Wireless In-Network Processing for Multimedia Applications

Haitham Afifi

Universitätsdissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr. Ing.)

in Internet-Technology
and Softwarization

eingereicht an der
Digital-Engineering-Fakultät
der Universität Potsdam

Betreuer

Prof. Dr. Holger Karl
Hasso Plattner Institute, University of Potsdam

Gutachter

Prof. Dr. Olaf Landsiedel
Kiel University

Prof. Dr. Christoph Sommer
Technische Universität Dresden

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-60437
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-604371

Abstract

With the recent growth of sensors, cloud computing handles the data processing of many applications. Processing some of this data on the cloud raises, however, many concerns regarding, e.g., privacy, latency, or single points of failure. Alternatively, thanks to the development of embedded systems, smart wireless devices can share their computation capacity, creating a local wireless cloud for in-network processing. In this context, the processing of an application is divided into smaller jobs so that a device can run one or more jobs.

The contribution of this thesis to this scenario is divided into three parts. In part one, I focus on wireless aspects, such as power control and interference management, for deciding which jobs to run on which node and how to route data between nodes. Hence, I formulate optimization problems and develop heuristic and meta-heuristic algorithms to allocate wireless and computation resources. Additionally, to deal with multiple applications competing for these resources, I develop a reinforcement learning (RL) admission controller to decide which application should be admitted. Next, I look into acoustic applications to improve wireless throughput by using microphone clock synchronization to synchronize wireless transmissions.

In the second part, I jointly work with colleagues from the acoustic processing field to optimize both network and application (i.e., acoustic) qualities. My contribution focuses on the network part, where I study the relation between acoustic and network qualities when selecting a subset of microphones for collecting audio data or selecting a subset of optional jobs for processing these data; too many microphones or too many jobs can lessen quality by unnecessary delays. Hence, I develop RL solutions to select the subset of microphones under network constraints when the speaker is moving while still providing good acoustic quality. Furthermore, I show that autonomous vehicles carrying microphones improve the acoustic qualities of different applications. Accordingly, I develop RL solutions (single and multi-agent ones) for controlling these vehicles.

In the third part, I close the gap between theory and practice. I describe the features of my open-source framework used as a proof of concept for wireless in-network processing. Next, I demonstrate how to run some algorithms developed by colleagues from acoustic processing using my framework. I also use the framework for studying in-network delays (wireless and processing) using different distributions of jobs and network topologies.

Zusammenfassung

Mit der steigenden Anzahl von Sensoren übernimmt Cloud Computing die Datenverarbeitung vieler Anwendungen. Dies wirft jedoch viele Bedenken auf, z. B. in Bezug auf Datenschutz, Latenzen oder Fehlerquellen. Alternativ und dank der Entwicklung eingebetteter Systeme können drahtlose intelligente Geräte für die lokale Verarbeitung verwendet werden, indem sie ihre Rechenkapazität gemeinsam nutzen und so eine lokale drahtlose Cloud für die netzinterne Verarbeitung schaffen. In diesem Zusammenhang wird eine Anwendung in kleinere Aufgaben unterteilt, so dass ein Gerät eine oder mehrere Aufgaben ausführen kann.

Der Beitrag dieser Arbeit zu diesem Szenario gliedert sich in drei Teile. Im ersten Teil konzentriere ich mich auf drahtlose Aspekte wie Leistungssteuerung und Interferenzmanagement um zu entscheiden, welche Aufgaben auf welchem Knoten ausgeführt werden sollen und wie die Daten zwischen den Knoten weitergeleitet werden sollen. Daher formuliere ich Optimierungsprobleme und entwickle heuristische und metaheuristische Algorithmen zur Zuweisung von Ressourcen eines drahtlosen Netzwerks. Um mit mehreren Anwendungen, die um diese Ressourcen konkurrieren, umgehen zu können, entwickle ich außerdem einen Reinforcement Learning (RL) Admission Controller, um zu entscheiden, welche Anwendung zugelassen werden soll. Als Nächstes untersuche ich akustische Anwendungen zur Verbesserung des drahtlosen Durchsatzes, indem ich Mikrofon-Taktsynchronisation zur Synchronisierung drahtloser Übertragungen verwende.

Im zweiten Teil arbeite ich mit Kollegen aus dem Bereich der Akustikverarbeitung zusammen, um sowohl die Netzwerk- als auch die Anwendungsqualitäten (d. h. die akustischen) zu optimieren. Mein Beitrag konzentriert sich auf den Netzwerkteil, wo ich die Beziehung zwischen akustischen und Netzwerkqualitäten bei der Auswahl einer Teilmenge von Mikrofonen für die Erfassung von Audiodaten oder der Auswahl einer Teilmenge von optionalen Aufgaben für die Verarbeitung dieser Daten untersuche; zu viele Mikrofone oder zu viele Aufgaben können die Qualität durch unnötige Verzögerungen verringern. Daher habe ich RL-Lösungen entwickelt, um die Teilmenge der Mikrofone unter Netzwerkbeschränkungen auszuwählen, wenn sich der Sprecher bewegt, und dennoch eine gute akustische Qualität zu gewährleisten. Außerdem zeige ich, dass autonome Fahrzeuge, die Mikrofone mit sich führen, die akustische Qualität verschiedener Anwendungen verbessern. Dement-

sprechend entwickle ich RL-Lösungen (Einzel- und Multi-Agenten-Lösungen) für die Steuerung dieser Fahrzeuge.

Im dritten Teil schließe ich die Lücke zwischen Theorie und Praxis. Ich beschreibe die Eigenschaften meines Open-Source-Frameworks, das als Prototyp für die drahtlose netzinterne Verarbeitung verwendet wird. Anschließend zeige ich, wie einige Algorithmen, die von Kollegen aus der Akustikverarbeitung entwickelt wurden, mit meinem Framework ausgeführt werden können. Außerdem verwende ich das Framework für die Untersuchung von netzinternen Verzögerungen unter Verwendung verschiedener Aufgabenverteilungen und Netzwerktopologien.

Acknowledgments

I would like to express my sincere gratitude to all those who have supported and encouraged me throughout my journey towards completing this PhD thesis.

First and foremost, I am deeply indebted to my supervisor Prof. Dr. Holger Karl, whose expertise, guidance, and constant encouragement have been invaluable throughout my research. I am grateful for his patience, encouragement, and the countless hours spent reviewing my work and providing constructive feedback. I would not have been able to complete this thesis without his support. Not to forget, he always made sure to have a comfortable working environment.

I would also like to thank the members of my thesis committee, Prof. Dr. Olaf Landsiedel and Prof. Dr. Christoph Sommer, for their valuable time to review my Thesis. Furthermore, I would like to express my gratitude to Prof. Dr. Andreas Polzen, who agreed to chair my doctoral examination board, as well as Prof. Dr. Christian Dörr who agreed to be my second supervisor and Prof. Dr. Patrick Baudisch for taking part in my examination board

I am grateful to my colleagues at “Computer Networks” and at “Internet-Softwarization and Technology" groups for their friendship and support. My thanks also go to Ms. Kerstin Miers for being always ready to help. Additionally, I would like to thank my colleagues at “Deutsche Forschungsgemeinschaft (DFG) - Acoustic Sensor Networks”, whose insights, feedback and encouragement have been invaluable in helping me navigate the challenges of the PhD journey.

I am also grateful to my family and friends for their unwavering support and encouragement. Their belief in me and my abilities has been a constant source of inspiration throughout my PhD journey. Finally, I would like to express my gratitude to the DFG, whose financial support made this research possible.

Thank you all for your invaluable contributions towards the completion of this thesis.

Contents

Part I

The Problem

Wireless Sensor Networks (WSNs) were originally developed to collect data from the surroundings and process them. Managing these networks (e.g., routing and resource allocation) to optimize energy efficiency, data accuracy and latency has been well studied over the past decades. The results were then adopted by similar networks such as Internet of Things (IoT).

However, there is an overall lack of research how to best manage these networks for newly emerging applications, especially those with substantial amounts of data to collect under tight delay constraints – for example, wireless multimedia applications. Moreover, these applications may be used in a highly dynamic environment, making network management even more complex.

This thesis aims to identify and evaluate WSN management approaches – which are also applicable for IoT networks – focusing on wireless multimedia applications such as Wireless Acoustic Sensor Network (WASN), while exploiting recent trends in hardware and network development.

This part will provide an introduction to the thesis by first discussing the motivation and context, followed by the background, where I highlight the specific needs and challenges of WASN, explain the wireless properties used in this thesis and provide a background on Reinforcement Learning (RL) as it will be a main contribution in later chapters. Finally, I will introduce the research aims and questions

1 Introduction

1.1 Motivation

WSNs have conventionally focused on simple data collection applications owing to their hardware constraints. With the advent of more powerful yet still cheap hardware (e.g., Arduino or Raspberry Pis), a new class of applications for WSNs is emerging where the collected data is more voluminous and the application constraints like maximum acceptable delays are tighter. Examples for such applications often come from the acoustic or video-signal processing domain: distributed microphone arrays, collecting streams of audio data, or acoustic-based localisation of speakers. In such applications, nodes are often plugged into easily available power outlets, but wired data communication is often not available or too costly or cumbersome to install. Hence, conventional figures of merit for WSNs like energy efficiency take second place; application-oriented ones like delays, dependability, or feasibility with constrained wireless are more important. Wireless communication is still an absolute necessity.

