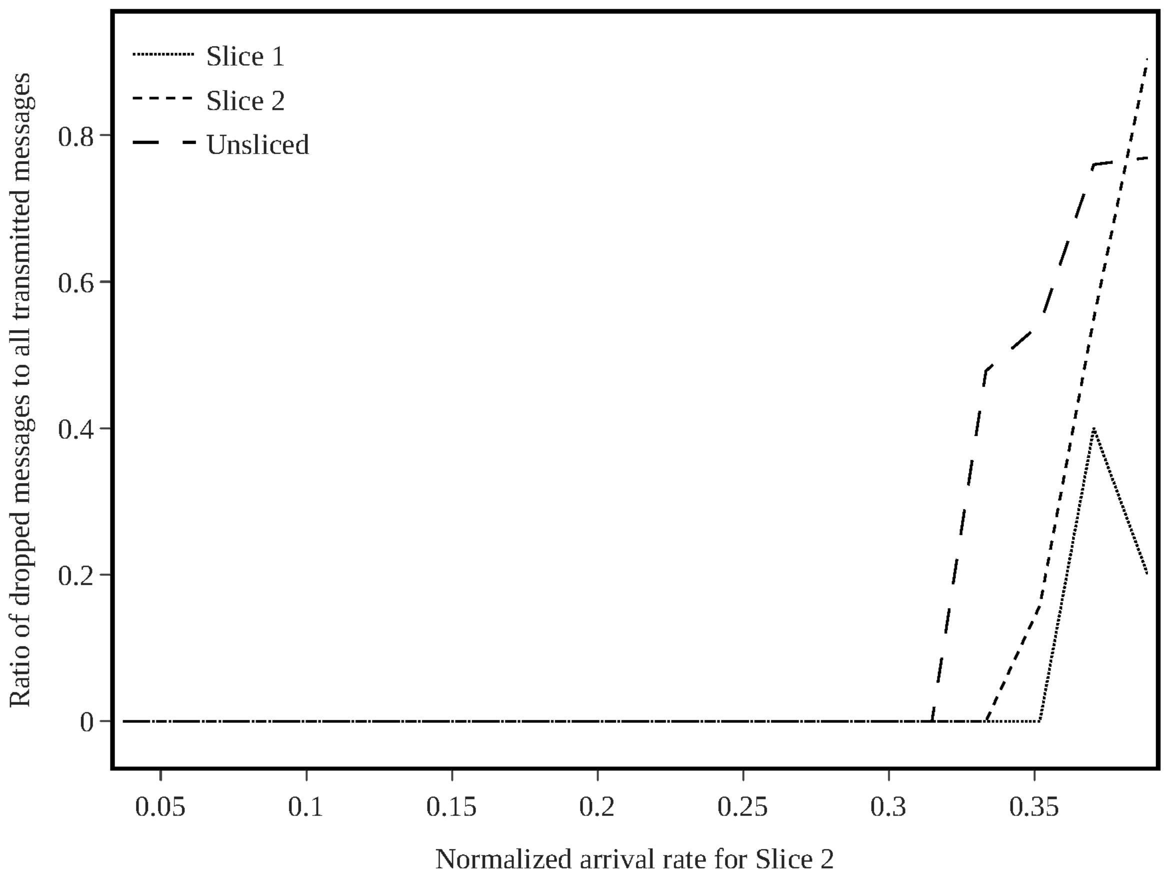


Figure 5.17. *The ratio of dropped messages to all transmitted messages* for 2-slice scenario while the arrival rate of second slice consistently increased using CRF-based NEAT.

5.6. The Performance of the Methods Using Traffic Distribution in Sec. 5.1.2.

In this part, the temporal behavior of the proposed methods are presented to show how these react to changes in the traffic demands and how much time needed to rearrange the preamble allocations to meet the requirements. Differently from the previous experiments, here not only the requests of the high-priority service change, all service requests may vary in time. The simulations are run for the 5-slice scenario in addition to the 3-slice scenario. For these simulations, only *CRF* is used . The traffic pattern and the behaviour of the proposed methods are given in Figs. 5.34.-5.37., the *ideal algorithm* is demonstrated in Figs. 5.38.-5.39..

The uppermost plots in figures show the normalized arrival rate for each slice over time. The following below plots show the response of the proposed method to the changing environment as reserved preamble counts for each slice. The next below plots demonstrate the collision rates of each slice in that random access opportunity which is found using c_j/m_j.

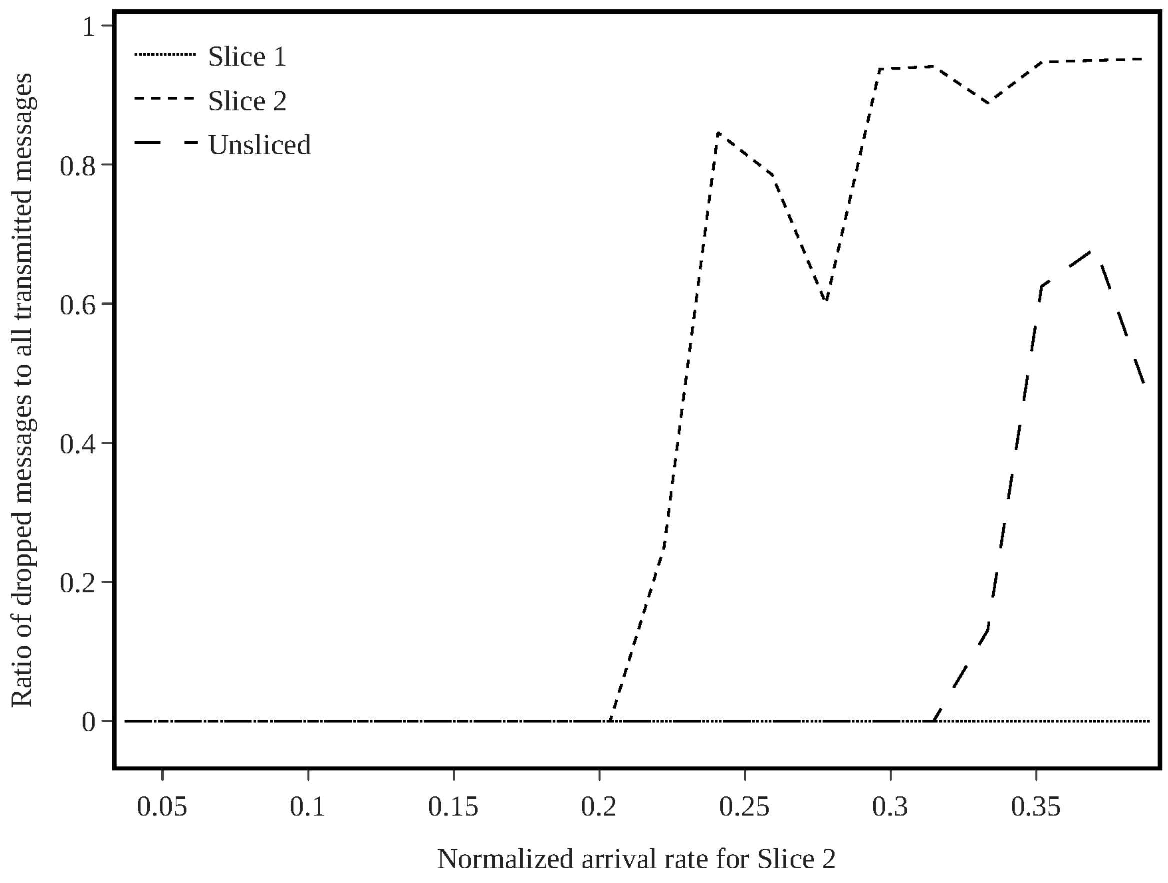


Figure 5.18. *The ratio of dropped messages to all transmitted messages* for 2-slice scenario while the arrival rate of second slice consistently increased using PRF-based NEAT.

Finally, the bottom plots show *the ratio of dropped messages to all transmitted messages* on that random access opportunity.

5.6.1. 3-slice Scenario

Figs. 5.34., 5.36. and 5.38. plot the simulation results for the 3-slice scenario for DRL, NEAT and the *ideal algorithm* respectively. In Fig. 5.34., from $t = 0$ to $t = 2$, the reserved preamble count for each slice stays almost constant since there exists no congestion and change in traffic. At $t = 2$, λ_1 increases suddenly while other slices still have the same normalized arrival rates. After a reaction time of approximately $50ms$, the DRL method gives most of the preambles to $1st$ slice immediately. First, the collision rate of $1st$ slice increases a little, then, immediately drops thanks to the reallocation of preambles.

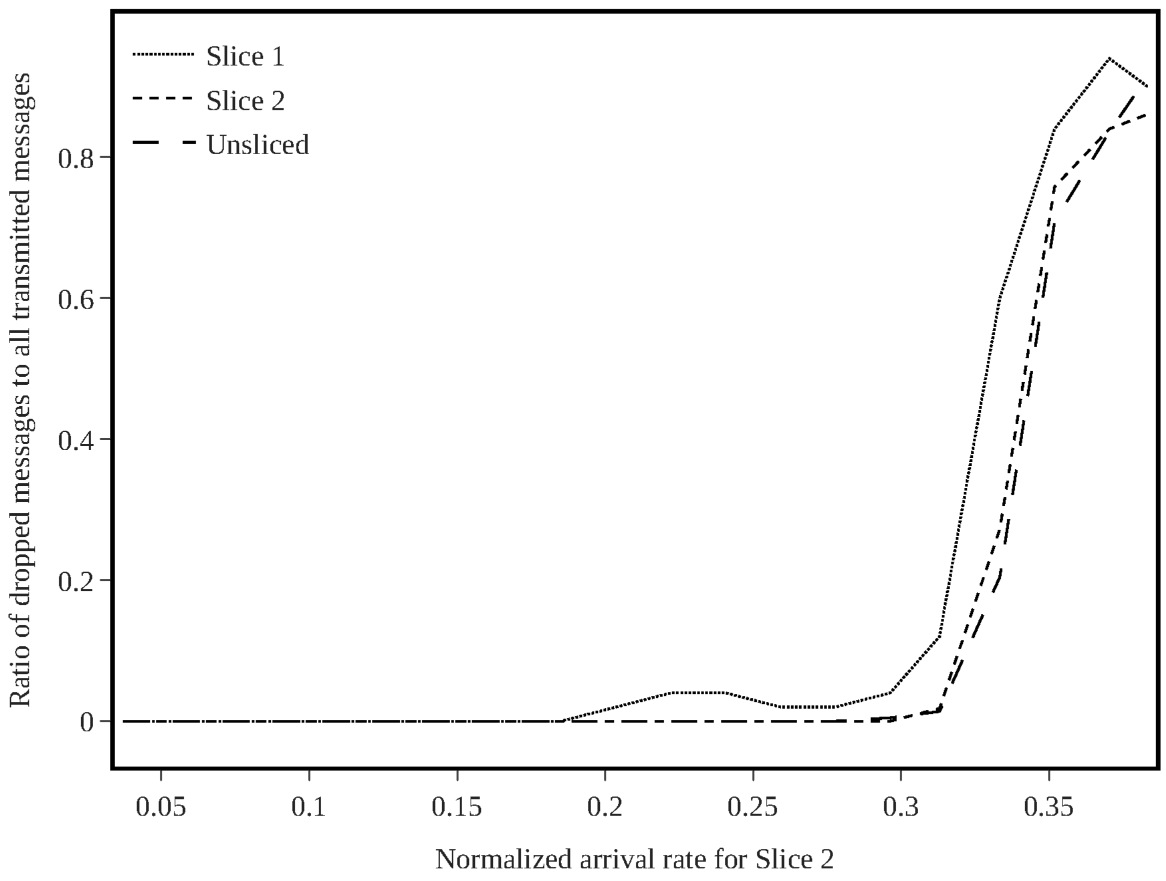


Figure 5.19. *The ratio of dropped messages to all transmitted messages* for 2-slice scenario while the arrival rate of second slice consistently increased using ideal algorithm.

Similarly, at $t = 2.5$, λ_2 increases to a higher value than λ_1. Right after, m_2 increases and $2nd$ slice gets a major part of the preambles. Up until $t = 3$, m_1 and m_2 stays around the same levels with little difference between them. The total load exceeds the total capacity at $t = 3$. λ_3 increases to $12/54 \approx 0.22$ and $\lambda_1 + \lambda_2 + \lambda_3 \approx 0.39$ becomes greater than 0.37. At this point, since the preamble resources are not enough for all slices, the DRL method allocates most of the preambles to the high-priority slice, $3rd$ slice. Therefore, m_3 spikes, m_2 is nearly halved and m_1 is dropped to zero.

As all slices cannot be supported at the same time, the DRL method sacrifices the low-priority slice in favor of other slices. The collision rate for the $1st$ slice oscillate around 1 up to $t = 8$. The collision rates for the other two slices change between 0.2 and 0.6, yet the collision rate for $2nd$ slice is lower than $3rd$ slice since the traffic to $3rd$ slice is more than twice even if $3rd$ slice has the highest priority. In the last two seconds, λ_3 falls of suddenly, right after, m_3 also drops. m_1 and m_2 increase to their previous values where right before

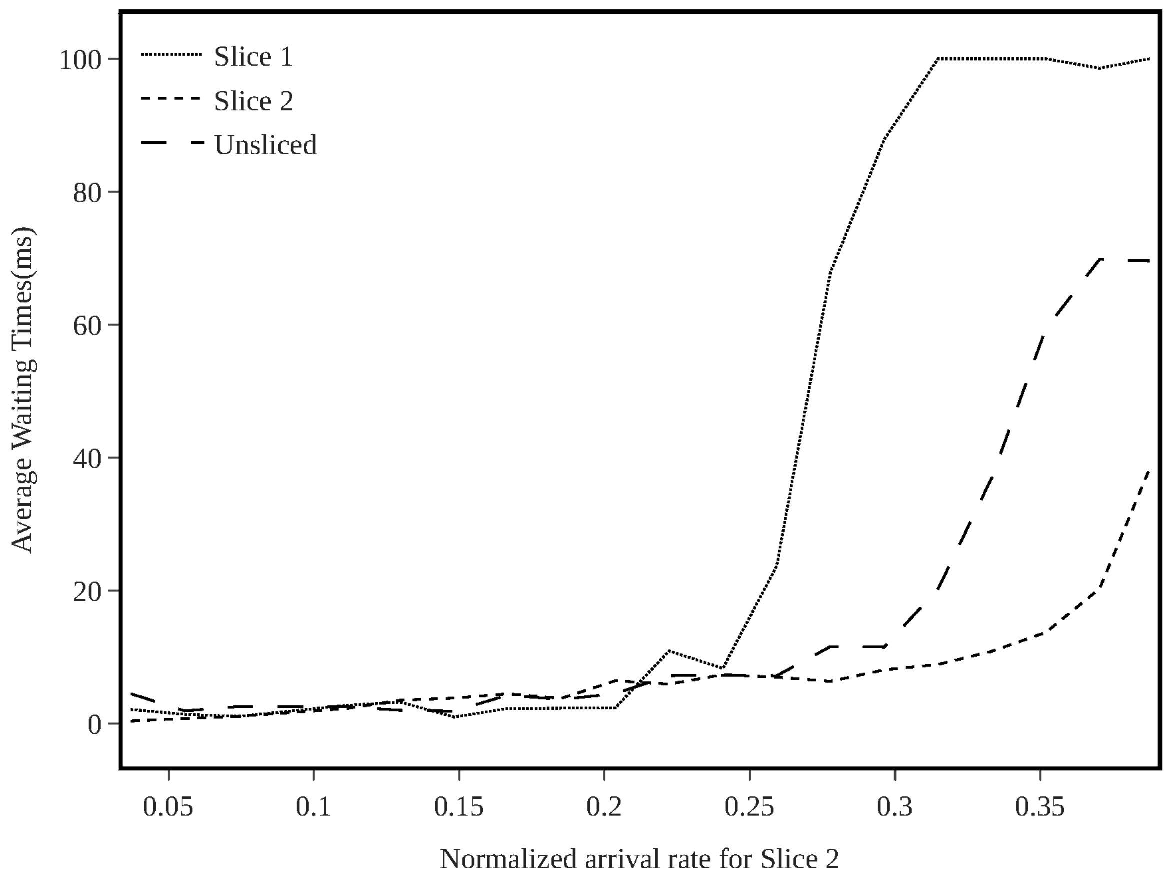


Figure 5.20. *The average waiting time* for 2-slice scenario while the arrival rate of second slice consistently increased using CRF-based DRL.