A simplistic approach to support such applications in acoustic sensor networks would be to record data on distributed nodes and send all that data to a central server where the signals included in the data is processed. Yet an alternative approach exists: to move some (not necessarily all) of the processing jobs from a central server to the wireless nodes themselves. This thesis will investigate whether and which advantages exist in this approach. In the following, I identify different viewpoints on the benefits.

1. Real-time applications (e.g., WASN) can require powerful and expensive processors for fast computation. Distributing some of these processing jobs to wireless smart nodes like Raspberry Pis or Arduinos will considerably **lower the cost** requirements of a central server and leverage good price/performance ratios of this class of devices.

   To support this hypothesis, Google would have to double the number of its data centres if each user used google-voice services for 3 minutes a day, due to *high computation loads* [Jou+17]. In fact, this was the motivation of developing Google's embedded device, Tensor Processing Unit (TPU). But

here, it is not necessary to buy new devices, this viewpoint holds when exploiting the idle smart devices as seen in smart homes.

2. There are two options to increase **dependability**
   - Option 1: Lest there is a glitch or failure at the central processor that may breakdown the system, we distribute the jobs on wireless nodes. Thus, if a node has failed, we can easily re-assign its job to another node. Even if we intend to have standby central servers, we are back to the first point, where we save costs for the standby servers.
   - Option 2: There will typically be multiple wireless nodes, as seen in smart homes or smart factories, that can act as mutual standby devices. Although provisioning standbys for central servers is still possible, it would, as mentioned earlier, increase cost and complexity, compared to the wireless nodes.

3. It might actually be more **energy-efficient** when some of the jobs compress data (which is often, but not always the case in such distributed multimedia applications) as the well-known energy trade-off between processing and sending data still holds [Kad16].

4. Distributing processing can **reduce latency** compared to central solutions as only smaller amounts of data have to be communicated. This holds under the assumption that the wireless nodes are supported with advanced processing units, being capable of processing the jobs in reasonable time.

5. It can save wireless data rate by **reducing the amount of data to communicate**. Even with high-performance Wireless Local Area Networks (WLANs), this can still be a concern in dense deployments, when all wireless nodes are sending raw data at high rates.

6. It can provide **higher privacy** by relying on local computation instead of sharing raw data with third parties on a cloud.

Another type of WSN environment is when the nodes can move, i.e., they are not power-plugged anymore. This adds an additional degree of freedom to improve the data collection, especially for online streaming. Although, this raises again the concern of energy consumption, I assume that stored energy, needed as a power supply for the motor, eclipses the energy needed for sensing and sending the data.

1.2 Research Questions

The network architecture of WSNs were originally designed for low data rate requirements, and hence, they cannot be directly applied for high data rates, because they will over-utilize the wireless network. The most basic architecture forwards the raw data to the central server. To reduce power dissipation during signal transmission, data reduction techniques (such as aggregation [KW05; LFZ13] and compressive sensing [MCN17]) has been introduced. Although these techniques use in-network processing, they are limited to specific types of applications and were designed to do specific jobs (e.g., finding correlation or summing and averaging), so that they are not applicable for arbitrary jobs with predefined connections. Data centres and wide-area networks have a similar problem, with respect to voluminous data traffic, supported with numerous studies proposing different solutions, such as virtual network functions [Sun+22] and service function chains [GB16; WCY20]. They use the network devices for data processing and rely on virtualized jobs (let them be functions or services) that could be easily migrated from one device to another. Hence, these solutions were also adopted in wireless networks. Yet, there is a literature gap, due to the differences between how wired (as in data centres) and wireless networks behave.

It is important to fill this gap, otherwise, practical implementations yield a different (and very likely worse) results compared to the expectations from theory. For example, in wired networks every port on a router is typically in a separate collision domain, while in wireless networks, wireless devices operate in the same collision domain, adding the interference challenge and the multi-cast property. Additionally, wireless networks can be mobile, which is a feature that wired networks do not have.

Given the lack of research regarding using WSN for in-network processing in wireless multimedia applications, this thesis aims to identify and evaluate wireless network management approaches, while considering the wireless properties and exploiting the wireless network features in general[1].

Accordingly, the research objectives are:

1. to adopt resource allocation approaches for in-network processing from wired for wireless networks, while considering physical wireless properties (e.g., Signal-to-Interference-Noise Ratio (SINR)) and data link properties (e.g., Medium Access Control (MAC) protocols).

1 codes can be found in git@gitlab.hpi.de:itsw-workgroup/itsw-theses/hatiham-afifi/codes.git

2. to exploit wireless features in in-network processing, such as multi-cast transmission, power control and mobility.
3. to evaluate the effectiveness of these approaches in wireless multimedia applications.
4. to provide an in-situ test-bed as a proof-of-concept for wireless network management in general and for showing demos from wireless multimedia applications.

In this context, I focus on the following research questions:

- what is the gain of considering wireless properties in in-network processing approaches? I focus here on delay and symbol error rate as metrics.
- how effective is it to exploit the wireless features? I choose multi-cast, power control and mobility as wireless features.
- how to exploit multimedia features to assist in network management? More specifically, I choose the synchronization feature in WASN to assist the wireless MAC.

1.3 Thesis Overview

In Chapter 2, I introduce the context of the study and the background of some of the applied solutions. In Part II, I focus on wireless metrics when evaluating the network approaches. In Chapter 3, I consider wireless network management by distributing the application processing inside the network and show the gain of considering the wireless properties and features, compared to wired networks approaches. I worked with Sébastien Auroux on the optimization problem in [AAK18], and built up on it to add power control [AK19b] and develop a heuristic solution [AK19a].

In Chapter 4, I worked with Konrad Horbach [AHK19] to solve the same problem, yet, the focus is on meta-heuristic approaches to quickly find feasible solutions and show the gap between these solutions and the ones in Chapter 3. In Chapter 5, I worked with Fabian Sauer [ASK21] with the assumption that multiple applications are competing for wireless network resources and we use RL for choosing which applications to admit. In Chapter 6, I use synchronization from WASN to synchronize the transmissions in WSN; I highlight in which scenarios such approaches can be applied and what their gain would be. The acoustic processing was developed by Tobias Gburrek and Joerg Schmalenstroeer; the work was published in [Afi+22].

In Part III, I focus on features from wireless multimedia (more specifically, WASN) applications for network management. In Chapter 7, I show the impact of selecting a subset of jobs from the application on the performance of wireless networks. The WASN application was developed by Michael Gunther and Andreas Brendel, while I evaluated the network aspects, so that the work was published in [Gun+21] and received the “Best Poster" award.

In Chapter 8, I study the impact of selecting a subset of sensors on both wireless network and application performance. Again, the application was developed by Michael Gunther and Andreas Brendel, while I developed the selection approach. Here, the evaluation took place jointly and the results were published in [Afi+21]

In Chapter 9, I exploit the mobility feature of wireless networks and show how it improves application performance. This work was jointly done with Arunselvan Ramaswamy where we jointly work on the RL formulation and published the work in [ARK21b] and submitted it to [ARK22].

In Part IV, I present a test-bed for wireless network management, where in Chapter 10, I describe the features and implementation of the test-bed. The test-bed is an open-source [Afi] and has been successfully shown in many Demos in the field of acoustic sensor networks. Examples of WASN applications were show in cooperation with:

- Aleksej Chinaev [CA22], Tobias Gburrek and Joerg Schmalenstoeer [TA22] for acoustic synchronization
- Markus Bachman and Andreas Brendel for acoustic signal extraction [Mar18]
- Alexander Nelus and Luca Becker for privacy preserving in acoustic application [Ale22]
- Janek Ebbers for acoustic scene classification on a Raspberry-Pi network [JA22]

Thanks to this cooperation, I was able to update the test-bed with additional features needed by the acoustic applications for an easy setup and better management.

In Chapter 11, I present case studies for WASN applications, where I worked with Konrad Horbach on evaluating acoustic applications, running on the tesbed, from the network perspectives.

2 Background

2.1 Wireless Acoustic Sensor Networks

In acoustic sensor networks where there can be a cooperation between wireless connected devices when exchanging audio information (either raw or processed data). Such a cooperation can reduce the communication data rate and the computational load, improve scalability and flexibility and enhance the performance.

2.1.1 Sensor Network Basics

The structure of sensor networks is mainly based on two roles: *node* and a *sink* (a.k.a. gateway) [KW05]. In principle, nodes are responsible for collecting data and transmitting them to the sink. Thanks to hardware development, nodes are also able to preprocess data and forward it.

Based on these two simple roles, different variants of the network architecture arise. For example, the network can have several sinks, while the nodes choose which sink to forward the data to, based on factors such as proximity and availability.

Additionally, the data transmission can be single-hop (nodes are only sensors) or multi-hop (nodes act as sensors and relays). The management of data transmission should be considered on at least two different layers: data link and network layers. In this thesis I consider MAC protocols for the data link layer and routing for the network one.

The mobility in sensor networks is considered in two ways. First, network mobility, where nodes and/or sinks continuously move over time (e.g., sensors installed on vehicles) or occasionally being moved (e.g., a node is moved from one position to another over long measurement time). The latter is used for re-planning while the former is live events, where both changes are controllable. Second, event mobility, where the case of the event change position over time (e.g., speaker localization). In this case, changes are not controllable.

2.1.2 Acoustic applications

Different acoustic applications may have different requirements. Here, I list some of the applications as well as their requirements, which will be used in later chapter.

For instance, collecting audio from multiple microphones requires that the audio sampling clocks (i.e., the collected audio samples) are synchronized. This imposes the challenge of estimating and correcting the Sampling Rate Offset (SRO) between the microphones [GSH21]. This require high data rates, especially when data are being relayed (i.e., multi-hop routing), hence, over utilizing the wireless network.

Some applications inherit the dynamic topology in sensor networks as seen in mobility or node failures, i.e., uncontrollable changes. Detecting the dynamicity may be easy from the networking perspective, but the acoustic application still needs to handle the changes in the network, since dynamicity also impacts acoustic performance. As an example, by losing nodes we also lose their contribution, i.e., information in terms of raw or processed data.

For specific types of applications, such as speaker localization and diarisation [BAK04b; JSH12] the applications are time-sensitive. Meanwhile, as the number of wireless devices increases, the network delay will very likely increase. Additionally, forwarding high quality audio data (i.e., large number of audio samples) results in high delays. In case the data are processed on the wireless devices before being forwarded (e.g., feature extraction or audio compression), computational delay can also be a bottleneck, because these devices, unlike cloud servers, have very limited computational resources. This results in the trade-off between raw data forwarding and local data processing (a.k.a. in-network processing).

When multiple applications or audio processing algorithms run on the top of the sensor networks, the wireless network may have conflicting or complementary jobs that over-utilize the network, decreasing the performance of all applications. Alternatively, either the applications should self-configure themselves to cope with the available network resources or the other way around, the applications could be given priorities (based on their importance or revenue) and the network decides how to allocate its resources. I focus on the latter in this thesis.

2.1.3 Wireless Technology Specifications

Selecting the most adequate wireless technology depends on the application requirement. For example, mobile broadband protocols e.g., 3G/4G/5G, offer high data rates but they are far away from being cost effective in dense sensor networks, especially if there is a need for planning additional relay cells (such as femto and pico cell) in case of bad coverage, in addition to the subscription costs [ABZ20].

On the contrary, technologies operating in unlicensed bands are low cost, hence, they are more popular for end-users with dense networks as in smart homes. I summarize in Table 2.1 their specifications. For example, LoRaWAN seems to have the longest battery life and covers more than 100 m, which makes it very suitable

Table 2.1: Summary of wireless technology specifications

Protocol	Bluetooth [02]	BLE [Gup16]	WiFi [21]	Zigbee/ 6LoWPAN [20]	LoRaWAN [Fra+20]
Frequency	2.4 GHz	2.4 GHz	2.4/5 GHz	2.4/0.868/0.915 GHz	169/868/915 MHz
Coverage (m)	30	10	25–50	30	>100
Data rate	2 Mbps	270 Kbps	0.45-2.4 Gbps	150 kbps	12.5/21 kbps
Battery life	Month	Year	Day	Month-Year	Year(s)

for outdoor applications. However, in this thesis I focus on indoor sensor networks, more specifically, WASNs requiring high data rates, which makes WiFi (specified by IEEE 802.11 standards) the most adequate technology.

2.2 Wireless Medium Access Control Protocols

The main objective of wireless MAC is to coordinate which nodes access the shared wireless medium when. An additional requirement, when focusing on WSN, is energy efficiency and its trade-off with typical network metrics such as delay and data rates [KW05]. However, this is not the case in this thesis. Unlike traditional WSN applications (such as motion monitoring and temperature alarms in forests) that are battery-operated and run for months/years, wireless multimedia applications (e.g., WASN) are normally power-plugged and even when they are battery-operated (e.g., smart phones), they have a short life-cycle expectancy (hours or a couple of days) (Table 2.1).

In Chapter 11, I explain how the delay requirements of acoustic applications evince into MAC protocols.

2.2.1 Basics of Medium Access Control

Since I focus in this thesis on wireless multimedia applications, the focus is on MAC protocols with high throughput and low delay. There are multiple factors that impact these metrics such as collisions and MAC signaling overhead. Collisions take place when transmission from two or more nodes arrives at a receiver at the

same time. Hence, the receiving nodes (which are also in the same collision domain) cannot decode the arrived packets, resulting in data losses (affecting application performance) or/and retransmissions, resulting in lower throughput and higher delays.

The MAC is tightly coupled with the physical channel status (e.g., fast vs slow fading, heated devices, etc.). For example, as the noise level at the receiver changes rapidly, it is hard to define SINR. In this thesis, I assume that the physical layer is (very) slowly changing, so I can ignore small-scale parameters' impact (e.g., coherence time and shadowing) and focus on large-scale parameters impact)(e.g., attenuation). Accordingly, using such properties, it is easy to define the size of the collision domain and estimate the SINR, and accordingly, correctly estimate assigned data rates.

Signaling overhead is used to regulate the access to a shared medium of multiple nodes, by providing extra information or instructions. For example, solutions such as RTS/CTS and busy tone are used to tackle the exposed- and hidden-terminal problems. Other signaling protocols are used to synchronize wireless transmissions of different nodes. In my work, I ignore the impact of such protocols because I assume the impact of these protocols is a constant to subtract from performance metrics such as delay and throughput. Additionally, it makes the analysis and problem formulations simpler. Note that, however, different MAC protocols may have different signaling overhead. I show in Chapter 6 how MAC protocols can benefit from the properties of acoustic application.

2.2.2 Contention-Based and Contention-Free Protocols

There is a huge number of MAC protocols that have been developed over the past decades. I focus on two popular classes: contention-based and contention-free protocols.

On the one hand, contention-free protocols assign the available wireless resources to the nodes so that the assignment is long-term. Assignments last minutes or even longer. A key source of overhead is to distribute these assignments as a schedule to all nodes. To handle changes in the network (e.g., mobility and failing nodes), signaling is used to reassign the resources, in order to avoid collisions. Fixed-schedule Time Division Multiple Access (TDMA) (or just briefly, TDMA) is an example of contention-free protocols where the time is divided into time slots and each node is assigned one or more time slots for transmission. The period between the time of sending a packet and the time of sending the next packet (i.e., a node sends a packets and waits for other nodes to send then restart the cycle) is defined as a time frame. Another signaling overhead is synchronization

because TDMA requires tightly synchronized clocks between the nodes. There exists also other multiplexing dimensions such as frequency, carrier and spatial division multiple access. Contention-free protocols are conventionally centralized, yet there exists proposals for distributed implementations as well.

Contention-based protocols are typically distributed and usually have a random access behavior. Packets to be sent may be generated at random (e.g., motion detection) or on-demand (e.g., acoustic application), the timers for sending the packets are random as well as the reattempts. A popular protocol Carrier Sense Multiple Accesses with Collision Avoidance (CSMA/CA) by IEEE 802.11 [21] is a listen-before-send protocol: a node senses the channel before sending, if the channel is idle, it sends the packet. Otherwise, the node awaits the channel being idle then waits a random (a.k.a. back-off) time that is chosen from the current contention window. Although the back-off time reduces the probability of two or more nodes sending at the same time after the channel being idle, it decreases the channel throughput since the channel is idle while nodes have packets to send. Further details on the operation of TDMA and CSMA/CA will be discussed in Chapter 6.

2.3 Reinforcement Learning

RL is an area of machine learning that aims to optimise an *agent*'s behaviour in an *environment* over time. It works by taking *actions* to maximize the *reward* in a particular situation or environment [SB18, Ch. 3]. Unlike supervised learning, which needs the knowledge about labels or the answer key in advance, RL trains an agent to learn from trial and error without any prior knowledge about the ground truth.

The agent's behaviour, i.e., its translation from *states* to corresponding actions, is called a *policy* and it can be controlled via feedback to the agent. This feedback is commonly called *reward*, which is then perceived by the agent. Hence, it makes it easy for the agent to adapt a policy to different objectives by just updating the reward. For each *action* the agent takes, the environment reports back not only the reward but also the new state.

The concept of an environment is very similar to the concept of an agent. It also reacts to inputs with certain outputs. In contrast to the agent's workflow, the environment will use the agent's outputs (actions) as input and will feed its outputs back to the agent as new states. The difference is that the environment's translation from inputs to outputs cannot be influenced externally. Therefore, there is also no reward mechanism involved in the environment.

The rewards can be both good or bad to indicate positive or negative behaviours, respectively. Using a series of consecutive actions, the agent aims at maximizing the cumulative discounted reward over time. Note that this is my main objective in this thesis when using RL, but it is not only the use case.