λ_3 increases. Then, respectively λ_2 and λ_3 drops to their previous values and the preamble allocation returns to its initial assignment.

In Fig. 5.36., the same scenario is conducted using NEAT-based method. At $t = 2$, λ_1 increases suddenly, unlike DRL, NEAT does not react to the change. Therefore, the collision rate of the $1st$ slice increases a little. Than at $t = 2.5$, λ_2 increases to a higher value than λ_1. Right after, m_2 increases and $2nd$ slice gets a considerable amount of the preambles. Yet, NEAT allocates more number of preambles to $1st$ slice. It seems the prioritization is not implemented correctly. Up until $t = 3$, m_1 and m_2 stays around the same levels with little difference between them. The total load exceeds the total capacity at $t = 3$. λ_3 increases to $12/54 \approx 0.22$ and $\lambda_1 + \lambda_2 + \lambda_3 \approx 0.39$ becomes greater than 0.37. At this point, similar to DRL, NEAT also allocates most of the preambles to the high-priority slice, the $3rd$ slice. Therefore, m_3 spikes, m_1 and m_2 is nearly halved.

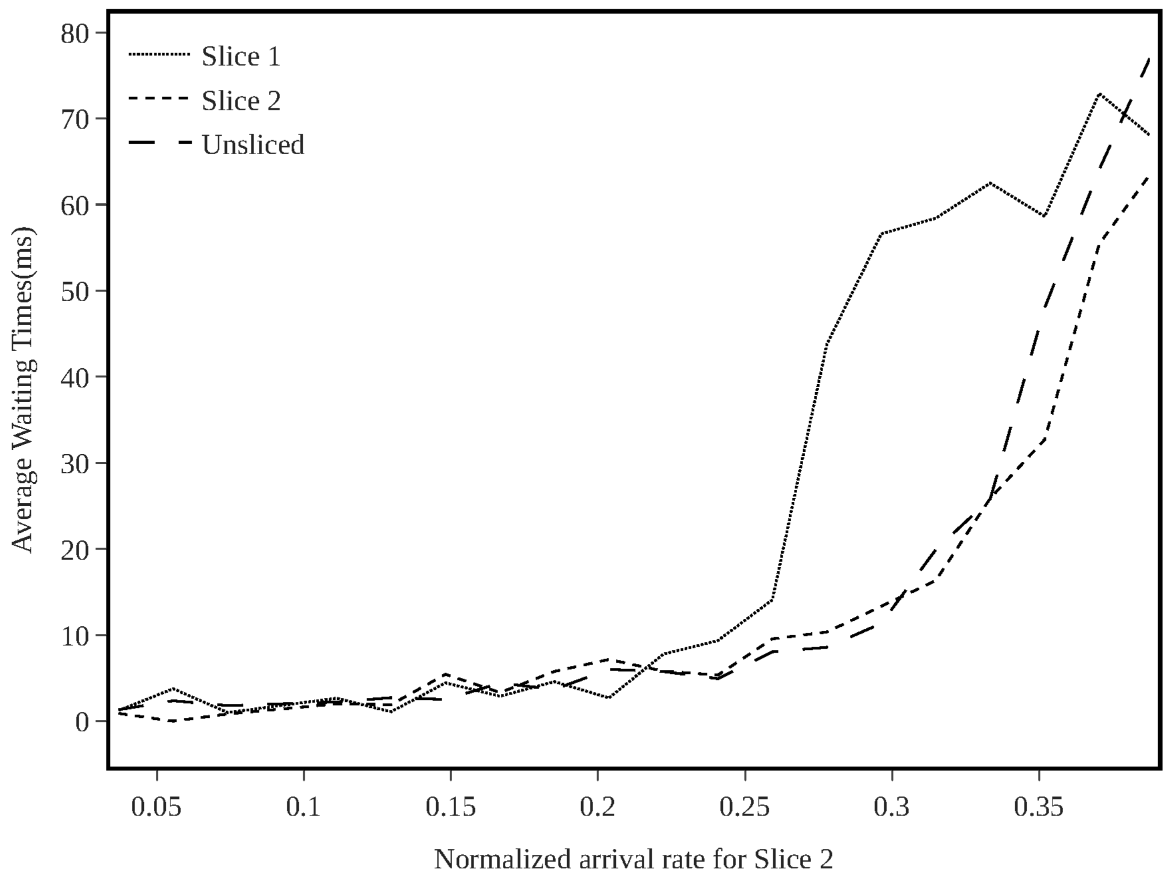


Figure 5.21. *The average waiting time* for 2-slice scenario while the arrival rate of second slice consistently increased using PRF-based DRL.

Unlike DRL, NEAT can not handle the prioritization between $2nd$ and $1st$ slices correctly. Although $\lambda_2 > \lambda_1$, NEAT allocates preambles to the $1st$ slice nearly 2 times more than to the $2nd$ slice. In addition, even if the majority of the preambles are allocated to the $3rd$ slice, the collision rate for the $3rd$ slice and $2nd$ slice oscillate around 1 up to $t = 8$. After $t = 9$, λ_2 and λ_3 drops to their previous values and the preamble allocation returns to its initial assignment. However, comparing to DRL, NEAT-based method fails reducing collision rate and ratio of dropped messages to transmitted messages for the $2nd$ and $3rd$ slices.

5.6.2. 5-slice Scenario

Similarly, Fig. 5.35. and 5.37. plot the timeline simulation for the 5-slice scenario for DRL and NEAT respectively. In Fig. 5.35., from $t = 0$ to $t = 2$, there is no congestion and the preamble allocation of slices does not change. During this period, the number of preambles

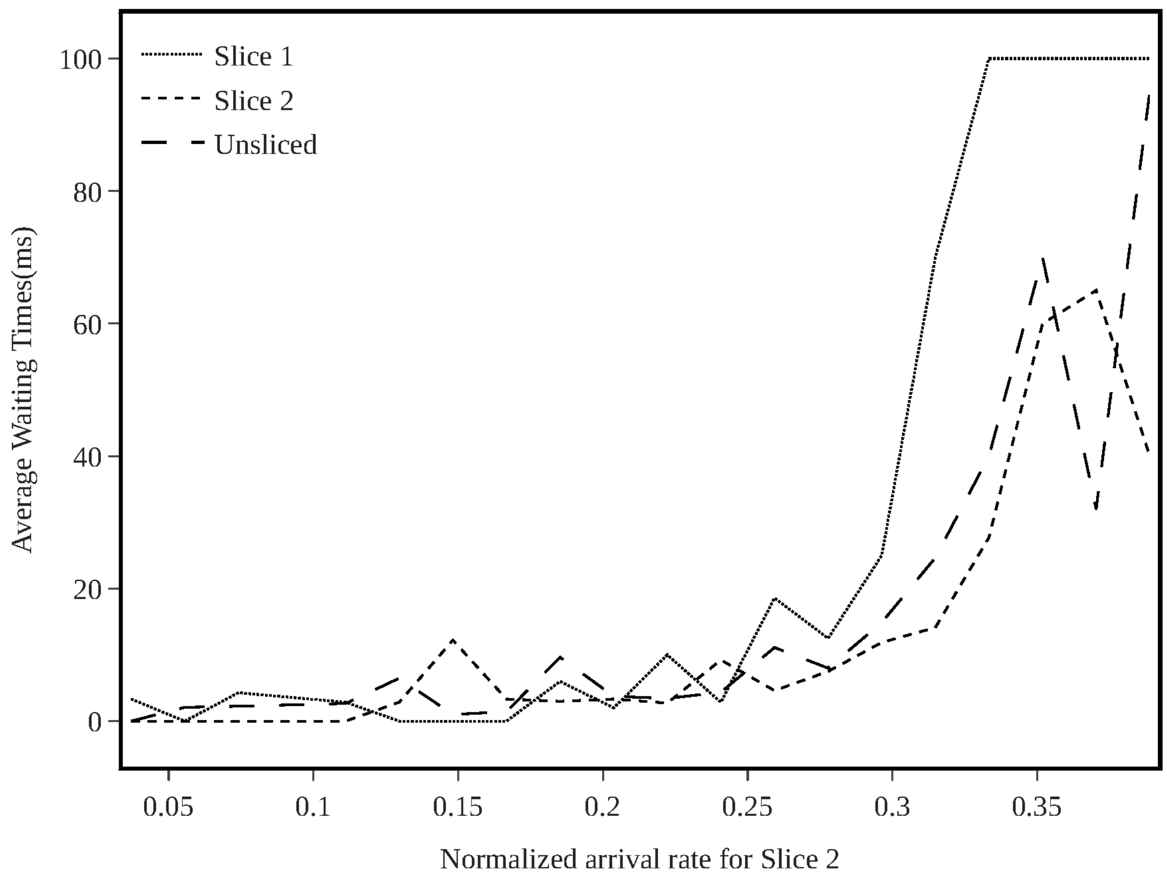


Figure 5.22. *The average waiting time* for 2-slice scenario while the arrival rate of second slice consistently increased using CRF-based NEAT.

reserved to each slice is proportional to their priority levels. Normalized arrival rates of slices are increased starting from the low-priority slice to the high-priority slice. At the end of the simulation, the rates are decreased in the reverse order. The preamble allocation behavior is similar to the 3-slice scenario and the DRL agent successfully prioritizes the slices.

The full load is exceeded when the normalized arrival rate of $5th$ slice increases. In that case, the majority of preambles are reserved for $5th$ slice. λ_1 increases to $2/54$ at $t = 2$, λ_2 increases to $3/54$ at $t = 2.5$, λ_3 increases to $4/54$ at $t = 3$, λ_4 increases to $5/54$ at $t = 3.5$ and λ_5 increases to $7/54$ at $t = 5$ and also reserved preamble count for each slice increases in similar order. Between $t = 3$ and $t = 4$, the total load $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 \approx 0.39$ passes the channel capacity as in the case of the 3-slice scenario when t is between 3 and 8. Unlike the 3-slice scenario, when λ_1 increase, only a tiny change can be observed in m_1. However, for higher priority slices, the method makes major changes on the reserved preamble counts.

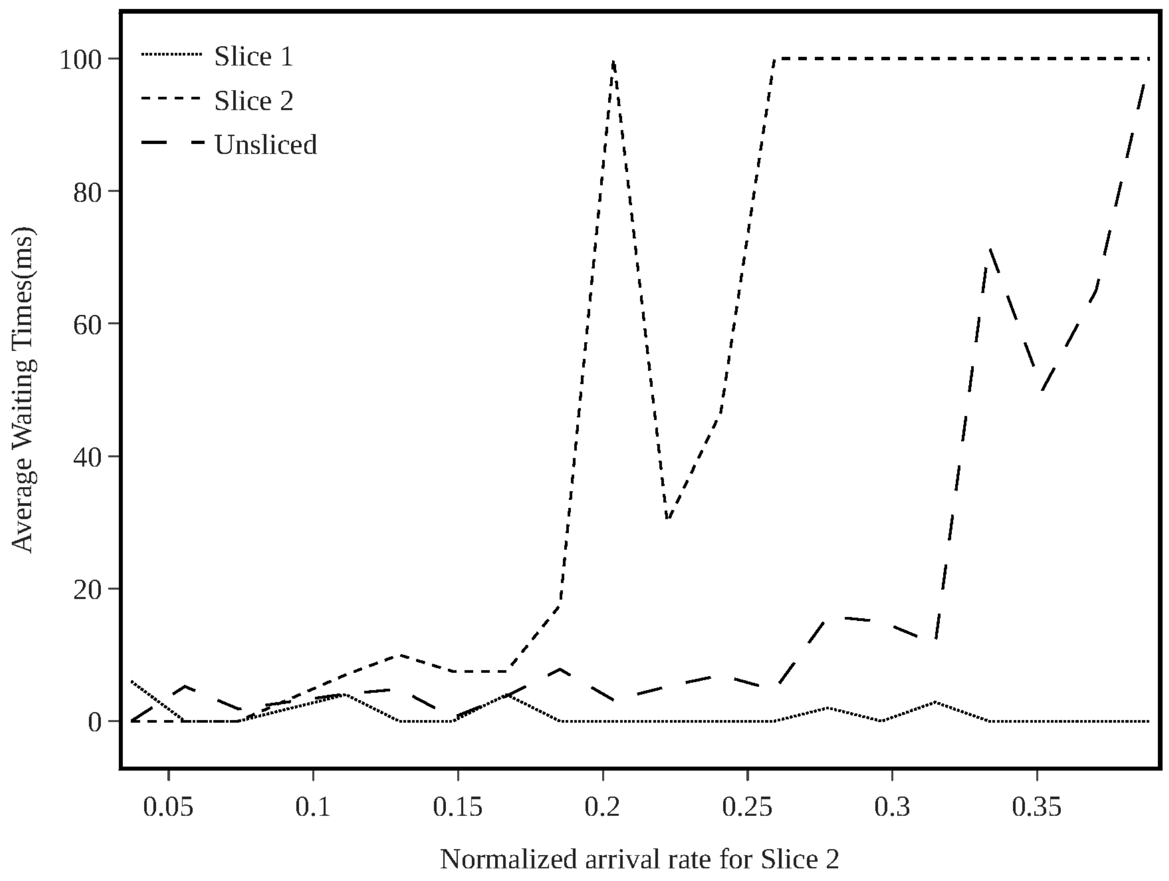


Figure 5.23. *The average waiting time* for 2-slice scenario while the arrival rate of second slice consistently increased using PRF-based NEAT.

This observation can be seen in also collision rate plot. While the line of $1st$ slice walks around 1, the other slices walk between 0.2 and 0.6 likewise to 3-slice. At $t = 7$, $t = 7.5$, $t = 8$, $t = 8.5$ and $t = 9$ normalized arrival rates drop respectively and also reserved preamble counts decreases in the same order. In addition, the ratio of dropped messages to transmitted messages for high priority slices never increases during the whole simulation. For the lowest priority 2 slices, the ratio oscillates between 0.6 and 1.

In Fig. 5.37., from $t = 0$ to $t = 2$, likewise DRL there is no congestion and the preamble allocation of slices does not change. However, unlike DRL, during this period, the number of preambles reserved to each slice is not proportional to their priority levels. Even, there is no logic behind the preamble allocation mechanism up to $t = 3.5$. After, the normalized arrival rate of $4th$ slice, λ_4 increases to $5/54$ at $t = 3.5$. This is the first point that NEAT-based method reacts to the changes in the normalized arrival rates. Although from $t = 3.5$ to $t = 7.5$ NEAT seems to manage prioritize preamble allocations, the high priority slices $3rd$,

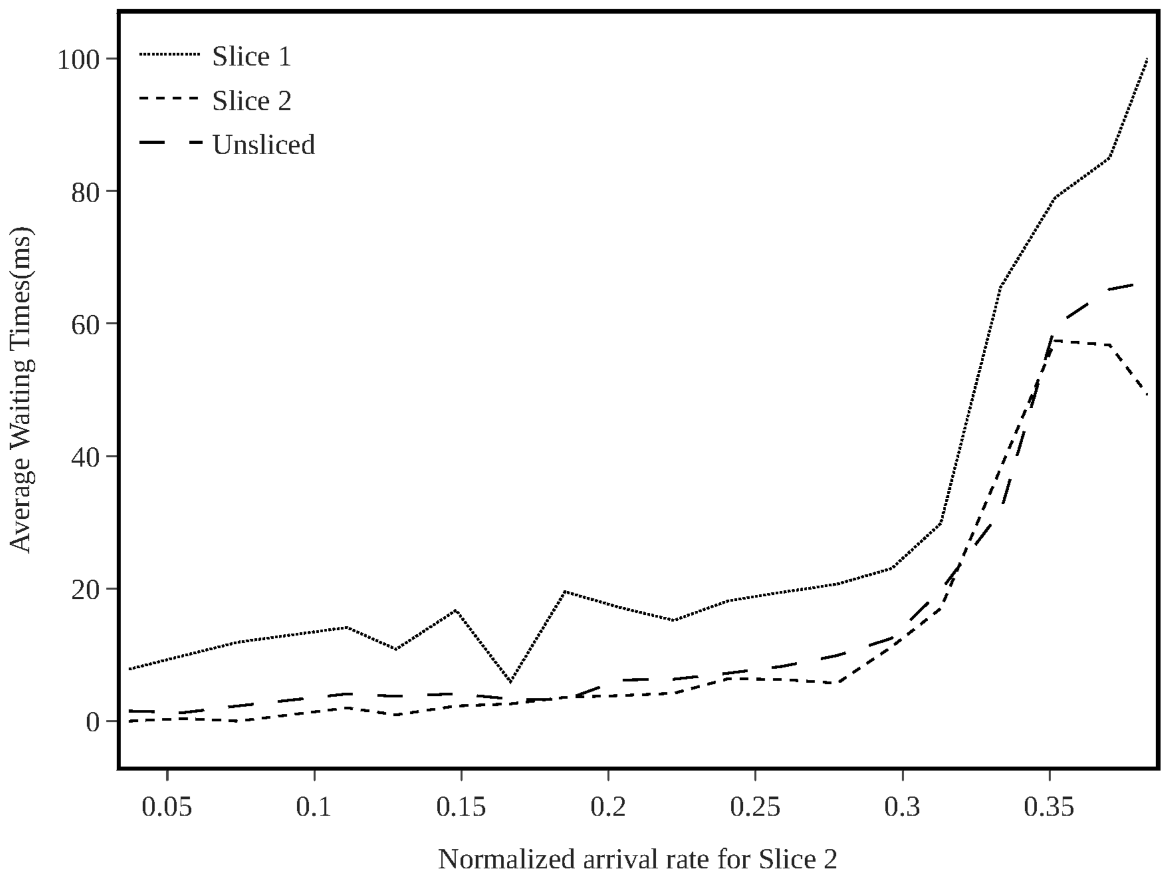


Figure 5.24. *The average waiting time* for 2-slice scenario while the arrival rate of second slice consistently increased using ideal algorithm.

$4th$ and $5th$ slices have the collision rate between 0.6 and 1, also the $3rd$ and $5th$ slices have the ratio of dropped messages to transmitted messages between 0.4 and 1. After $t = 7.5$, the preamble allocations return to the initial position.

Figs. 5.38. and 5.39. plot the timeline simulation of the *ideal algorithm* for 3-slice and 5-slice scenarios respectively. As pointed out above before, the algorithm slightly prioritizes the slices in a way that the collision rates in each figure are very close to each other, hence the allocation of each slice j is slightly better than the allocation of slice $j - 1$. The plots in the time intervals [2,3] and [8,9] for 3-slice in Fig. 5.38. and, in the time intervals [2,4] and [7,9] for 5-slice in Fig. 5.39. show that the *ideal algorithm* can not prevent collisions for lower priority slices when the total load in the network is less than $(1/e \approx 0.37)$.

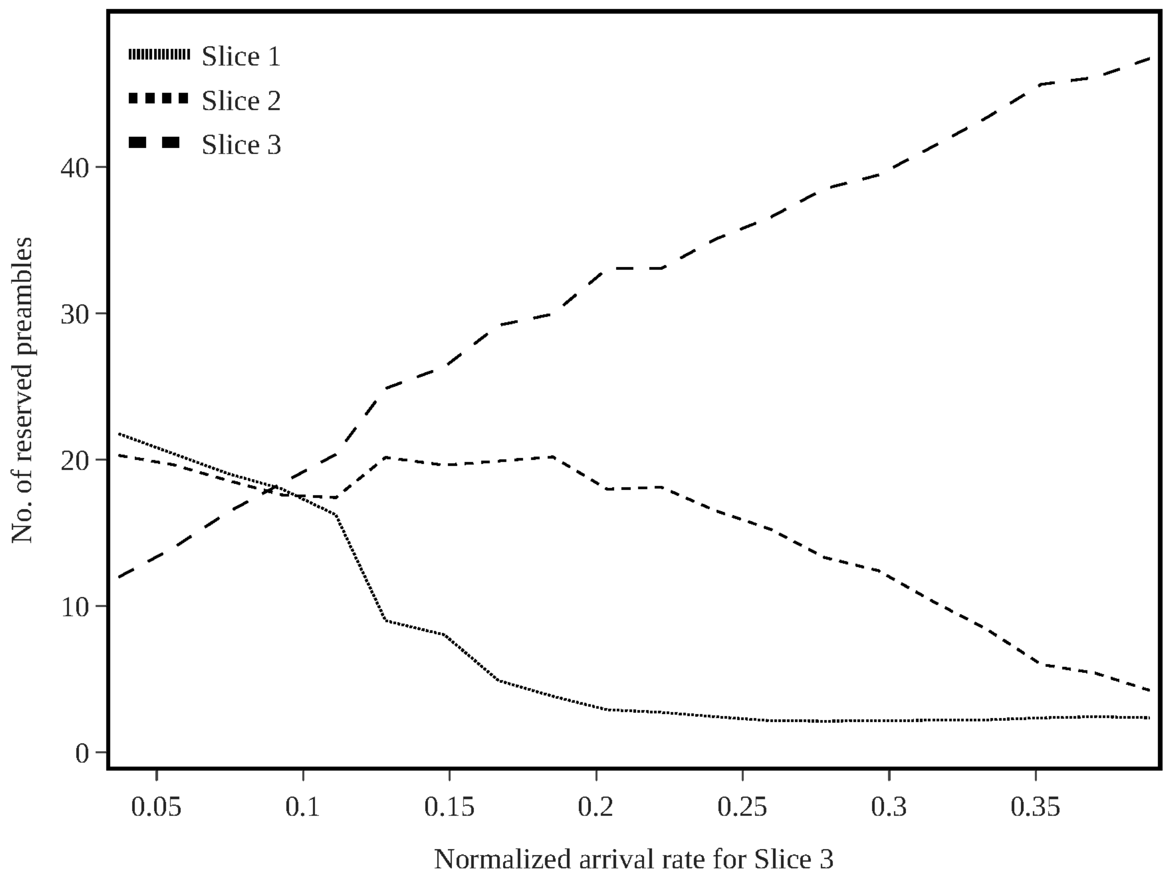


Figure 5.25. *The number of reserved preambles* of the CRF-based DRL for 3 network slices while the arrival rate of third slice consistently increased.

5.6.3. General Discussion

In summary, in timeline simulations, the sudden increase of the number of arrival preamble requests to slices from low-priority to high-priority and the sudden decrease from high-priority to low-priority respectively is tested. Using DRL, for both the 3-slice and 5-slice scenarios, when the traffic of the low-priority slice increases the preamble allocation is increased in favour of the low-priority slice. Following, the traffic to slices increases from lower to higher priority ones respectively and in each increase more preambles are reserved to the slice having lastly increased traffic. The moment when the traffic to high-priority slice increased, the total normalized arrival rate reaches 0.39 so that passes the full load ($1/e \approx 0.37$). After that, the DRL method suddenly allocates the vast majority of preambles to the high-priority slice.

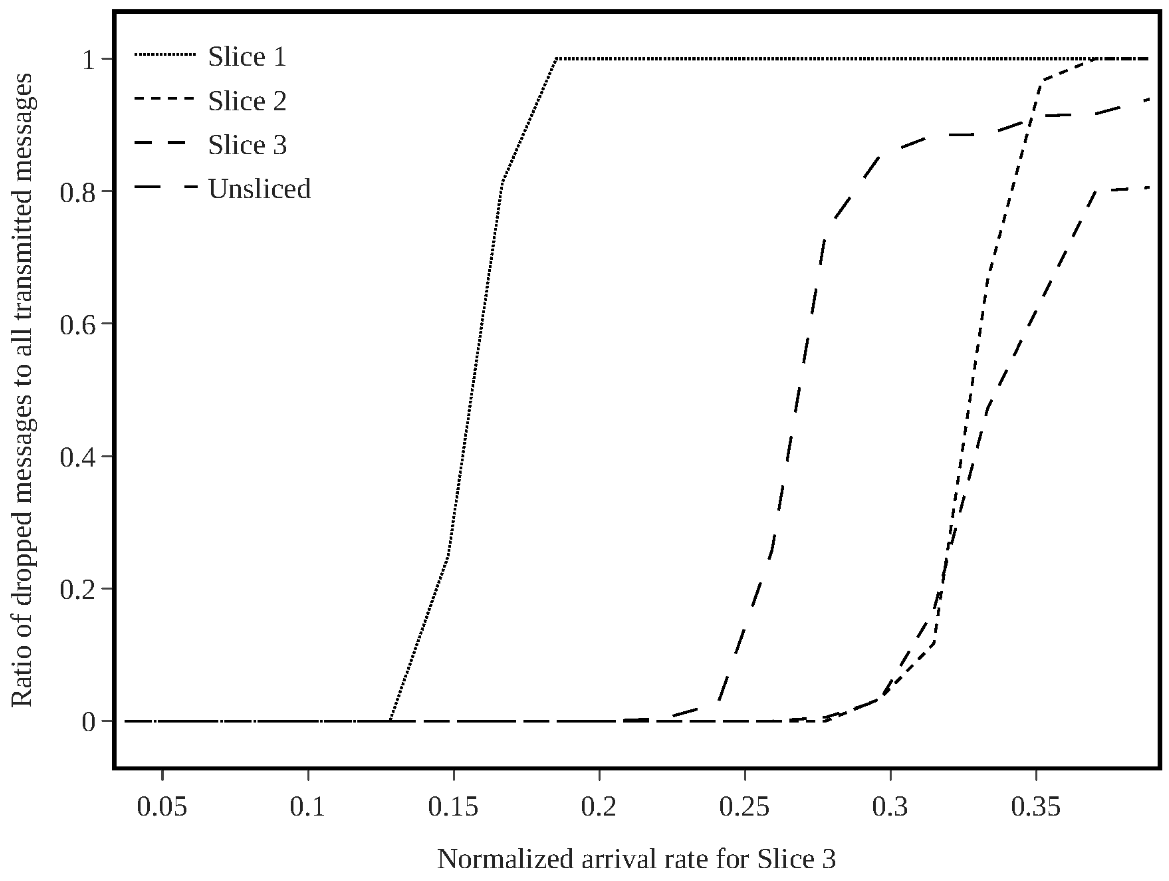


Figure 5.26. *The ratio of dropped messages to all transmitted messages* of the CRF-based DRL for 3 network slices while the arrival rate of third slice consistently increased.

Nonetheless, Fig. 5.34. and 5.35. show that the DRL-based method does not neglect performances of lower priority slices, while the collision rate of the high-priority slice is kept under control. With approaching to the end of the timeline, the traffic assigned to slices decreases starting from the high-priority slice to the low-priority slice respectively. The preamble allocation of the DRL method acts in a reverse direction. In the end, the preamble allocations return to the first position. These results confirm the effectiveness and adaptability of the DRL approach in a dynamic environment. On the contrary, using NEAT, for both 3-slice and 5-slice scenarios high collision rates and drop ratios are observed.

5.7. The Performance of the Method When Comparing with the Method in [1]

The authors in [1] propose to assign weights γ_i to services based on their priorities. In this specific scenario they use two services having priority weights $\gamma_1 = 2$ and $\gamma_2 = 1$.

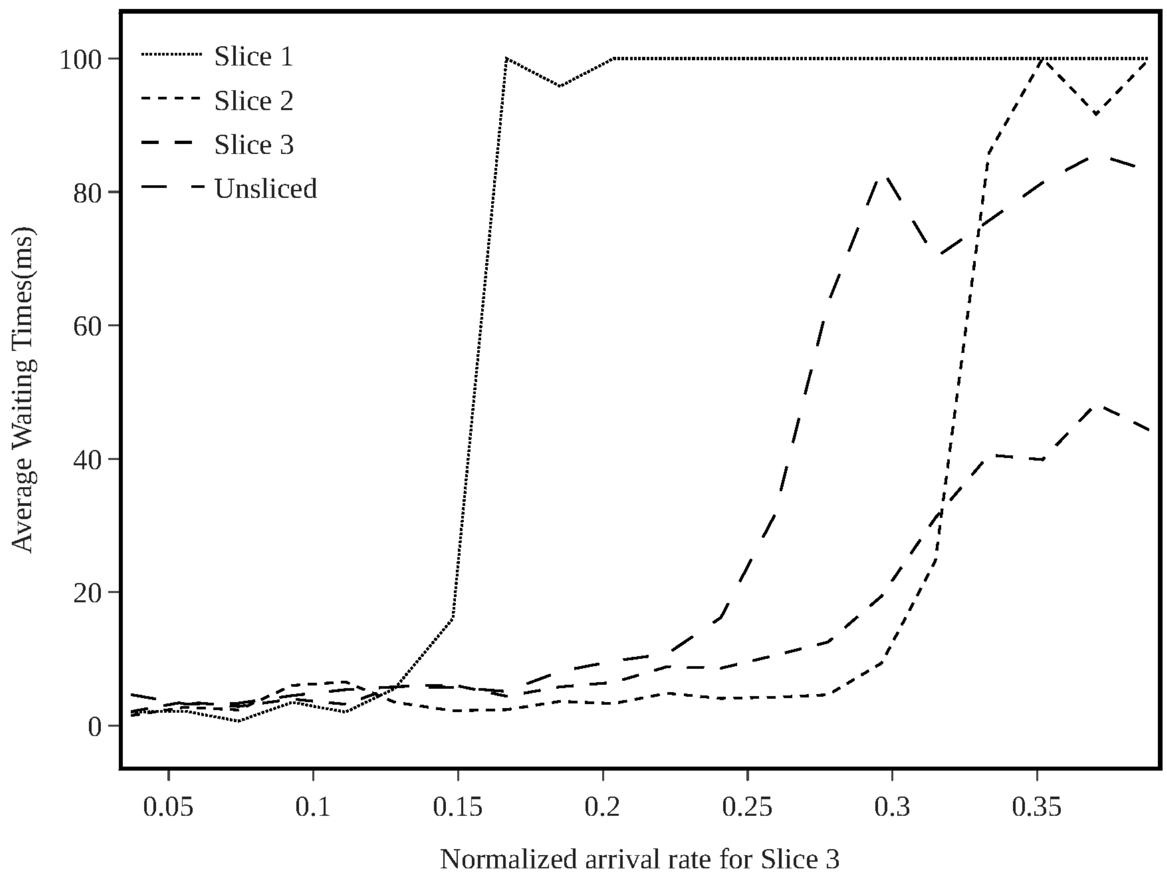


Figure 5.27. *The average waiting time* of the CRF-based DRL for 3 network slices while the arrival rate of third slice consistently increased.

For each service, 10.000 devices are activated over 10 seconds (2000 RAO slots each $5ms$) with uniform arrival for the low-priority service and bursty beta arrival for the high-priority service. They stated that their algorithm produce results close to the ideal case in which the average delay (slots/device) is close to 0. In addition, they specify the average delay for the high-priority service is less than for the low-priority scenario. For a fair comparison, the same scenario is implemented for 2-slice and plotted the results in Figs. 5.40.-5.41.. Please note that the RAO period is taken as $5ms$ here, whereas it is taken as $(10ms)$ in previous experiments. Although the authors use $W = \infty$ and this value is taken as 10 in this current study, since no dropped preamble request is observed in these simulations, W has no effect on comparison.