The learning process relies on a cycle of interactions between the agent and the environment. At time t, the agent observes state $s_t \in S$ and performs action $a_t \in A$ to receive a reward r_t and the new state s_{t+1}. The RL *return* can be either episodic or continuous. On the one hand, episodic return has finite number of steps to calculate the reward. In other words, there is at least one state in S as a terminal. On the other hand, continuous return encounters infinite number of steps, aiming at maximizing the reward on long term and not just current state. To achieve this, they both use discounted return R_t given by

$$R_t = \frac{1}{T} \sum_{k=0}^{T} \gamma^k r_{t+k+1} \quad , \tag{2.1}$$

where $0 < \gamma < 1$ is the discount factor.

There are different classifications for reinforcement learning algorithms, such as model-based vs. model-free [SB18, Ch. 8] and on-policy vs off-policy [SB18, Ch.7].

2.3.1 Model-Based and Model-Free Reinforcement Learning

Model-based RL assumes that an action takes time to be executed, thus the agent uses this time to simulate a *model* to learn more about the environment given the previously taken actions and corresponding rewards. It aims to estimate the transition probabilities $p(s_{t+1}|s_t, a_t$ and the corresponding expected reward $E[r_t|s_t, s_{t+1}, a_t]$ when going from s_t to s_{t+1} via action a_t.

This becomes impractical when the state and action space is large. Nonetheless, it is still useful when getting data is expensive and computation does not matter. Unlike model-based algorithms, *model-free* RL learns directly by trial and error and does not require the relatively large memory of a model-based approach. In either of these models, learning can take place using value-based or policy-based RL.

2.3.2 Value-Based and Policy-Based Reinforcement Learning

Conventionally, RL algorithms consider Markov Decision Problem (MDP) with a single criterion. For each state s_t at time instant t, an action a_t is selected based on a *policy* $\pi(s_t)$. Given the policy value $V^\pi(s_t) = \sum_{t=0}^{\infty} \gamma^t r_t$, the main objective is, as stated above, is to maximize the discounted reward r_t . This is approached

by searching for the optimal policy$\pi^*(s_t)$, defined as the one that maximizes the policy's value in each state

Value-based RL uses a value function to represent how "good" it is to be in a certain situation. I focus here on a specific type of value function known as Q-Learning, which maps a combination of state and action to a Q-value via a $Q(s_t, a_t)$ function. Hence, it extends the representation of the value to be for an action at a particular state.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \arg\max Q(s_{t+1}, a_t) - Q(s_t, a_t)) \quad , \tag{2.2}$$

where $\alpha \in [0, 1]$ is defined as the learning rate; the extent to which Q-values are updated per each step. Accordingly, the policy π is a straight-forward greedy policy:

$$\pi(s) = \arg\max Q(s_{t+1}, a_t) \tag{2.3}$$

The fact that Q-Learning relies on a arg max function makes it only applicable to discrete action space. For a continuous action space, other value-based functions (e.g., actor-critic function) can be used.

Policy-based RL avoids the problem of finding the action with maximum value arg max, which makes it popular for continuous action and state spaces. Unlike Q-Learning, states are mapped to actions directly by means of a function approximation, without utilising Q-values; it is equivalent to the actor-critic architecture without a critic. The parameters of the function approximation are updated following gradient representing the policy's performance [PS08].

In this thesis I focus on value-based functions since I use RL with problems that have discrete action spaces, which is faster in learning (i.e., fewer training steps).

2.3.3 Exploration and Exploitation

Initially, the agent has no knowledge about the environment. The aim when exploring the environment is to be broad enough not to miss good solutions and economical in the sense of not requiring too many trials. This results in the *exploration/exploitation* dilemma: without exploration, the agent will only choose the best solution found so far, not learning about potentially better solutions. Too much exploration leads to a random behaviour without exploiting the available knowledge. The trade-off between exploration and exploitation is dependent on the problem definition and must be carefully balanced.

In this thesis Section 4.2, I focus on two variants for exploring the environment;

greedy epsilon and epsilon decay [Mor20, Ch. 4]. Both variants rely on an ϵ parameter; a random action is selected (i.e., exploration) with probability ϵ. Greedy epsilon fixes the epsilon parameter over all steps to explore/exploit the environment. Epsilon decay, on the other hand, initially sets ϵ to a high value, meaning that the agent chooses actions at random to explore the new environment and update the Q-function. Then, the epsilon's value decays over time steps to shrink the exploration rate as the Q-values are refined through several steps, hence, the agent starts exploiting the learned experience.

2.3.4 Deep Q-Learning

In general, Q-Learning is a tabular method that is applied when states and actions are in discrete spaces. When the number of states is very large (i.e., large storage is required) or the states are in continuous space, the table is then replaced by a neural network (a.k.a Deep Q-Learning) that acts a value function approximation. This is an example of deepRL, which still guarantees convergence to the optimal solution in case of a discrete state space [Tes94] (under the assumption that the environment is static). Meanwhile, for continuous state spaces, it has been shown that even for simple toy problems, the value function approximations can become unstable and diverge [BM94]. Hence some techniques were proposed to avoid divergence [Bai95; Rie05]. In this thesis, I choose the Bellman residual approach [SB18, Ch. 3]. First, let us parameterize the function approximation with θ so that the optimal value function Q^* is given by

$$\arg\max_a Q^*(s_t, a_t) \equiv \arg\max_a Q(s_t, a_t; \theta^*) \quad , \tag{2.4}$$

where θ^* represents the optimal neuron weights. Optimal weights are those that minimizes the Bellman loss function $\mathcal{B}$

$$\mathcal{B}_t = (r_t + \gamma E[Q(s_{t+1}, a_{t+1}; \theta_t) p(s_{t+1}|s_t, a_t) - Q(s_t, a_t)])^2 \tag{2.5}$$

Note that the Markov state transition $p(s_{t+1}|s_t, a_t)$ is not known in practice (and in model-free RL), therefore, the expected value $E[Q(s_{t+1}, a_{t+1}; \theta_t) p(s_{t+1}|s_t, a_t)]$ is substituted with $\max_a Q(s_{t+1}, a; \theta_t)$ to use the sample-gradient instead. This is known as temporal difference $\mathcal{L}$.

$$\mathcal{L}(t)^2 = (r_t + \gamma \max_{a'}(Q^*(s_{t+1}, a')) - Q^*(s_t, a_t))^2 \tag{2.6}$$

After T steps, the parameter θ is then given by

$$\theta_T = \theta_0 + \sum_{t=0}^{T-1} \alpha_t \nabla_\theta \mathcal{B}_t \tag{2.7}$$

where θ_0 represents the initial assigned weights to the neural network. In Chapter 9, I extend this analysis to consider the impact of the environment on the convergence.

2.3.5 Multi-Objective Reinforcement Learning

The main difference between a single-objective and a multi-objective RL is the reward function. In the former, the reward is a numeric feedback $r \longrightarrow \mathbb{R}$, while the latter has a vector of rewards $\mathbf{r} \longrightarrow \mathbb{R}^d$, where d is the number of objectives. Hence, single-objective RL has at least one optimum solution, while multi-objective RL has a set of solutions, such as convergence set and Pareto front [Roi+14], where it is not necessarily possible to compare two solutions with respect to a single optimality criterion.

There are different ways to handle such a case. For example, there could be a policy (i.e., an agent) for each objective and then based on the user's preference or the most dominating agent (whose policy value at a given state is better than the corresponding policy values of other agents) is selected [WZ15]. In this work, I use the scalarisation approach [Rad+19]; I reduce the multi-objective reward's domain to a single domain: $u(\mathbb{R}^d) \longrightarrow \mathbb{R}$ via a utility function u. In this thesis, I use weighted sum rewards since it follows the assumption of additive return in the Bellman equation Equation (2.5), which is necessary when using RL models like Q-Learning [RSN20; RWO15].

Accordingly, the utility function computes the product of a weight vector $\mathbf{q}$ and the reward vector $\mathbf{r}$

$$u(\mathbf{V}^\pi) = \mathbf{q}^{\mathrm{T}}\mathbf{V}^\pi \tag{2.8}$$

where the weights $\mathbf{q}$ are defined based on the user's preference.

2.3.6 Constrained Reinforcement Learning

In principle, constrained-RL is a special case of multi-objective RL. Even the methods used for solving constrained-RL are very similar to multi-objective RL. For example, the later uses scalarisation for optimizing multiple rewards, while the former also uses scalarisation methods such as Lagrangian relaxation [Ber99], to handle multi-dimension objectives. In contrast to multi-objective RL, constrained-RL have constraints with corresponding thresholds that must not be exceeded.

There are, however, some constrained-RL problems that do not have hard constraint thresholds but rather soft-constraints; violating a constraint is undesirable but not catastrophic [Hua+22]. Another type of RL that also achieves constraint satisfaction is safe-RL, not only in the deployment but also during learning [GFF15a].