To sum up, the proposed method can successfully prioritize the slices in a way that the average delay (slots/device) of each slice is close to 0. Moreover, the high-priority slice has lower average delay than the low-priority slice. It is also stated in [1] that they obtain average

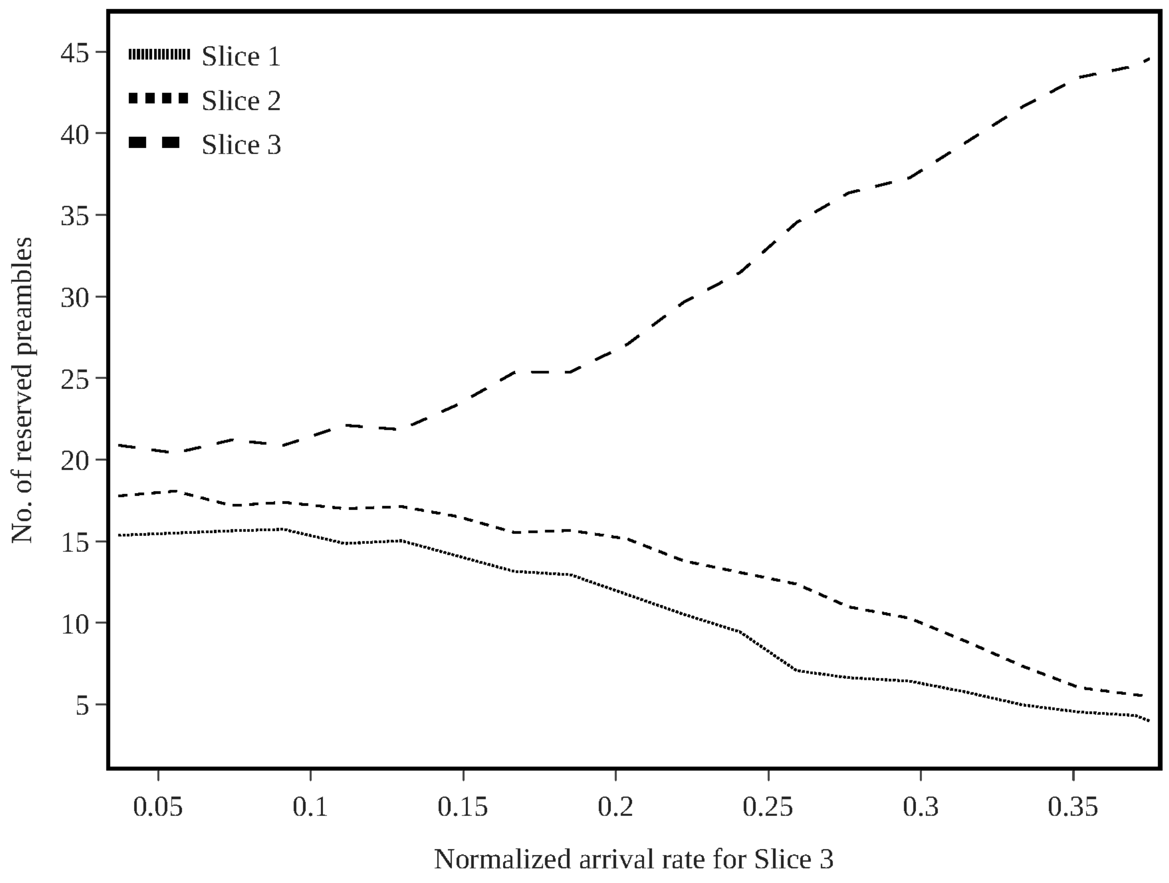


Figure 5.28. *The number of reserved preambles* of the ideal algorithm to 3 network slices while the arrival rate of third slice consistently increased.

delays (slots/device) close to 0 without specifying exact values. Although the *ideal algorithm* manages to keep average delays (slots/device) lower than 1, there is a noticeable difference between the slices (0.94 for the low-priority slice and 0.13 for the high-priority slice). Even the *ideal algorithm* performs slightly better than our proposed approach for the high-priority slice based on the average delay metric, it drops a few number of preamble requests from the low-priority slice. As stated above, the proposed approach does not drop any requests.

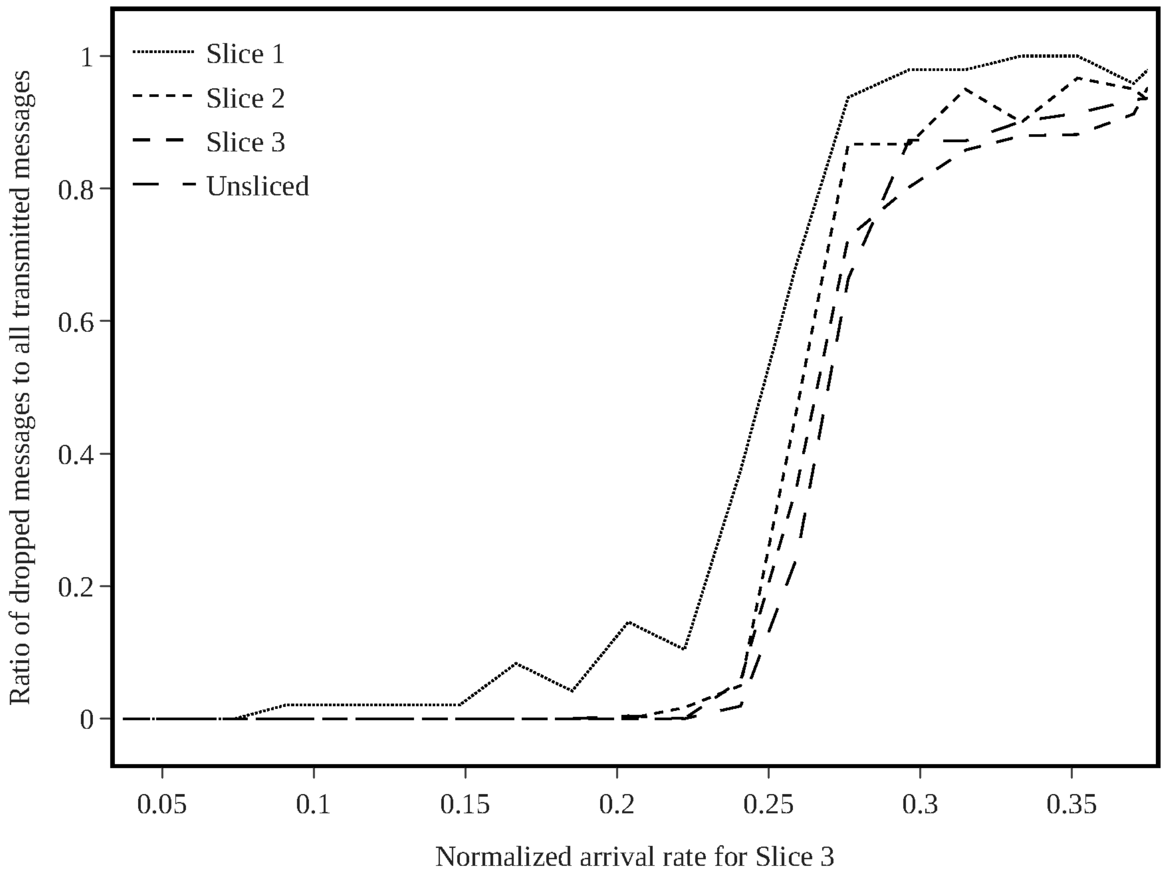


Figure 5.29. *The ratio of dropped messages to all transmitted messages* of the ideal algorithm to 3 network slices while the arrival rate of third slice consistently increased.

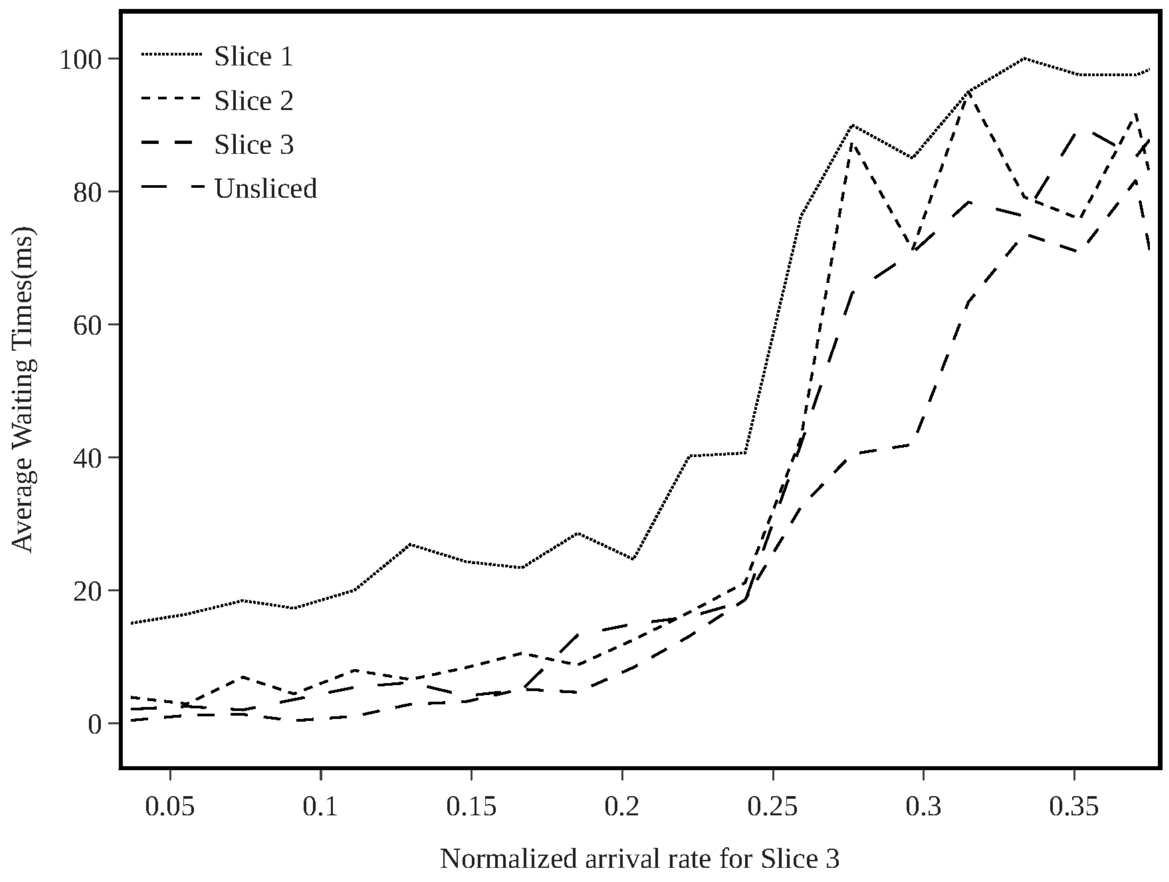


Figure 5.30. *The average waiting time* of the ideal algorithm to 3 network slices while the arrival rate of third slice consistently increased.

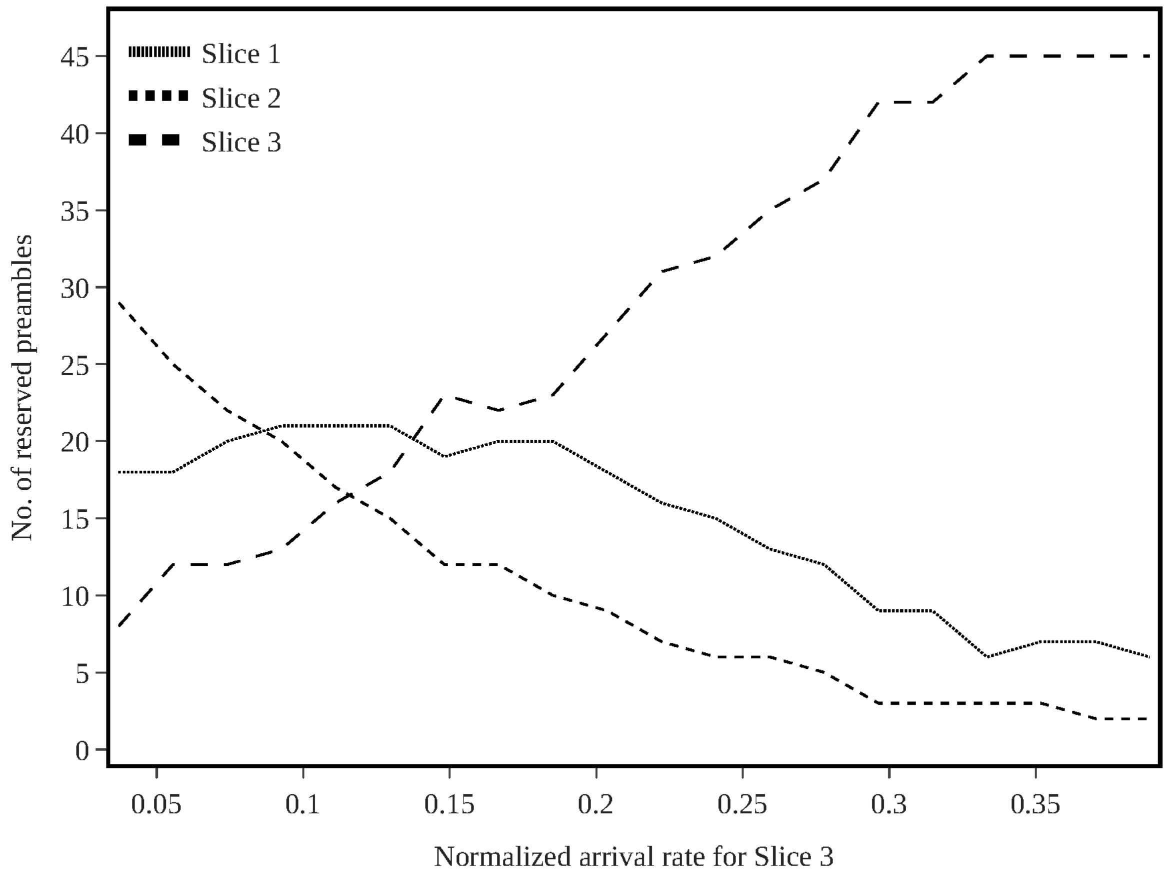


Figure 5.31. *The number of reserved preambles* of the CRF-based NEAT for 3 network slices while the arrival rate of third slice consistently increased.

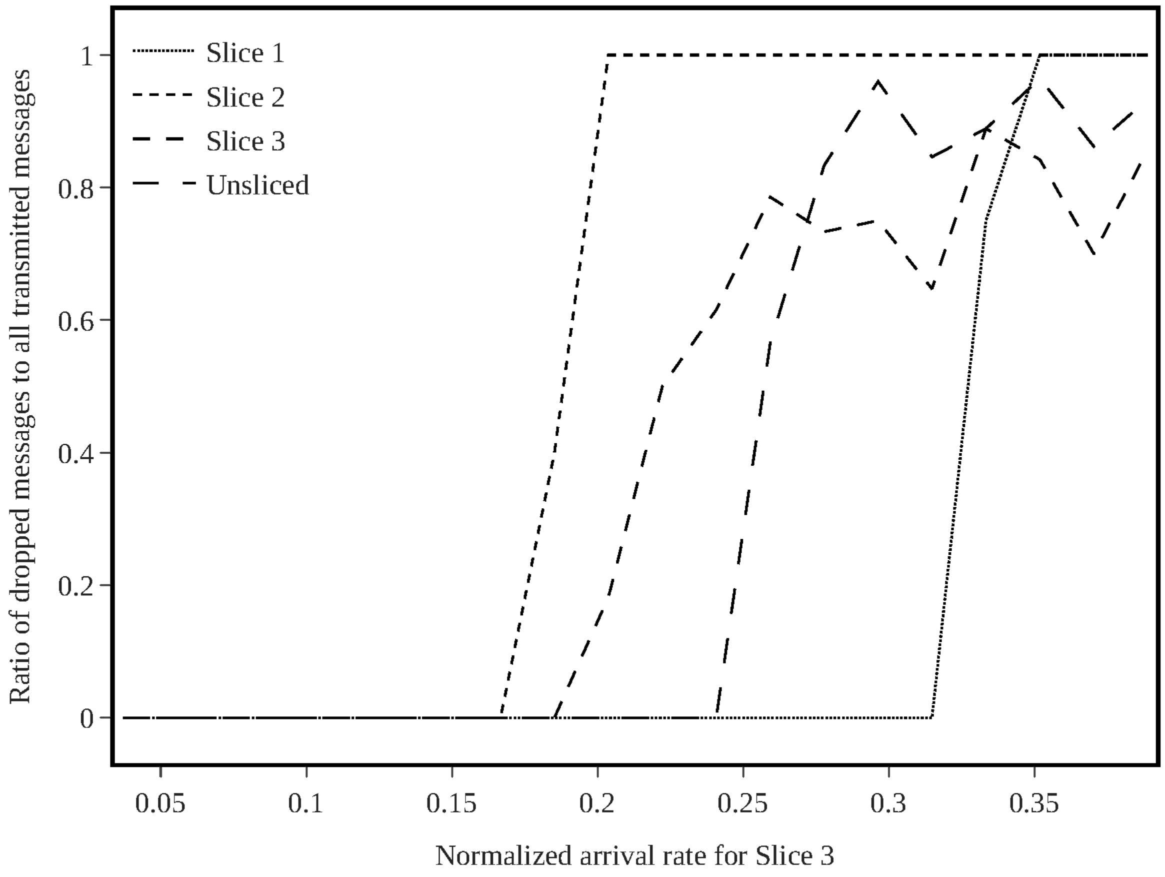


Figure 5.32. *The ratio of dropped messages to all transmitted messages* of the CRF-based NEAT for 3 network slices while the arrival rate of third slice consistently increased.