In constrained-RL, a second reward function $p(t)$ (as defined by [Gei06]) is used to denote a constrained value function $P^{\pi}(s_t)$. Hence, the objective is

$$\max_{\pi} V^{\pi} \tag{2.9}$$

$$\text{s.t.} \quad P^{\pi} \leq P_0 \tag{2.10}$$

where P_0 is an upper-limit bound. There exist multiple approaches for constrained RL problems [Ach+17; Gei06; GFF15b]. I focus in this thesis on approaches that can be used in online learning scenarios.

In general, the average value of the second reward function is

$$\mathbb{E}[P^{\pi}(s_t)] = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t p(t) \;\leq\; P_0 \tag{2.11}$$

where $p(t), P_0 \geq 0$.

Considering constrained-RL with risk-neutral criterion [GFF15b], I can define the constrained valued function as:

$$P^{\pi}(s_t) = \max(\bar{p}, P^{\pi}) \tag{2.12}$$

$$= \max\left(\bar{p}, \mathbb{E} \sum_{t=0}^{\infty} \gamma^t p(t)\right) \tag{2.13}$$

where $\bar{p} \leq P_0$. Note that the second value function changes over time. Additionally, $\forall s_t, \quad P^{\pi} \;\leq\; P_0 \;\implies\; \mathbb{E}P(s_t) \;\leq\; P_0$, but not necessarily the other way around ($\impliedby$). Hence, the output of this approach could be sub-optimal [Gei06].

Another criterion is the risk-sensitivity. It is a weighted approach that jointly maximizes the primary reward, while minimizing the second one (i.e., the risk). This is achieved by introducing an empirical parameter w (also known as sensitivity parameter), so that the objective is to maximize the modified reward $g(t)$, where

$$g(t) = r(t) - w \cdot p(t) \quad . \tag{2.14}$$

Similar to the risk-neutral criterion, the weighted approach may lead to a sub-optimal solutions. Moreover, for problems with multi-objective optimization, the

w in the weighted approach controls (but does not guarantee) the Pareto frontier [KW06]. In Section 8.2, I use constrained RL and show how the sensitivity parameter can be used to optimize RL performance.

Part II

Wireless Virtual Network Embedding

In this part, I propose to exploit the computation capabilities of the surrounding smart devices to offload the processing from the cloud server. In this context, a processing request is broken down into smaller jobs. Hence, I need to answer the following questions: Which nodes should be selected for processing the jobs and how is data routed from one job to the next, possibly running on different nodes? In [AAK18], I provide a formal problem formulation using a mixed-integer quadratically-constrained optimisation problem. The main highlights of this formulation are that it

- exploits the multi-cast property of the wireless infrastructure for routing and
- allows cycles in the application graph (these cycles are needed for applications that require feedback loops in their algorithms).

Additionally, a novel embedding heuristic is proposed that uses continued fraction [BT17] for routing. The heuristic showed a close performance to the optimization model.

The work in [AK19b] adds power allocation to the problem, yet it simplifies the formulation to be mixed-integer linear programming (instead of the quadratic formulation from above). Moreover, the heuristic is simplified to be faster. This overcomes the limitation, in [AAK18], of evaluating a maximum of 6 wireless nodes in the infrastructure. The approximation ratio of this heuristic (without power allocation) has been derived in [AK19a]. Additionally, I compare in the latter [AK19a] the symbol/packet error rate of the proposed formulation to the related work. Consequently, I show that the proposed solution performs better because it considers the SINR (and the resulting collision probability) to change based on the transmitting nodes, unlike in related work that considers the probability of collision to be fixed between the nodes.

A meta-heuristic is proposed in [AHK19] using genetic algorithms, which also performs well compared to the optimization problem. An additional feature compared to the heuristic in [AK19a] is that the genetic algorithm can provide sub-optimal solutions in case of early termination. This is useful for online optimization where we may sacrifice optimality to find just feasible solution. Similarly, another heuristic is proposed in [AK20] using RL. Here, RL is used only for placement, while shortest-path-first is used for routing. The performance compared to the optimization model is still good and more importantly, the run-time is much shorter compared to all other proposed heuristics.

So far we have been dealing with one request at a time. But when the load of incoming requests to reserve resources increases, it becomes challenging to decide which requests should be admitted and which one should be rejected. In [ASK21],

I propose an RL solution for admission control that maximizes the acceptance rate and also consider the requests' priorities. Then, I show that the RL solution outperforms a simple first-come-first-serve admission control.

In all of the aforementioned work, I assume that the used MAC protocol is a collision-free TDMA with perfect synchronization. But TDMA is not popular within wireless sensor networks protocols due to its signalling overhead compared to CSMA/CA protocols, despite their potential collisions. Yet, since synchronization is also an issue for distributed microphones in acoustic sensor networks, what if we use the synchronization from the acoustic applications to synchronize the wireless transmission? That way, we could have the benefits of a collision-free TDMA without its overhead – but only if the acoustic synchronization is good enough to keep collisions probability reasonably small. In [Afi+22], I investigate the probability of collision due to synchronization error when using acoustic synchronization. Additionally, I compare the throughput to that of CSMA/CA and show that using TDMA – synchronized via acoustic applications – yields a higher throughput than CSMA/CA.

3 Optimization Problems for Resource Allocation

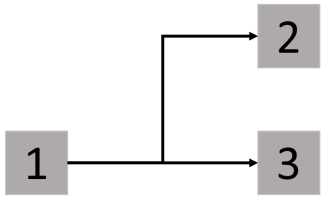
Embedded devices have evolved from processing-limited units to computing devices capable of doing heavy processing. Accordingly, wireless sensor nodes benefit from such evolution by relying on on-device processing [Qua19]. For instance, mobile phones are attached to many sensors (e.g., microphones, cameras, GPS, etc.), and many home appliances are supported with wireless connectivity and connected to sensors (e.g., light, motion and electrical sensors). A common property for all of these devices is that they are running open-source firmware or supported with APIs, which allows us to add extra functions for data processing, rather than limiting their functionalities to data collection.

This idea of in-network processing has been considered in WSNs before but typically for much simpler applications than signal processing applications (be it acoustic or otherwise). Typically, only simple aggregation functions and low data rates were investigated [LFZ13; PAA19]. The scenario here is more challenging as the processing requirements of signal processing jobs can differ substantially, as do data rate and delay requirements between them.

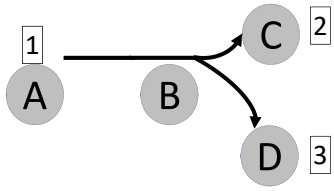
A similar idea of in-network processing is currently considered in the context of Network Function Virtualization (NFV), coming from wired networks. Solutions from that field are not easily applicable due to the inherently different characteristics of wired and wireless networks. Just to name a few: in wired networks, paths between the nodes have dedicated bandwidth while all nodes in wireless networks share the same bandwidth. Hence, wireless networks have an extra challenge since they need to handle interference from simultaneously transmitting nodes. Nevertheless, sharing the medium can be an opportunity, since it allows using multicast transmissions, which is not popular (nor recommended) in wired networks.

Formally, the problem is related to *virtual network embedding* (VNE) [Fis+13]: given a wireless network modelled as an *infrastructure graph* and a distributed application modelled as an *application graph*, map the jobs of the application to nodes of the infrastructure and map links of the application to paths in the infrastructure, under typical node and link capacity constraints. Figure 3.1 illustrates an example scenario for the differences: The signal processing jobs 1, 2, and 3 are mapped to the nodes A, C, and D, respectively; job 1 sends the same data to jobs 2 and 3, which might be exploited by cleverly multicasting from B.

The main difference between wired and wireless VNE arise from the wireless

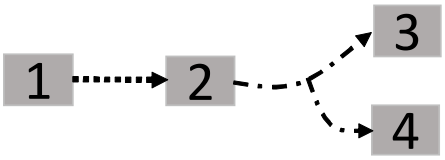


(a) Application graph.

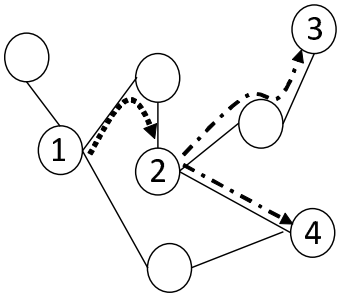


(b) Multicast in infrastructure.

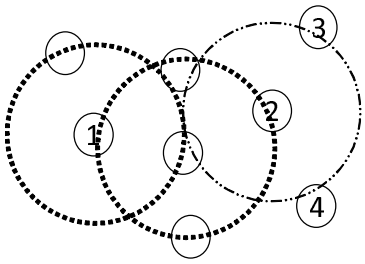
Figure 3.1: VNE maps processing jobs to nodes.



(a) Application graph.



(b) Wired VNE.



(c) Wireless VNE.

Figure 3.2: Wired vs. wireless virtual network embedding.