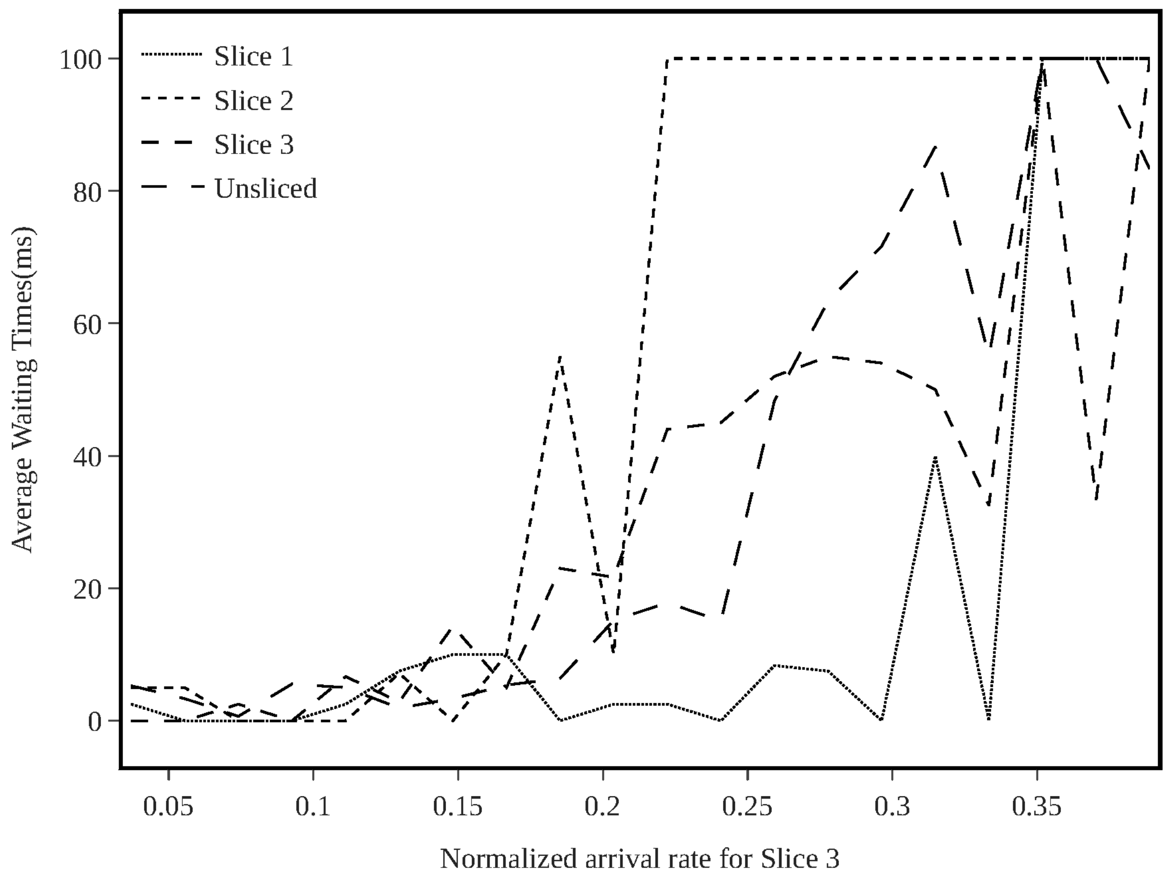


Figure 5.33. *The average waiting time* of the CRF-based NEAT for 3 network slices while the arrival rate of third slice consistently increased.

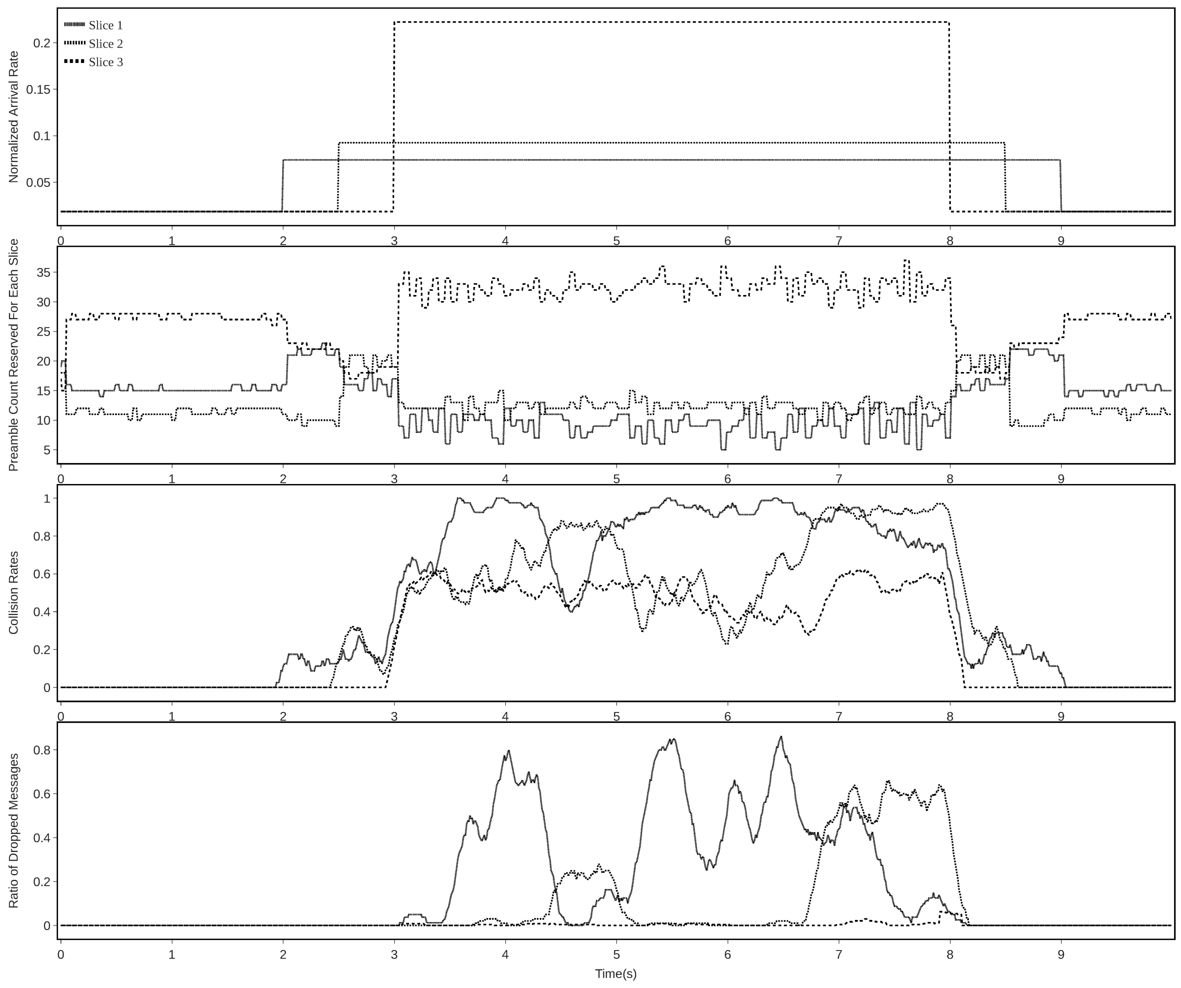


Figure 5.34. The timeline simulation graphs for 3-slice *CRF*-based DRL.

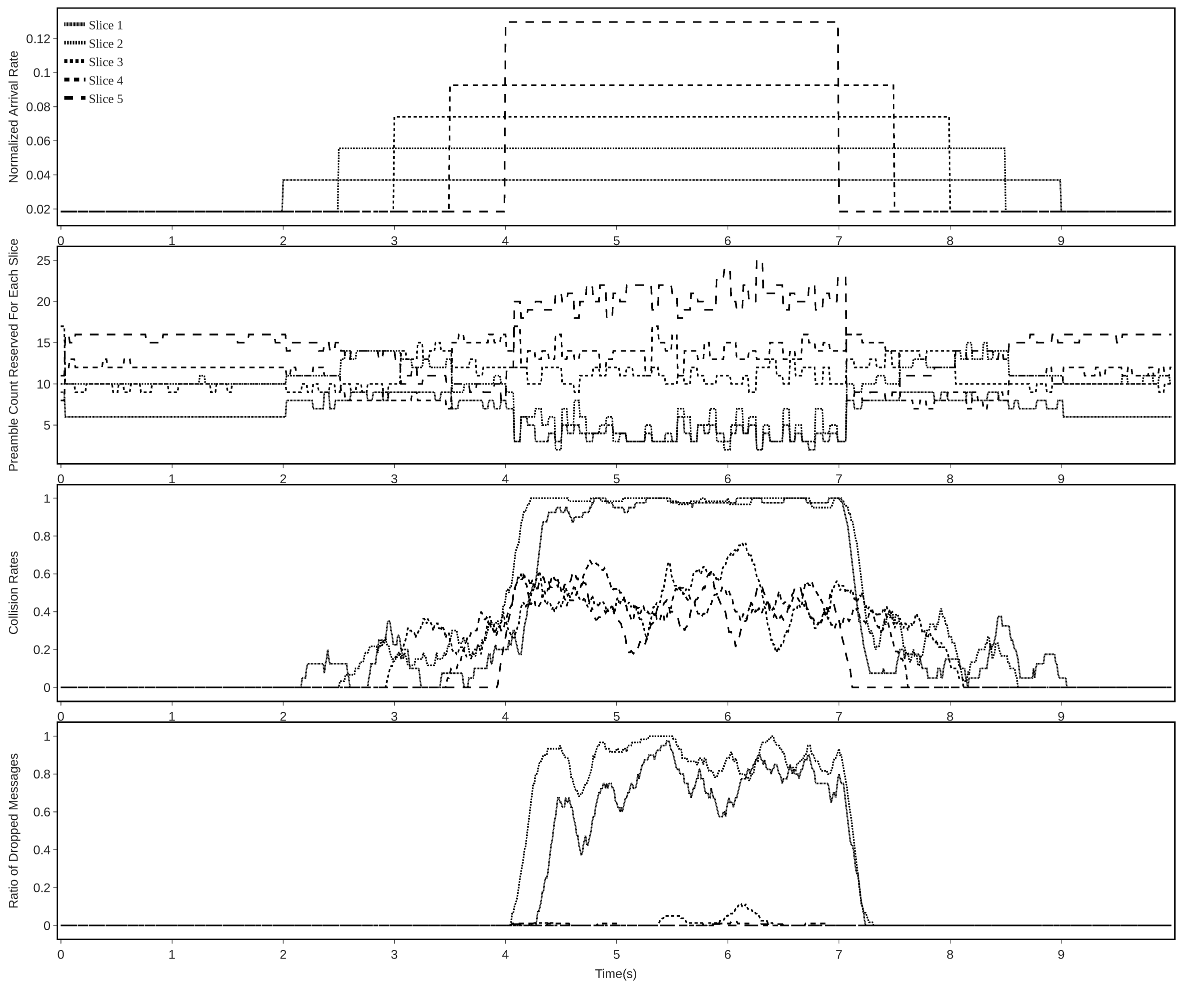


Figure 5.35. The timeline simulation graphs for 5-slice *CRF*-based DRL.

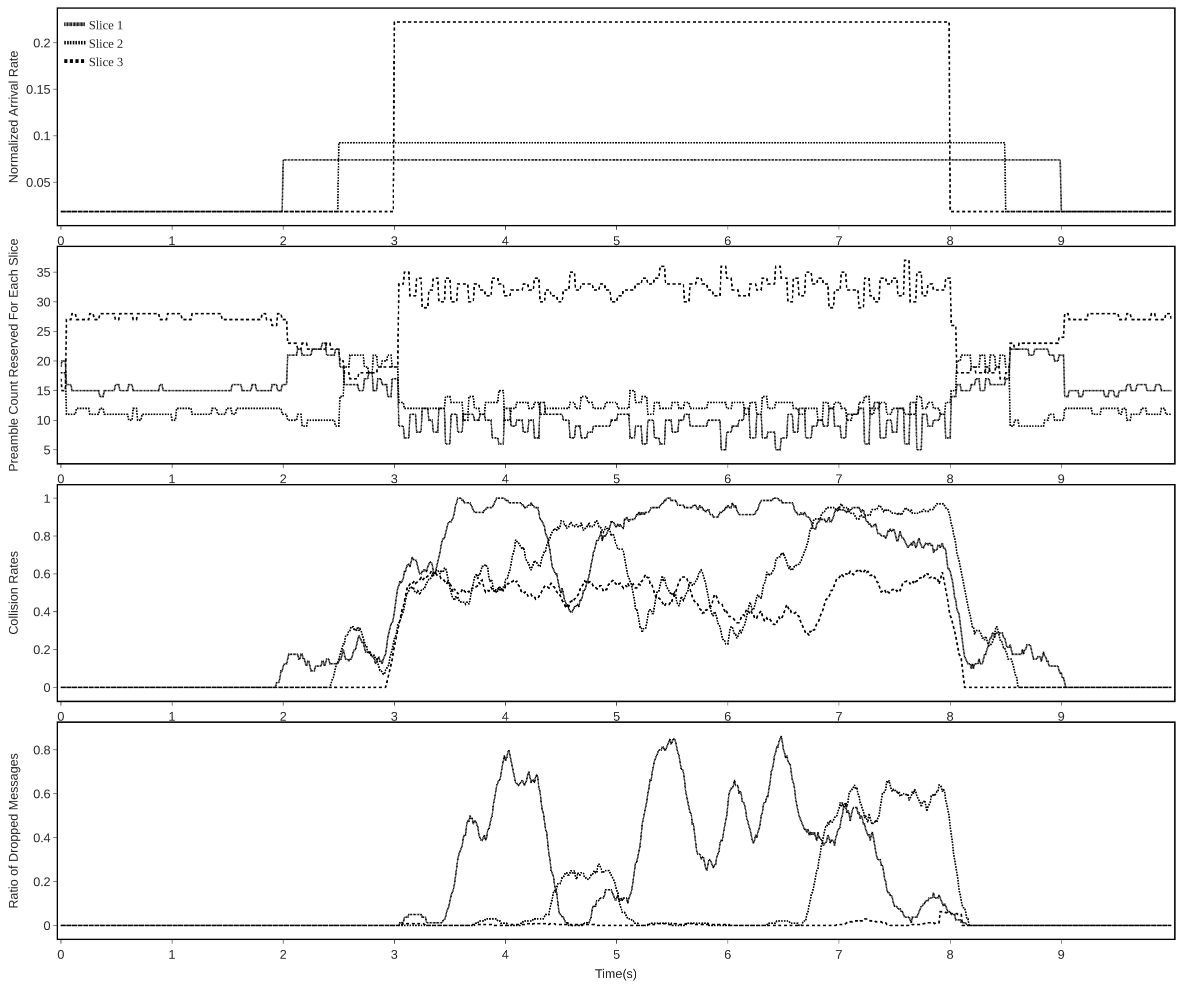


Figure 5.36. The timeline simulation graphs for 3-slice *CRF*-based NEAT.