Table 3.1: Summary of wireless VNE

	VNE	Shared medium	Multicast	wireless parameter
[BAD14]	single link embedding	yes	No	time and frequency
[Sel+17]	only link embedding	yes	No	stochastic data rate per path
[Abd+16a]	Node and link embedding	limited to direct nodes	No	fixed bandwidth per path
[Rig+16]	Node and link embedding	limited to receiving nodes and not interfering nodes	No	fixed bandwidth per node
[SAC13]	Node and link embedding	yes	No	SINR between the nodes
[Li+16]	Node and link embedding	yes	No	SINR between the nodes
[Lv+12]	Only a single multi-cast no multi-hop jobs	No	Yes	fixed paths reliability between the nodes
[Gao+15]	Only a single multi-cast, no multi-hop jobs	No	Yes	fixed paths reliability between the nodes
[Li+17]	Node and link embedding	No	Yes	fixed delay per path

nature when one active path interferes on neighbouring nodes [BAD17]. This fact was compensated for in [BAD14][Sel+17] by assuming a perfect interference cancellation mechanism running at nodes. But such perfect interference cancellation is not necessarily available in WSNs, making this approach not directly applicable.

The work in [Abd+16a][Rig+16] also studied the wireless VNE problem, but they assumed a limited interference model, only neighbours who directly connect to a node are considered to be interfering with this node. However, when nodes operate in the same collision domain (i.e., same space, spectrum, time), this assumption oversimplifies the problem due to two reasons. First, the interference of one path on its neighbours may break the connectivity of neighbours' paths. Second, even nodes that do not have a direct connection still contribute to the interference.

Although the authors of [SAC13] [Li+16] considered interference from all neighbouring nodes when assigning jobs to node, their network flow model follows the flow conservation rule, ignoring the wireless multicast advantage.

In [Lv+12], the authors proposed a heuristic solution for the wireless multicast problem, while in [Gao+15] the authors proposed an exact MILP formulation for a multi-cast VNE problem. However, neither solution directly supports multi-hop paths. This is rectified by the authors of [Li+17] who formulated the multi-hop paths using a non-linear formulation. Nevertheless, they all assumed that the packet loss ratio for each wireless path is fixed; overlooking the dependency between packet loss ratio and the signal to interference noise ratio (SINR). I summarize the related work in Table 3.1.

3.1 Placement, Routing and Scheduling

The following formulation and results are inherited from [AAK18].

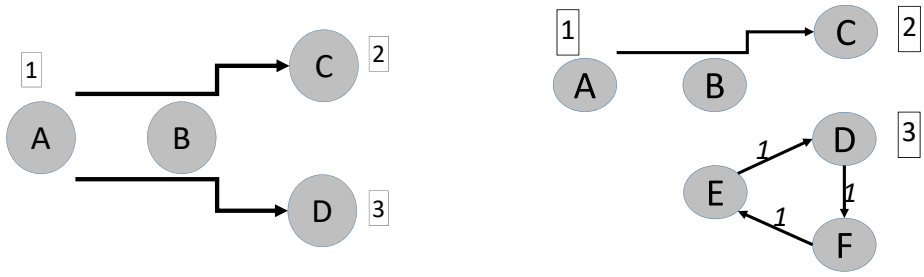
The infrastructure represents the physical network consisting of multiple nodes with wireless connectivity, where multiple application may run on a single infrastructure. The **infrastructure graph** is defined as $G_I = (V, T, \Gamma, C, \mathrm{SINR}_{th}, N_o)$. Each node $v \in V$ has a capacity $c_v = C(v)$ (representing resources such as memory and CPU). Moreover, I define $\left[v_{\mathrm{src}_1} \dots v_{\mathrm{src}_M}\right]$ and v_{sink} for M sources and a sink node, respectively. The former can sense input signals (e.g., via a microphone); the latter is a gateway node. Later on in Chapter 8 I look into how to select this set of sources when there are multiple available ones.

For simplicity, all nodes transmit with the same transmission power, and have an identical normalized noise floor N_0. The transmission power assumption will be generalized in Section 3.2. The matrix Γ contains the long-term average, slowly varying attenuation $\gamma_{v,v'}$ between any two nodes $(v, v'); v \neq v'$. I assume central knowledge of Γ, updated within its coherence time, and consider potential fast-fading phenomenon to be handled by lower-layer mechanisms.

A time-slotted model is assumed where transmissions take place in distinct time slots $t \in T$ grouped into time frames; $|T|$ is the maximum number of slots in a time frame. Consequently, a time frame is the duration a node has to wait before it can send again. All nodes are perfectly synchronized to these slots. For a node to receive at time slot t at a given, desired transmission rate R bit/s , its SINR must exceed a given threshold $\mathrm{SINR}_{\mathrm{th}}$ to enable transmission at negligible error rate. We only consider a single transmission rate here; this is not difficult to generalize.

The **application graph** denotes the distributed application as a directed graph $G_O = (P, L, W)$. Each processing job $p \in P$ requests node resources given by $w_p = W(p)$. Job p sends the same data to all its successors p' with $(p, p') \in L \subset P \times P$. A link (p, p') needs a data rate of at most $R/|T|$ bit/s; this effectively means that a link can be scheduled in a single time slot per time frame. Extending this model to different link rates and spreading one transmission over multiple time slots or grouping links in a time slot is not difficult but requires notation that is a bit cumbersome. As this would detract from the core points here, I will focus on the scenario at hand. Similar to the infrastructure graph, I define $\left[p_{\mathrm{src}_1} \dots p_{\mathrm{src}_M}\right]$and p_{end} as the M source and sink jobs of the application. The source jobs are assigned to the predefined source nodes and the sink job is assigned to the sink node, so that they define the start and end points of the traffic flow of the application graph

The task is now to map jobs to nodes and overlay links to infrastructure paths. For job mapping, typical capacity constraints hold (Section 3.1.1). For link mapping,



(a) Flow conservation **(b)** Unwanted loops

Figure 3.3: Wireless-VNE flow challenges

one would be tempted to use common flow conservation constraints: a node's incoming paths equals its outgoing ones unless a job is placed on this node.

However, this would not do justice to the goal of leveraging multicast. Let us reconsider Figure 3.1. Node B receives one flow from A but has to forward it to C and D. Under the flow conservation rule, it could only do that if it received the flow from A twice, but that is wasteful (Figure 3.3 (a)). Hence, we have to loosen the flow conservation restriction by allowing a node to forward *at least as much* traffic as it has received.

This relaxation, however, ensues an unfortunate consequence. Consider Figure 3.3 (b), which shows a loop of job's A traffic being forwarded among nodes D, E, and F. While conventional flow conservation rules would prevent such a loop (as block 3 on D would remove this flow), under this relaxed rule, this loop is consistent with all constraints (say, node D receives flow 1, pushes it into its job 3 but D *also* forwards it to F; flow 1 at E and F is balanced). Hence, I need to come up with additional constraints to prevent such loops. In doing so, I also have to deal with loops deliberately created in the overlay graph (Figure 3.4).

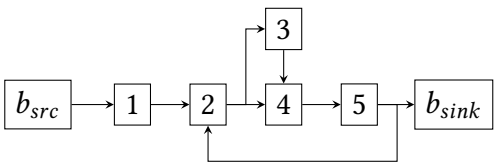


Figure 3.4: Overlay graph from acoustic signal processing [Sch+17a] with a loop from block 5 to 2

3.1.1 Optimisation Problem

This section formalises the above model as an optimisation problem. I target multicast flow embedding in combination with virtual network embedding, where multiple outputs of the same job have the same traffic. In order to focus on this feature, I ignore the fact that multiple outputs from a job may carry different traffic. The VNE solution is not concerned with scaling out of processing jobs, and allows multiple jobs to be placed on same nodes (many to one) but not vice versa (one to many; an application job cannot be automatically distributed over multiple nodes).

Decision variables

Given the infrastructure and application graphs, I use a binary variable $\theta \in \Theta = P \times V$ for placing a job p on an actual node v.

Additionally, I use a binary variable $f \in F = P \times V \times T$ to indicate if the output traffic of job p is either originated or forwarded by node v at time slot t ($f(p,v,t) = 1$).

The binary variable $s \in S = V \times V \times P \times V \times T$ is used for scheduling and path routing; $s(v_1, v_2, p, v_3, t) = 1$ if and only if node v_1 is sending to v_2 at time slot t the output traffic of job p, which is placed on node v_3. Otherwise, $s(v_1, v_2, p, v_3, t)$ is equal to 0.

In fact, s fully determines θ and f, which are mostly for conceptual and notational convenience. For additional convenience, I use a binary variable $\beta \in B = T$ to indicate if a time slot t is used or not.