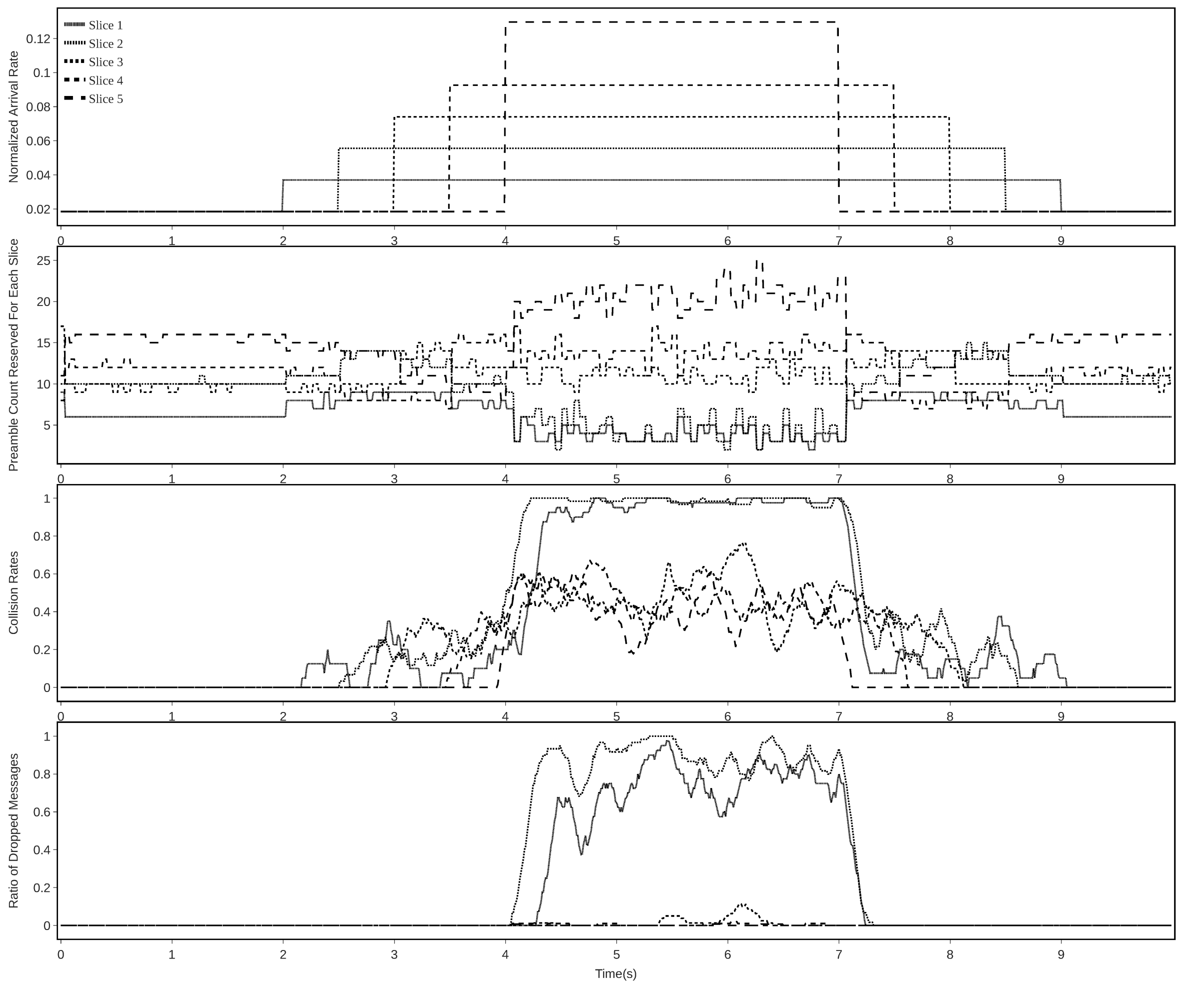


Figure 5.37. The timeline simulation graphs for 5-slice *CRF*-based NEAT.

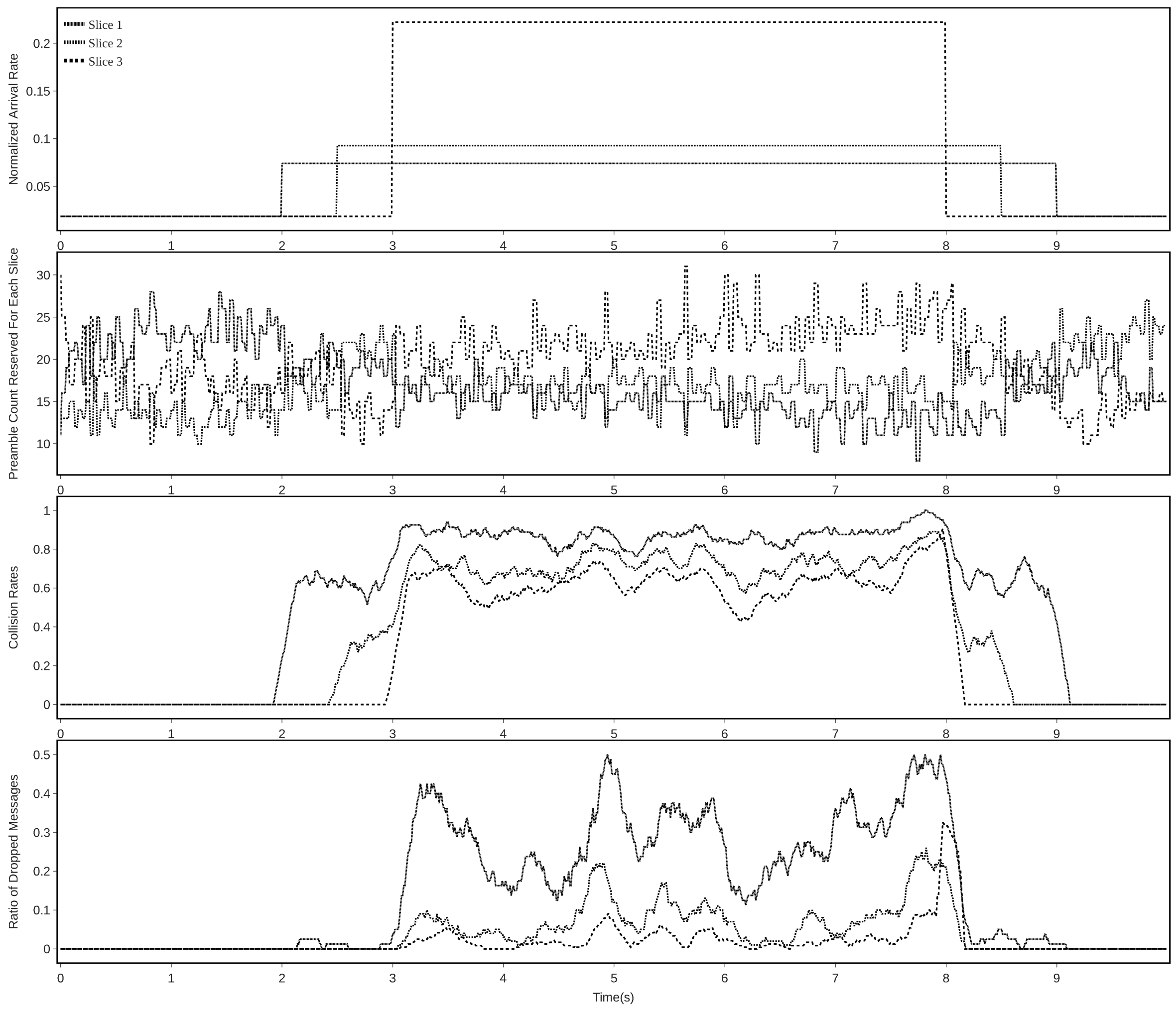


Figure 5.38. The timeline simulation graphs for 3-slice using the ideal algorithm.

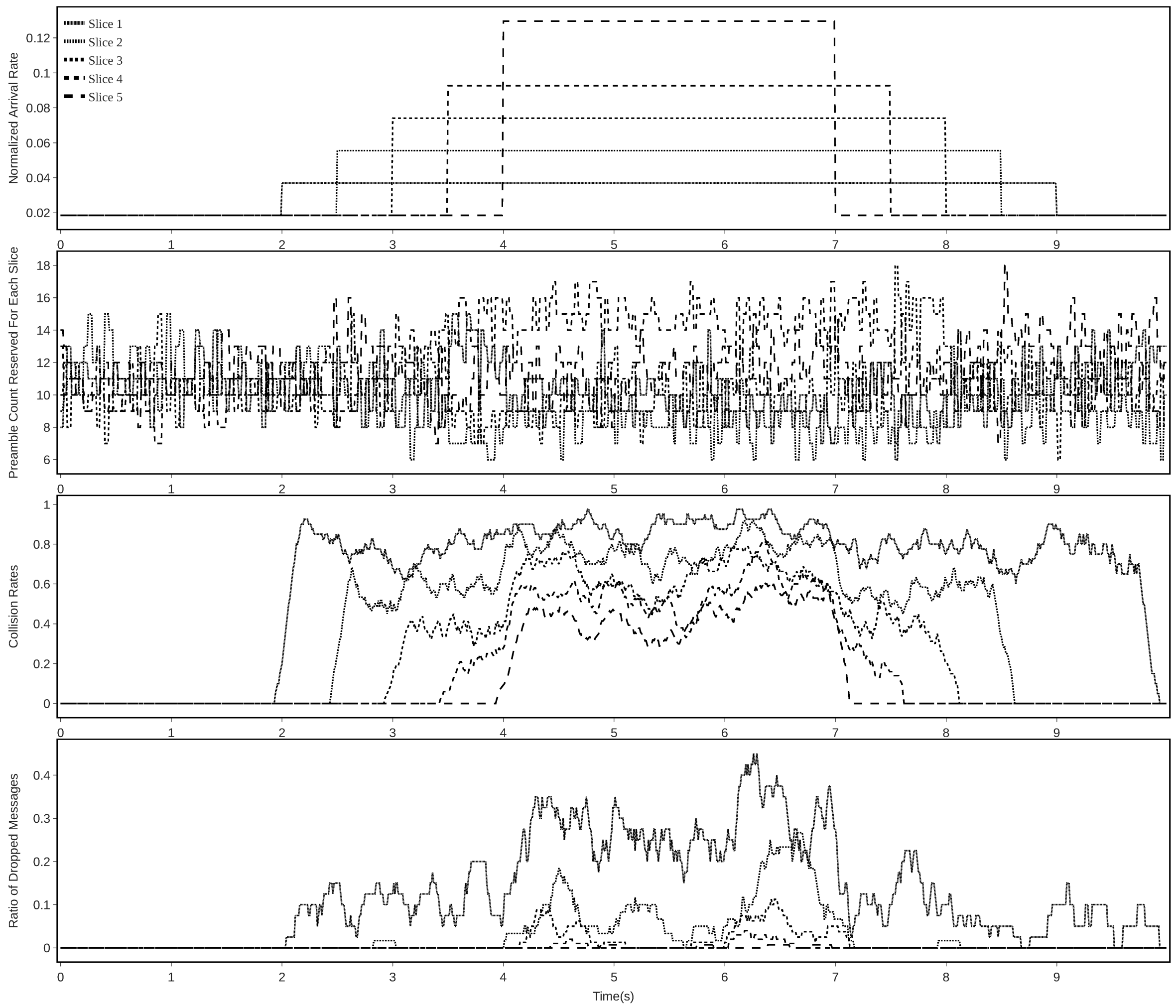


Figure 5.39. The timeline simulation graphs for 5-slice using the ideal algorithm.

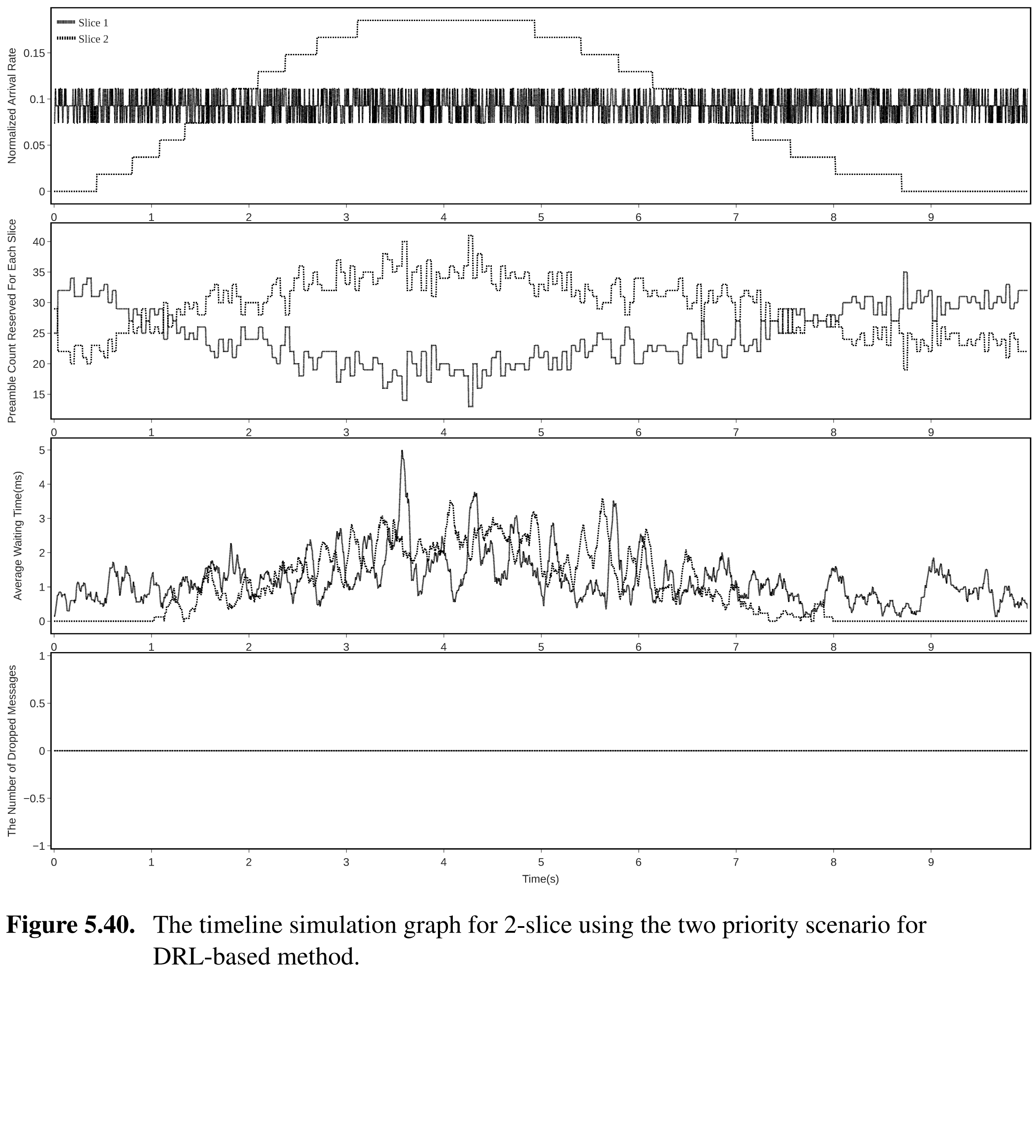


Figure 5.40. The timeline simulation graph for 2-slice using the two priority scenario for DRL-based method.