Constraints

I group the constraints into four main groups. **First**, I define variable interdependency between s, f, and β (variables s and θ are coupled via the flow constraints later on). The relationship $f(p,v,t) > 0 \Leftrightarrow \sum_{v_i \in V \setminus v} \sum_{v_j \in V} s(v, v_i, p, v_j, t) > 0 \, \forall p, v, t$ is expressed by the constraints (3.1) and (3.2) by means of a big-M construction. In other words, $f(p,v,t) = 1$ if and only if $s(v, v_i, p, v_j, t) = 1$. Similarly, (3.3) and (3.4) expresses $\beta(t) > 0 \Leftrightarrow \exists v_i, v_j, v_k, p : s(v_i, v_j, p, v_k, t) > 0 \, \forall t$. So that $\beta(t) = 1$ if and only if $s(v, v_i, p, v_j, t) = 1$

$$\sum_{v_i \in V \setminus v} \sum_{v_j \in V} s(v, v_i, p, v_j, t) - f(p,v,t) \geq 0, \quad \begin{matrix} \forall v \in V \\ \forall p \in P \\ \forall t \in T \end{matrix} \tag{3.1}$$

$$\sum_{v_i \in V \setminus v} \sum_{v_j \in V} s(v, v_i, p, v_j, t) - \mathcal{M} \cdot f(p,v,t) \leq 0, \quad \begin{matrix} \forall v \in V \\ \forall p \in P \\ \forall t \in T \end{matrix} \tag{3.2}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} \sum_{p \in P} \sum_{v_k \in V} s(v_i, v_j, p, v_k, t) - \mathcal{M} \cdot \beta(t) \leq 0 \quad \forall t \in T \tag{3.3}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} \sum_{p \in P} \sum_{v_k \in V} s(v_i, v_j, p, v_k, t) - \beta(t) \geq 0 \quad \forall t \in T \tag{3.4}$$

Second, I ensure node mapping and adequate wireless communication in constraints (3.5) – (3.8).

In (3.5) and (3.6), I ensure that a job is placed only once and the nodes' capacity constraints are not violated, respectively. Since I assume unit data rate between jobs, (3.7) ensures that only one job's traffic is sent by a node in one time slot. Also, a node is allowed either to transmit or receive in a given time slot (i.e., half-duplex radios). In (3.8), I have the only quadratic constraint, which allows transmissions from node v to v' if the SINR at v' is bigger than or equal to SINR_{th}. In this check, I consider interference from all nodes except the transmitting node v.

$$\sum_{v \in V} \theta(p, v) = 1, \quad \forall p \in P \tag{3.5}$$

$$\sum_{p \in P} \theta(p, v) \cdot w_p \leq c_v, \quad \forall v \in V \tag{3.6}$$

$$\sum_{p \in P} f(p, v, t) + \sum_{v_i \in V} \sum_{p \in P} \sum_{v_j \in V} s(v_i, v, p, v_j, t) \leq 1, \quad \begin{array}{c} \forall v \in V \\ \forall t \in T \end{array} \tag{3.7}$$

$$\sum_{p \in P} \sum_{v_i \in V} s(v, v', p, v_i, t) \cdot \text{SINR}_{\text{th}} \leq \frac{\sum_{p \in P} \sum_{v_i \in V} s(v, v', p, v_i, t)}{N_o + I(v, v')},$$

$$\text{where } I(v, v') = \sum_{p \in P} \sum_{\substack{u \in V \\ u \neq v}} f(p, u, t) \cdot \gamma_{u,v'}, \quad \begin{array}{c} \forall v, v' \in V \\ \forall t \in T \end{array} \tag{3.8}$$

Third, I check flow constraints in the infrastructure graph. I consider in these constraints the mapping of the application's links $l = (p_1, p_2)$ to a path between nodes.

Constraint (3.9) checks that a node v has received p_1's traffic, irrespective from which node v_i, before placing p_2 on node v. For flow control, (3.10) ensures that when node v receives p_1's traffic, it will either forward this traffic or place p_2 on v. Conversely, (3.11) allows node v to send job p's traffic only if v has received p's traffic or p is placed on v. Note that when not supporting multicast, these three inequalities collapse into one equality constraint.

$$\sum_{v_i \in V} \sum_{v_j \in V} \sum_{t \in T} s(v_i, v, p_1, v_j, t) - \theta(p_2, v) \geq 0, \quad \begin{array}{c} \forall v \in V \\ \forall (p_1, p_2) \in L \end{array} \tag{3.9}$$

$$\sum_{v_i \in V} \sum_{t \in T} s(v_i, v, p_1, v_j, t) - \sum_{(p_1,p_2) \in L} \theta(p_2, v) - \sum_{v_i \in V \setminus v} \sum_{t \in T} s(v, v_i, p_1, v_j, t) \leq 0, \quad \begin{array}{c} \forall v, v_j \in V \\ \forall p_1 \in P \end{array} \tag{3.10}$$

$$\sum_{v_i \in V} s(v, v_i, p, v', t) - \mathcal{M} \sum_{v_i \in V \setminus v} \sum_{t_i \in T} s(v_i, v, p, v', t_i) - \mathcal{M}\theta(p, v) \leq 0, \quad \begin{array}{c} \forall v, v' \in V \\ \forall p \in P \setminus p_{\text{sink}} \\ \forall t \in T \end{array} \tag{3.11}$$

Fourth, I need to exclude loops as in Figure 3.3 (b) (page 31). In that figure, e.g. F cannot really send to E the traffic originated by job 1 on A – because there is no active path via which F could receive 1's traffic. Checking whether such a path exists is not obvious. Constraints for a maximum path length of 1 are easy to write down; it gets more and more complex the longer I allow the paths to become. Hence, lest I have to write down all these constraints manually, I use Algorithm 1 to construct these constraints systematically.

Algorithm 1's goal is to generate a constraint expressing whether a node v_{start} has received job p's traffic originating from node v. To do so, it generates all possible infrastructure paths from v to v_{start} via a depth-first search (DFS). For each path, a conjunctive constraint is produced to check whether *each* node on this path has received this traffic. Then, I need to check whether at least one of all these paths does carry the traffic; this is expressed by a disjunction of these conjunctions.

The challenge to expressing conjunctions and disjunctions is to find a linear form for it. A conjunction between variables x_i, $i = 1, \ldots, n$ (with 0=False and True, otherwise) can be expressed as $\frac{1}{2^n} + \sum_{i=1}^{n} \frac{x_i}{2^i} \geq 1$. A disjunction corresponds to $\frac{1}{2} + \sum_{i=1}^{n} \frac{x_i}{2} \geq 1$.

In the present case, x_i corresponds to the fact that a node v_2 receives a particular job p's traffic (placed at some v) from a neighbour v_1, irrespective of the time slot. This corresponds to $\sum_{t \in T} s(v_1, v_2, p, v, t)$ being zero or larger. These sums have to be computed, for every p, for (v_1, v_2) along all possible paths starting from a particular node v_{start} under consideration to p's hosting node (i.e., v). This happens by calling Algorithm 1 recursively with parameters $(v_{\text{start}}, v_{\text{end}}, p, v, \text{visitedNodes}, r)$. visitedNodes represents nodes on the *currently* considered path from v_{start} to v; r is the recursively constructed conjunction along this path. Algorithm 1 constructs the conjunction sum as a continued fraction, which is simpler to do in a recursive algorithm. It produces the terms stated above. For details, please see the listing of Algorithm 1.

The result of this algorithm, called for all node combinations, is then forming the following constraint:

$$\text{trackFlow}(v_{\text{start}}, v_{\text{end}}, p, v, \{v_{\text{start}}, v_{\text{end}}\}, 1) \quad \begin{matrix} \forall v_{\text{start}} \in V \setminus v_{\text{end}} \\ \forall v_{\text{end}} \in V \setminus v_{\text{start}} \\ \forall p \in P \\ \forall v \in V \setminus \{v_{\text{start}}, v_{\text{end}}\} \\ \forall t \in T \end{matrix}$$

$$- s(v_{\text{start}}, v_{\text{end}}, p, v, t) \geq 0 \tag{3.12}$$

The initial value of r is equal to one and any new path updates r as follows $r_{\text{new}} = \frac{r+x_i}{2}$. Consequently, the value of r falls in range $[0, 1]$.

Figure 3.5 depicts the algorithm's progress. Given 4 fully connected nodes, I check if node A (i.e., v_{start}) can send to node B (i.e. v_{end}) traffic of job p that is originated by node C (i.e. v). There are two available routes from C to A (directly and via D), but only one route may be selected so that they do not conflict with (3.10) and (3.11). Each route's availability is characterised by the fractional terms on the right.

Algorithm 1: trackFlow function

Input : v_1, v_2, p, v, visitedNodes, r
Result: Constraint for possible paths in a DFS tree

```
if v_1 = v then
    return (Σ_{t∈T} s(v_1,v_2,p,v,t) + r) / 2
visitedNodes = visitedNodes + {v_1}
sum = 0
foreach v_i ∈ V \ visitedNodes do
    r_new = (Σ_{t∈T} s(v_i,v_1,p,v,t) + r) / 2
    sum = sum + trackFlow (v_i, v_1, p, v, visitedNodes, r_new)
return sum
```

3.1.2 Objective

The **objective** is to minimize the number of used time slots; min $\sum_{t \in T} \beta(t)$. This reflects latency requirements of typical signal-processing or real-time applications.