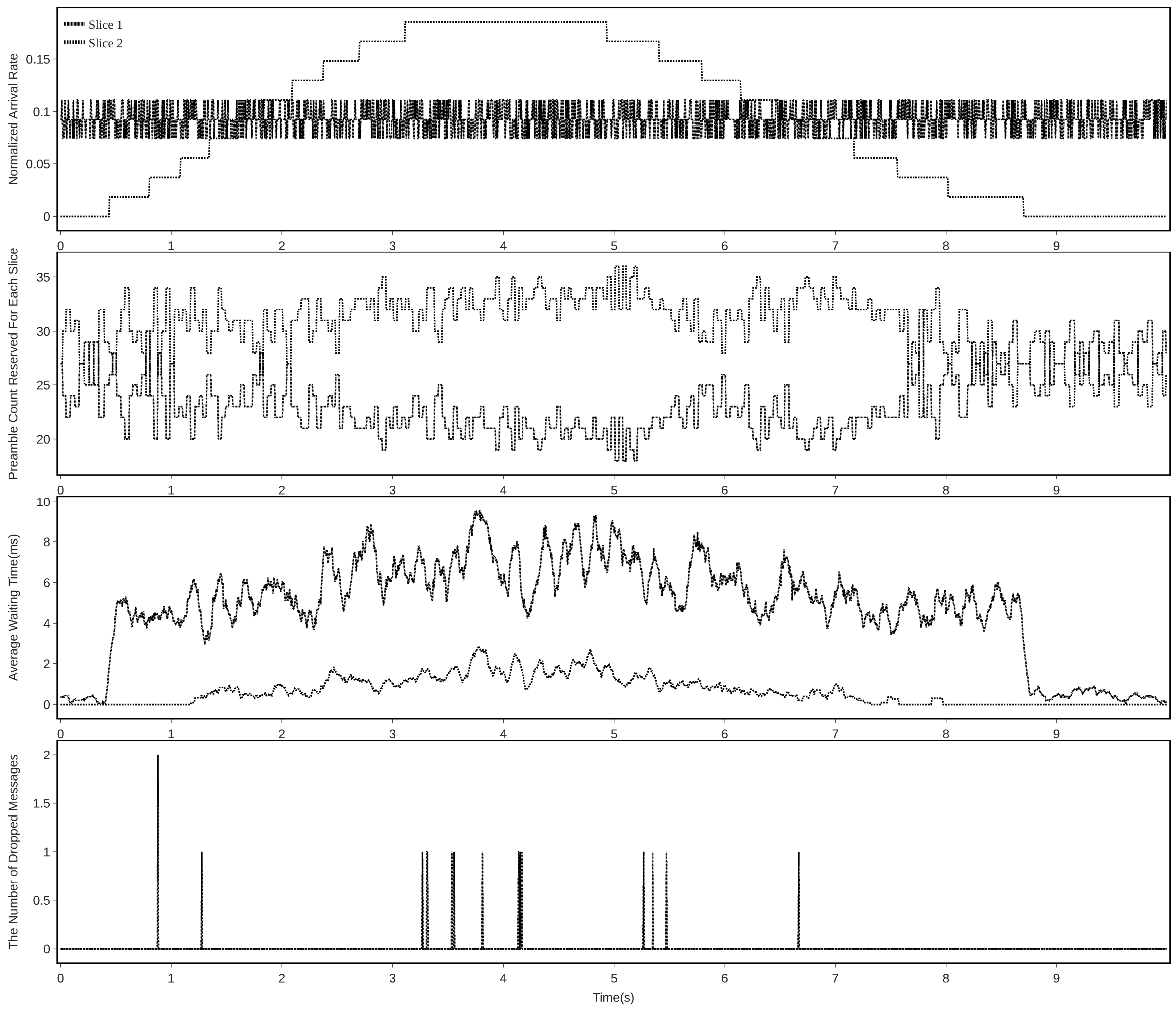


Figure 5.41. The timeline simulation graph for 2-slice using the *ideal algorithm* in [1].

6. CONCLUSION

With the increasing significance of RAN resource allocation in 5G, flexible preamble allocation becomes an important problem. This thesis aims to find a solution to the RAN slicing problem by prioritizing slices. It explores the use of DRL and GA in order to solve the optimum resource allocation problem in RAN slicing. Here, three reward functions are proposed for the DRL formulation and mathematically analyzed. These reward functions are also used as fitness functions in GA.

In the literature, since, there exists at least 3 service types are defined [10] and there are works which propose 5 service types [11], in this thesis, 3-slice and 5-slice scenarios are implemented. *The number of reserved preambles, the average waiting time and the ratio of dropped messages to all transmitted messages* metrics are defined and explained. Unsliced scenario, exhaustive search and *ideal algorithm* benchmarks are used in simulations for comparison. Extensive simulations are conducted to show success or failures of the proposed methods. The results of experiments are first presented and clarified then analyzed using these different metrics and benchmarks, in order to evaluate the performance. For the given metrics and benchmarks detailed graphs are plotted for both methods in same way.

In detail, the proposed approaches are compared with exhaustive search and it is shown that the proposed DRL-based method behaves optimally when compared to the NEAT-based method and overlap with the exhaustive search. While the DRL-based method can successfully distribute the preambles with respect to prioritization levels to the slices such that the collision rates and drop ratios are kept under control in 3-slice and 5-slice scenario, NEAT-based method can not. Yet, the results show NEAT-based dynamic model is also a promising solution. Among the reward functions, it is shown that a reward function which penalizes collisions in addition to rewarding successful preambles gives the best performance.

The analysis results showed that the DRL agent is tend to distribute preambles over groups dynamically when only *CRF* and *PRF* used. Therefore, SRF is eliminated and not used in training. In order to compare other two functions and show their success, the exhaustive search, which yields optimum profile, is used as a comparison basis. The comparison results showed DRL-based method behaves very closely to the exhaustive search for both functions. However, only CRF based NEAT follows the exhaustive search. On the other hand, GA and PRF based NEAT fails to follow exhaustive search.

Later on, for 2-slice and 3-slice scenarios some performance metrics are plotted with graphs to demonstrate the success of the DRL and NEAT against unsliced scenario using. For all metrics the CRF based DRL method performed better than unsliced scenario and successfully prioritize slices. While CRF based NEAT for some metrics performed better than unsliced, it fails distributing the preambles with respect to prioritization levels. In addition, it is observed that *CRF* has better performance in prioritization when compared to *PRF*.

In order to evaluate the proposed models in more complex scenarios using higher number of slices, the timeline simulations are presented to show the flexibility of the proposed methods in the rapid changing environment. Here, the DRL-based method improves the performance of the high-priority slices significantly. Also, it is shown that the both proposed methods adapts to the traffic changes very rapidly by reallocating the preambles to different slices. To sum up, the proposed DRL-based approach is shown to be a suitable approach for RAN slicing by adapting changes in the environment in a timely manner.

6.1. Limitations of the Study

In this thesis, the proposed methods are trained and evaluated by using simulated network traffic. Because, a real word RACH preamble traffic data is not available for training. In addition, it is hard to collect such data since BSs have limited idea about the outside world. Rather, in this thesis, the model is trained by using randomly generated traffic for training due to these reasons. On other hand, having a real world data for different service types which have various QoS may help training of models more precisely.

6.2. Future Work

The proposed methods can be used as layers that can settle above or between other mechanisms. In this thesis, ACB, EAB or any other proposed mechanism is not implemented. These 3GPP proposed methods and their implementations are available to public. As a future study, a multi-layer system model with different mechanisms can be implemented. How these layers interact with each other and integration of them can also be analyzed.

This study uses TRPO for the DRL based method. There are also other deep policy gradient algorithms that can be used for. For example there is Proximal Policy Optimization

(PPO) [110] algorithm which actually perform better than TRPO. While TRPO makes sure the policy will not be updated a lot by using KL-constrain, PPO removes KL penalty and makes adaptive updates. As a future implementation, PPO may be used in DRL for dynamic preamble allocation.

In this thesis, NEAT is used as a secondary model. The exact same reward function and model layout of DRL are used as fitness function and model layout for NEAT, since both NEAT and DRL have ANN layout under the architecture. The low success rate of the NEAT model compared to DRL may be originated using the same exact model layout without any modification. In addition, rather than using a GA method which uses ANNs as solutions, forming a new solution may increase the success rate of GA. For example, there are new approaches which combines evolutionary strategies with RL algorithms like evolutionary reinforcement learning. These evolutionary reinforcement learning methods may be used as future work to extend the success of the evolutionary and RL algorithms.

A APPENDIX : NEAT CONFIGURATION FILE

In this appendix, the configuration file of the NEAT-based model for training is given. The detailed description is given in [123]

Table 1.1. The parameters and their values in the NEAT-python configuration file.

Parameter Name	**Value**
pop_size	200
fitness_criterion	max
fitness_threshold	∞
reset_on_extinction	0
activation_default	relu
activation_mutate_rate	0.0
activation_options	relu
aggregation_default	sum
aggregation_mutate_rate	0.0
aggregation_options	sum
bias_init_mean	0.0
bias_init_stdev	1.0
bias_max_value	30.0
bias_min_value	-30.0
bias_mutate_power	0.5
bias_mutate_rate	0.7
bias_replace_rate	0.1
compatibility_disjoint_coefficient	1.0
compatibility_weight_coefficient	1.0
conn_add_prob	0.9
conn_delete_prob	0.2
enabled_default	True
enabled_mutate_rate	0.01
feed_forward	True

Table 1.2. The parameters and their values in the NEAT-python configuration file contd.

Parameter Name	**Value**
initial_connection	full
node_add_prob	0.9
node_delete_prob	0.2
num_hidden	2
num_inputs	2*J
num_outputs	J
response_init_mean	1.0
response_init_stdev	0.0
response_max_value	30.0
response_min_value	-30.0
response_mutate_power	0.0
response_mutate_rate	0.0
response_replace_rate	0.0
weight_init_mean	0.0
weight_init_stdev	1.0
weight_max_value	30.
weight_min_value	-30.
weight_mutate_power	0.5
weight_mutate_rate	0.8
weight_replace_rate	0.1
compatibility_threshold	3.0
species_fitness_func	max
max_stagnation	20
species_elitism	4
elitism	2
survival_threshold	0.2

REFERENCES

[1] J. Liu, M. Agiwal, M. Qu, and H. Jin. Online control of preamble groups with priority in massive iot networks. *IEEE Journal on Selected Areas in Communications*, pages 1–1, **2020**. doi:10.1109/JSAC.2020.3018964.

[2] Itu-t recommendation m.2083 : Imt vision - framework and overall objectives of the future development of imt for 2020 and beyond. **2015**.

[3] Md. Farhad Hossain, Ayman Mahin, Topojit Debnath, Farjana Mosharrof, and Khondoker Islam. Recent research in cloud radio access network (c-ran) for 5g cellular systems - a survey. *Journal of Network and Computer Applications*, 139, **2019**. doi:10.1016/j.jnca.2019.04.019.

[4] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung. Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges. *IEEE Communications Magazine*, 55(8):138–145, **2017**.

[5] Yasir Mehmood, Carmelita Görg, Maciej Muehleisen, and Andreas Timm-Giel. Mobile m2m communication architectures, upcoming challenges, applications, and future directions. *EURASIP Journal on Wireless Communications and Networking*, 2015:250, **2015**. doi:10.1186/s13638-015-0479-y.

[6] J. Kim, J. Lee, J. Kim, and J. Yun. M2m service platforms: Survey, issues, and enabling technologies. *IEEE Communications Surveys Tutorials*, 16(1):61–76, **2014**. doi:10.1109/SURV.2013.100713.00203.

[7] Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions). `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`. Accessed: 2021-02-01.

[8] Cisco. Cisco visual networking index: Forecast and trends, 2017–2022 white paper. (Visited December 2019) [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[9] G. Wunder, P. Jung, M. Kasparick, T. Wild, F. Schaich, Y. Chen, S. T. Brink, I. Gaspar, N. Michailow, A. Festag, L. Mendes, N. Cassiau, D. Ktenas, M. Dryjanski, S. Pietrzyk, B. Eged, P. Vago, and F. Wiedmann. 5gnow: non-orthogonal, asynchronous waveforms for future mobile applications. *IEEE Communications Magazine*, 52(2):97–105, **2014**. doi:10.1109/MCOM.2014.6736749.

[10] Patrick Marsch, Ömer Bulakci, Icaro Silva, Paul Arnold, Nico Bayer, Jakob Belschner, Thomas Rosowski, Gerd Zimmermann, Mårten Ericson, Alexandros Kaloxylos, Panagiotis Spapis, Ahmed Ibrahim, Yang Yang, Shubhranshu Singh, Haris Celik, Jens Gebert, Athul Prasad, Fernando Moya, Mikko Säily, and J.F. Monserrat. D2.2 draft overall 5g ran design, **2016**. doi:10.13140/RG.2.2.17831.14245.

[11] S. Vural, N. Wang, P. Bucknell, G. Foster, R. Tafazolli, and J. Muller. Dynamic preamble subset allocation for ran slicing in 5g networks. *IEEE Access*, 6:13015–13032, **2018**. doi:10.1109/ACCESS.2018.2800661.

[12] Wei Tian, Mingxing Fan, Cheng Zeng, Yajun Liu, Da He, and Qi Zhang. Telerobotic spinal surgery based on 5g network: The first 12 cases. *Neurospine*, 17:114–120, **2020**. doi:10.14245/ns.1938454.227.

[13] M. Vilgelm, S. Schiessl, H. Al-Zubaidy, W. Kellerer, and J. Gross. On the reliability of lte random access: Performance bounds for machine-to-machine burst resolution time. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. **2018**. doi:10.1109/ICC.2018.8422323.

[14] Lilatul Ferdouse, Alagan Anpalagan, and Sudip Misra. Congestion and overload control techniques in massive m2m systems: a survey. *Trans. Emerging Telecommunications Technologies*, 28, **2017**.

[15] Andrea Biral, Marco Centenaro, Andrea Zanella, Lorenzo Vangelista, and Michele Zorzi. The challenges of m2m massive access in wireless cellular networks. *Digital Communications and Networks*, 1(1):1 – 19, **2015**. ISSN 2352-8648. doi:https://doi.org/10.1016/j.dcan.2015.02.001.

[16] Fayezeh Ghavimi and Hsiao-Hwa Chen. M2m communications in 3gpp lte/lte-a networks: Architectures, service requirements, challenges, and applications.

Communications Surveys Tutorials, IEEE, 17:525–549, **2015**. doi:10.1109/COMST.2014.2361626.

[17] K. Ganesan, P. B. Mallick, J. Löhr, D. Karampatsis, and A. Kunz. 5g v2x architecture and radio aspects. In *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6. **2019**. doi:10.1109/CSCN.2019.8931319.

[18] Dajie Jiang and Guangyi Liu. *An Overview of 5G Requirements*, pages 3–26. **2017**. ISBN 978-3-319-34206-1. doi:10.1007/978-3-319-34208-5_1.

[19] 5G Initiative Team. 5g white paper v1. `https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf`, **2015**.

[20] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *IEEE Access*, 6:55765–55779, **2018**. ISSN 2169-3536. doi:10.1109/ACCESS.2018.2872781.

[21] P. Marsch, I. Da Silva, O. Bulakci, M. Tesanovic, S. E. El Ayoubi, T. Rosowski, A. Kaloxylos, and M. Boldi. 5g radio access network architecture: Design guidelines and key considerations. *IEEE Communications Magazine*, 54(11):24–32, **2016**.

[22] K. Samdanis and T. Taleb. The road beyond 5g: A vision and insight of the key technologies. *IEEE Network*, 34(2):135–141, **2020**.

[23] P. Arnold, N. Bayer, J. Belschner, and G. Zimmermann. 5g radio access network architecture based on flexible functional control / user plane splits. In *2017 European Conference on Networks and Communications (EuCNC)*, pages 1–5. **2017**.

[24] Y. L. Lee, J. Loo, T. C. Chuah, and L. Wang. Dynamic network slicing for multitenant heterogeneous cloud radio access networks. *IEEE Transactions on Wireless Communications*, 17(4):2146–2161, **2018**.

[25] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar. A network slicing prototype for a flexible cloud radio access network. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–4. **2018**.

[26] Stefan Rommer, Peter Hedman, Magnus Olsson, Lars Frid, Shabnam Sultana, and Catherine Mulligan. Chapter 3 - architecture overview. In Stefan Rommer, Peter Hedman, Magnus Olsson, Lars Frid, Shabnam Sultana, and Catherine Mulligan, editors, *5G Core Networks*, pages 15 – 72. Academic Press, **2020**. ISBN 978-0-08-103009-7. doi:https://doi.org/10.1016/B978-0-08-103009-7.00003-X.

[27] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, 19(3):1657–1681, **2017**.

[28] A. Ksentini and N. Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, **2017**.

[29] 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*, 20(3):2429–2453, **2018**.

[30] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, **2017**. doi:10.1109/MCOM.2017.1600935.

[31] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G, LTE-Advanced Pro and The Road to 5G, Third Edition*. Academic Press, Inc., USA, 3rd edition, **2016**. ISBN 0128045752.

[32] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, **1996**. ISSN 1076-9757.

[33] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, **2017**. doi:10.1109/MSP.2017.2743240.

[34] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, **2013**. doi:10.1177/0278364913495721.

[35] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, **2012**. doi:10.1109/TSMCC.2012.2218595.

[36] S. Amari and S. C. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 2, pages 1213–1216 vol.2. **1998**. doi:10.1109/ICASSP.1998.675489.

[37] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, **2012**. doi:10.1109/TSMCC.2012.2218595.

[38] S. Kakade. A natural policy gradient. In *NIPS*. **2001**.

[39] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, **2015**.

[40] Charalampos Andriotis and Konstantinos Papakonstantinou. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering System Safety*, 191, **2019**. doi: 10.1016/j.ress.2019.04.036.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, **2015**. doi:10.1038/nature14236.

[42] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham,

Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, **2016**.

[43] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. **2017**. doi:10.1109/ICRA.2017.7989385.

[44] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. **2017**.

[45] Yuxi Li. Deep reinforcement learning: An overview. **2017**.

[46] Anita Thengade and Rucha Dondal. Genetic algorithm – survey paper. *IJCA Proc National Conference on Recent Trends in Computing, NCRTC*, 5, **2012**.

[47] J. R. Koza. Survey of genetic algorithms and genetic programming. In *Proceedings of WESCON'95*, pages 589–. **1995**. doi:10.1109/WESCON.1995.485447.

[48] Matheus Cordeiro, Paulo Serafim, Yuri Nogueira, Creto Vidal, and Joaquim Cavalcante-Neto. A minimal training strategy to play flappy bird indefinitely with neat. **2019**. doi:10.1109/SBGames.2019.00014.

[49] Iaroslav Omelianenko. The efficiency comparison of neat libraries when looking for double pole-balancing controller. **2019**.

[50] Kenneth Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. volume 2, pages 1757–1762. **2002**. ISBN 0-7803-7282-4. doi:10.1109/CEC.2002.1004508.

[51] Neat framework. `https://neat-python.readthedocs.io/en/latest/neat_overview.html`. Accessed: 2020-05-03.

[52] Patrick Marsch, Ömer Bulakci, Icaro Silva, Paul Arnold, Nico Bayer, Jakob Belschner, Thomas Rosowski, Gerd Zimmermann, Mårten Ericson, Alexandros Kaloxylos, Panagiotis Spapis, Ahmed Ibrahim, Yang Yang, Shubhranshu Singh, Haris Celik, Jens Gebert, Athul Prasad, Fernando Moya, Mikko Säily, and J.F. Monserrat. D2.2 draft overall 5g ran design, **2016**. doi:10.13140/RG.2.2.17831.14245.

[53] W. T. Toor and H. Jin. Comparative study of access class barring and extended access barring for machine type communications. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 604–609. **2017**. doi:10.1109/ICTC.2017.8191051.

[54] N. Zangar, S. Gharbi, and M. Abdennebi. Service differentiation strategy based on macb factor for m2m communications in lte-a networks. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 693–698. **2016**. doi:10.1109/CCNC.2016.7444864.

[55] Ning Li, Chao Cao, and Cong Wang. Dynamic resource allocation and access class barring scheme for delay-sensitive devices in machine to machine (m2m) communications. *Sensors*, 17(6):1407, **2017**. ISSN 1424-8220. doi:10.3390/s17061407.

[56] T. Lin, C. Lee, J. Cheng, and W. Chen. Prada: Prioritized random access with dynamic access barring for mtc in 3gpp lte-a networks. *IEEE Transactions on Vehicular Technology*, 63(5):2467–2472, **2014**. doi:10.1109/TVT.2013.2290128.

[57] H. Althumali and M. Othman. A survey of random access control techniques for machine-to-machine communications in lte/lte-a networks. *IEEE Access*, 6:74961–74983, **2018**. doi:10.1109/ACCESS.2018.2883440.

[58] 3GPP. Backoff enhancements for ran overload control. *TR R2-112863*, **2011**.

[59] 3GPP. Separate backoff scheme for mtc 3gpp tsg-ran2. *TR R2-103776*, **2010**.

[60] 3GPP. Rach overload solutions. *TR R2-103742*, **2010**.

[61] A. Laya, L. Alonso, and J. Alonso-Zarate. Is the random access channel of lte and lte-a suitable for m2m communications? a survey of alternatives. *IEEE Communications Surveys Tutorials*, 16(1):4–16, **2014**. doi:10.1109/SURV.2013.111313.00244.

[62] H. D. Althumali, M. Othman, N. K. Noordin, and Z. M. Hanapi. Dynamic backoff collision resolution for massive m2m random access in cellular iot networks. *IEEE Access*, 8:201345–201359, **2020**. doi:10.1109/ACCESS.2020.3036398.

[63] S. K. Sharma and X. Wang. Collaborative distributed q-learning for rach congestion minimization in cellular iot networks. *IEEE Communications Letters*, 23(4):600–603, **2019**. doi:10.1109/LCOMM.2019.2896929.

[64] 3GPP. Study on ran improvements for machine type communications. *TR 37.868 V11.0.0*, **2011**.

[65] Osama Arouk and Adlen Ksentini. Multi-channel slotted aloha optimization for machine-type-communication. In *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '14, pages 119–125. ACM, New York, NY, USA, **2014**. ISBN 978-1-4503-3030-5. doi:10.1145/2641798.2641802.

[66] Kwang-Ryul Jung, Aesoon Park, and Sungwon Lee. Machine-type-communication (mtc) device grouping algorithm for congestion avoidance of mtc oriented lte network. *Communications in Computer and Information Science Security-Enriched Urban Computing and Smart Grid*, page 167–178, **2010**. doi:10.1007/978-3-642-16444-6_22.

[67] Ki-Dong Lee, Sang Kim, and Byung Yi. Throughput comparison of random access methods for M2m service over LTE networks. In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 373–377. **2011**. doi:10.1109/GLOCOMW.2011.6162474. ISSN: 2166-0077.

[68] K. Lee, M. Reisslein, K. Ryu, and S. Kim. Handling randomness of multi-class random access loads in lte-advanced network supporting small data applications. In *2012 IEEE Globecom Workshops*, pages 436–440. **2012**. doi:10.1109/GLOCOMW.2012.6477612.

[69] C. Kalalas, F. Vazquez-Gallego, and J. Alonso-Zarate. Handling mission-critical communication in smart grid distribution automation services through lte. In *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 399–404. **2016**. doi:10.1109/SmartGridComm.2016.7778794.

[70] X. Zhao, J. Zhai, and G. Fang. An access priority level based random access scheme for qos guarantee in td-lte-a systems. In *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, pages 1–5. **2014**. doi:10.1109/VTCFall.2014.6965863.

[71] Mikhail Vilgelm, Murat Gürsu, Wolfgang Kellerer, and Martin Reisslein. Latmapa: Load-adaptive throughput-maximizing preamble allocation for prioritization in 5g random access. *IEEE Access*, PP:1–1, **2017**. doi:10.1109/ACCESS.2017.2651170.

[72] J. Choi. On the adaptive determination of the number of preambles in rach for mtc. *IEEE Communications Letters*, 20(7):1385–1388, **2016**. doi:10.1109/LCOMM.2016.2546238.

[73] N. K. Pratas, H. Thomsen, Č. Stefanović, and P. Popovski. Code-expanded random access for machine-type communications. In *2012 IEEE Globecom Workshops*, pages 1681–1686. **2012**. doi:10.1109/GLOCOMW.2012.6477838.

[74] Diego Pacheco-Paramo, Luis Tello-Oquendo, Vicent Pla, and Jorge Martinez-Bauset. Deep reinforcement learning mechanism for dynamic access control in wireless networks handling mmtc. *Ad Hoc Networks*, 94:101939, **2019**. ISSN 1570-8705. doi:https://doi.org/10.1016/j.adhoc.2019.101939.

[75] A. Mohammed Mikaeil, W. Hu, and L. Li. Joint allocation of radio and fronthaul resources in multi-wavelength-enabled c-ran based on reinforcement learning. *Journal of Lightwave Technology*, 37(23):5780–5789, **2019**. ISSN 1558-2213. doi:10.1109/JLT.2019.2939169.

[76] J. Wang, L. Zhao, J. Liu, and N. Kato. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, **2019**. ISSN 2376-4562. doi:10.1109/TETC.2019.2902661.

[77] J. Li, H. Gao, T. Lv, and Y. Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. **2018**. ISSN 1558-2612. doi:10.1109/WCNC.2018.8377343.

[78] Y. Liu, H. Yu, S. Xie, and Y. Zhang. Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, 68(11):11158–11168, **2019**. ISSN 1939-9359. doi:10.1109/TVT.2019.2935450.

[79] Y. Wei, F. R. Yu, M. Song, and Z. Han. User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 17(1):680–692, **2018**. ISSN 1558-2248. doi:10.1109/TWC.2017.2769644.

[80] Y. Sun, M. Peng, and S. Mao. Deep reinforcement learning-based mode selection and resource management for green fog radio access networks. *IEEE Internet of Things Journal*, 6(2):1960–1971, **2019**. doi:10.1109/JIOT.2018.2871020.

[81] Helin Yang, Zehui Xiong, Jun Zhao, Dusit Niyato, Chau Yuen, and Ruilong Deng. Deep reinforcement learning based massive access management for ultra-reliable low-latency communications, **2020**.

[82] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti. Reinforcement learning for slicing in a 5g flexible ran. *Journal of Lightwave Technology*, 37(20):5161–5169, **2019**. ISSN 1558-2213. doi:10.1109/JLT.2019.2924345.

[83] R. Li, Z. Zhao, Q. Sun, C. I, C. Yang, X. Chen, M. Zhao, and H. Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, **2018**. doi:10.1109/ACCESS.2018.2881964.

[84] Qiang Liu, Tao Han, Ning Zhang, and Ye Wang. Deepslicing: Deep reinforcement learning assisted resource allocation for network slicing, **2020**.

[85] 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*, 24(9):2005–2009, **2020**. doi:10.1109/LCOMM.2020.3001227.

[86] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. Walid. Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. In *2019 15th International Conference on Network and Service Management (CNSM)*, pages 1–5. **2019**. doi:10.23919/CNSM46954.2019.9012702.

[87] Haozhe Wang, Yulei Wu, Geyong Min, Jie Xu, and Pengcheng Tang. Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach. *Information Sciences*, 498:106 – 116, **2019**. ISSN 0020-0255. doi:https://doi.org/10.1016/j.ins.2019.05.012.

[88] C. Jiang and X. Zhu. Reinforcement learning based capacity management in multi-layer satellite networks. *IEEE Transactions on Wireless Communications*, 19(7):4685–4699, **2020**. doi:10.1109/TWC.2020.2986114.

[89] C. Chang and N. Nikaein. Ran runtime slicing system for flexible and dynamic service execution environment. *IEEE Access*, 6:34018–34042, **2018**. ISSN 2169-3536. doi:10.1109/ACCESS.2018.2847610.

[90] H. D. R. Albonda and J. Pérez-Romero. An efficient ran slicing strategy for a heterogeneous network with embb and v2x services. *IEEE Access*, 7:44771–44782, **2019**. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2908306.

[91] D. Marabissi and R. Fantacci. Highly flexible ran slicing approach to manage isolation, priority, efficiency. *IEEE Access*, 7:97130–97142, **2019**. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2929732.

[92] 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*, 8:34538–34551, **2020**. doi:10.1109/ACCESS.2020.2974812.

[93] Y. Sun, S. Qin, G. Feng, L. Zhang, and M. Imran. Service provisioning framework for ran slicing: User admissibility, slice association and bandwidth allocation. *IEEE Transactions on Mobile Computing*, pages 1–1, **2020**. doi: 10.1109/TMC.2020.3000657.

[94] Tengteng Ma, Yong Zhang, Fanggang Wang, Dong Wang, and Da Guo. Slicing resource allocation for embb and urllc in 5g ran. *Wireless Communications and Mobile Computing*, 2020:1–11, **2020**. doi:10.1155/2020/6290375.

[95] X. Chen, Y. Tang, M. Zhang, and L. Huang. Ran slice selection mechanism based on satisfaction degree. In *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6. **2020**. doi: 10.1109/WCNCW48565.2020.9124780.

[96] H. Xiang, S. Yan, and M. Peng. A realization of fog-ran slicing via deep reinforcement learning. *IEEE Transactions on Wireless Communications*, 19(4):2515–2527, **2020**. doi:10.1109/TWC.2020.2965927.

[97] Yuansheng Wu, Guanqun Zhao, Dadong Ni, and Junyi Du. Dynamic handoff policy for ran slicing by exploiting deep reinforcement learning, **2020**. doi: 10.21203/rs.3.rs-137200/v1.

[98] Y. Liu, G. Feng, Y. Sun, S. Qin, and Y. C. Liang. Device association for ran slicing based on hybrid federated deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, pages 1–1, **2020**. doi:10.1109/TVT.2020.3033035.

[99] Hankz Hankui Zhuo, Wenfeng Feng, Qian Xu, Qiang Yang, and Yufeng Lin. Federated reinforcement learning. *CoRR*, abs/1901.08277, **2019**.

[100] Y. Shi, Y. E. Sagduyu, and T. Erpek. Reinforcement learning for dynamic resource optimization in 5g radio access network slicing. In *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. **2020**. doi:10.1109/CAMAD50429.2020.9209299.

[101] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao. Application of machine learning in wireless networks: Key techniques and open issues. *IEEE Communications Surveys Tutorials*, 21(4):3072–3108, **2019**. doi:10.1109/COMST.2019.2924243.

[102] A. Nassar and Y. Yilmaz. Reinforcement learning for adaptive resource allocation in fog ran for iot with heterogeneous latency requirements. *IEEE Access*, 7:128014–128025, **2019**. doi:10.1109/ACCESS.2019.2939735.

[103] X. Wang and T. Zhang. Reinforcement learning based resource allocation for network slicing in 5g c-ran. In *2019 Computing, Communications and IoT Applications (ComComAp)*, pages 106–111. **2019**. doi:10.1109/ComComAp46287.2019.9018774.

[104] R. Xi, X. Chen, Y. Chen, and Z. Li. Real-time resource slicing for 5g ran via deep reinforcement learning. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 625–632. **2019**. doi:10.1109/ICPADS47876.2019.00094.

[105] B. Han, A. DeDomenico, G. Dandachi, A. Drosou, D. Tzovaras, R. Querio, F. Moggio, O. Bulakci, and H. D. Schotten. Admission and congestion control for 5g network slicing. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6. **2018**. doi: 10.1109/CSCN.2018.8581773.

[106] X. Yang, Y. Liu, I. C. Wong, Y. Wang, and L. Cuthbert. Genetic algorithm for inter-slice resource management in 5g network with isolation. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6. **2020**. doi:10.23919/SoftCOM50211.2020.9238298.

[107] B. Han, J. Lianghai, and H. D. Schotten. Slice as an evolutionary service: Genetic optimization for inter-slice resource management in 5g networks. *IEEE Access*, 6:33137–33147, **2018**. doi:10.1109/ACCESS.2018.2846543.

[108] X. C. Xiao, X. W. Zheng, Y. Wei, and X. C. Cui. A virtual network resource allocation model based on dynamic resource pricing. *IEEE Access*, 8:160414–160426, **2020**. doi:10.1109/ACCESS.2020.3020944.

[109] Baker M. Sesia S., Toufik I. *The UMTS Long Term Evolution: From Theory to Practice, 2nd Edition*. **2011**.

[110] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, **2017**.

[111] Yuxi Li. Deep reinforcement learning: An overview, **2017**.

[112] Coady Pat. trpo. `https://github.com/pat-coady/trpo`, **2018**.

[113] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, **2014**.

[114] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, **2019**.

[115] Namig J. Guliyev and Vugar E. Ismailov. Approximation capability of two hidden layer feedforward neural networks with fixed weights. *Neurocomputing*, 316:262 – 269, **2018**. ISSN 0925-2312. doi:https://doi.org/10.1016/j.neucom.2018.07.075.

[116] Namig J. Guliyev and Vugar E. Ismailov. On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Networks*, 98:296 – 304, **2018**. ISSN 0893-6080. doi:https://doi.org/10.1016/j.neunet.2017.12.007.

[117] MorvanZhou. Evolutionary-algorithm. `https://github.com/MorvanZhou/Evolutionary-Algorithm/`, **2020**.

[118] Madalina Drugan. Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms. *Swarm and Evolutionary Computation*, 44, **2018**. doi:10.1016/j.swevo.2018.03.011.

[119] Mehmet Koseoglu. Lower bounds on the lte-a average random access delay under massive m2m arrivals. *IEEE Transactions on Communications*, 64(5):2104–2115, **2016**.

[120] M. Koseoglu. Smart pricing for service differentiation and load control of the lte-a iot system. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 187–192. **2015**. ISSN null. doi:10.1109/WF-IoT.2015.7389050.

[121] Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, **2018**.

[122] M. Koseoglu. Pricing-based load control of m2m traffic for the lte-a random access channel. *IEEE Transactions on Communications*, 65(3):1353–1365, **2017**. doi:10.1109/TCOMM.2016.2644667.

[123] CodeReclaimers. *NEAT Configuration File Description*, **2015-2017 (accessed January 3, 2020)**. `https://neat-python.readthedocs.io/en/latest/config_file.html`.