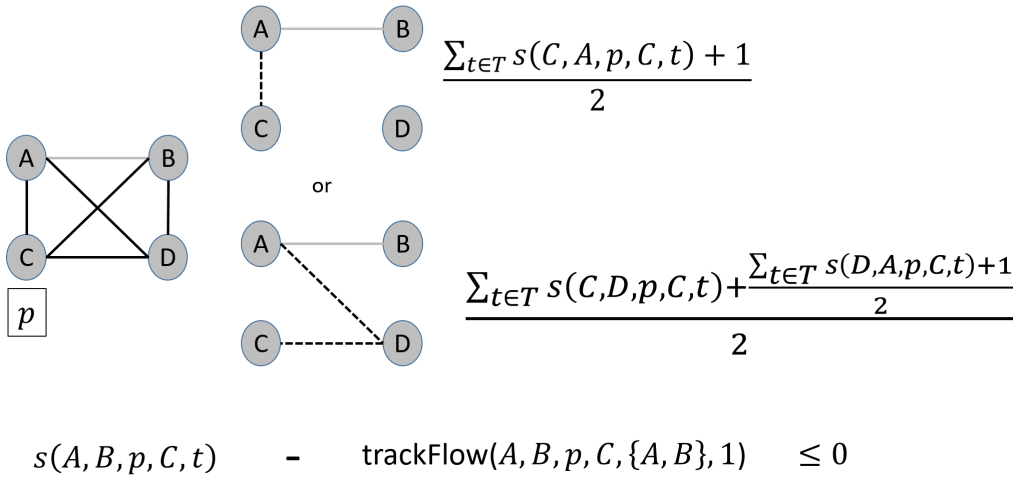


Figure 3.5: Example for Algorithm 1: p is placed on C, can A send to B the traffic of p originated by C?

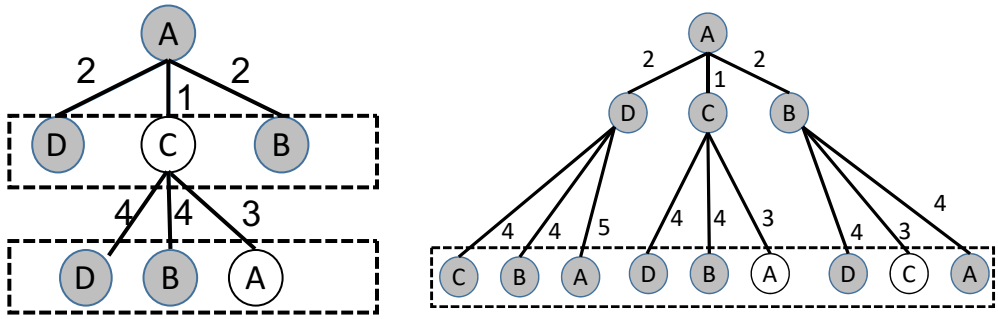
3.1.3 A Backtrack Heuristic

Overview

The core idea is to start from any of the source jobs, progress from one application link to another in a topological order, mapping the link to a path in the infrastructure and mapping jobs to nodes in the same step. More precisely, when we map a link (p_1, p_2), there are two cases: (1) The receiving job has already been placed, then link mapping just means finding a path between the two nodes hosting p_1 and p_2. (2) If p_2 is not yet placed, I also have to find a hosting node for p_2, jointly with finding a path towards that node.

We can hence think about this as (an application graph) a link mapping problem, where we progress from link to link. Whenever a link has been mapped, we have choices for mapping the next link (to different paths, or to different nodes and paths). This is a search problem in a tree of possible link mapping decisions. A sequence of link mapping decisions that maps all links constitutes a feasible solution.

A brute-force algorithm to find the optimal solution would have to explore this tree in its entirety. This is clearly not feasible. Hence, I introduce three control mechanisms to limit the search space: lookahead, backtracking, and degree of this search tree.



(a) 1-*level* lookahead **(b)** 2-*level* lookahead

Figure 3.6: Looking ahead in the link mapping search tree (dashed boxes indicate levels of decision making)

Lookahead level

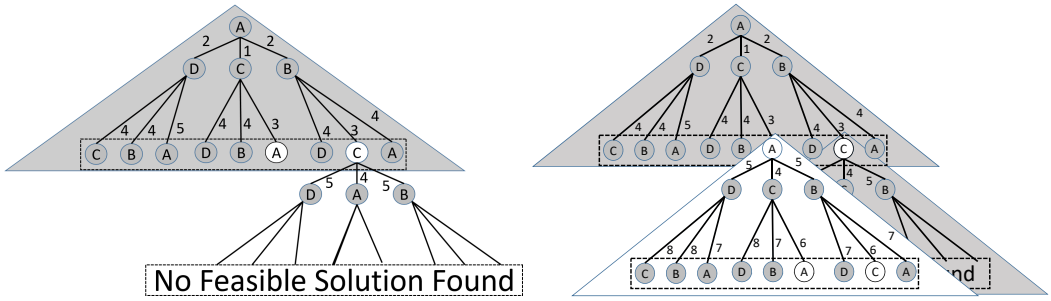
Suppose we have committed to the mapping of a link l_1. To determine how to map the next link l_2, we could just look at the options for this link and pick the best possible option. In addition, we could also *look ahead*: I consider all possible options for mapping the next *level* many links, exploring an entire subtree. Among those possible mapping combinations, I choose the best one and map *level* many links in a single step. Figure 3.6 shows examples for 1-level and 2-level lookahead mapping (note that numbers in these figures indicate the total number of used time slots and the circles are *link mappings*, not nodes!). When *level* equals the number of links, this scheme degenerates into exhaustive brute-force search.

Limit tree degree

As a second parameter, I limit the degree of the link mapping search tree. When mapping a link (p_1, p_2) with p_1 hosted on v_1, and trying to find a node v_2 to host p_2, I only look at the k neighbors of v_1 with best attenuation values.

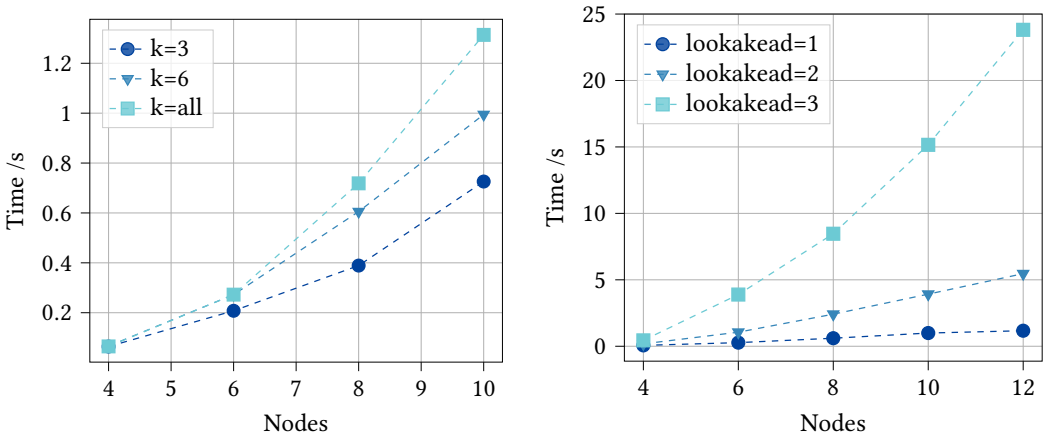
Backtracking

Suppose we have mapped *level* many links and try to find a solution for the next *level* links. What happens if no feasible solution can be found? As is typically done, I backtrack, reject the decisions taken for the previous mapping of *level* many links, and start again with the remaining best solution. This is illustrated in Figure 3.7.



(a) Mapping failed **(b)** Restarting after backtracking

Figure 3.7: Backtracking in the link mapping search tree



(a) *k* execution time **(b)** n-*level* execution time

Figure 3.8: Execution time for different configuration

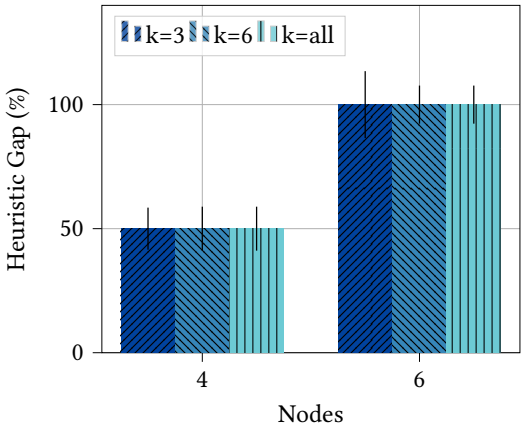


Figure 3.9: 1-*level* gap

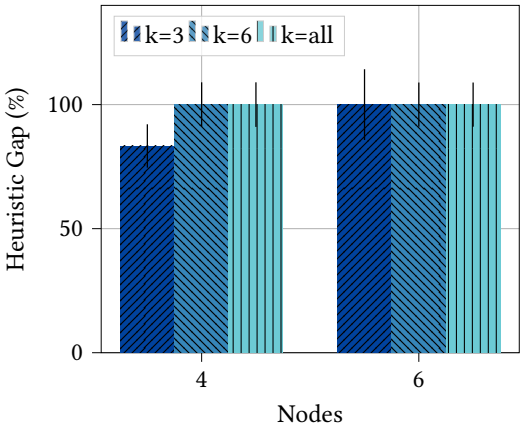


Figure 3.10: 2-*level* gap

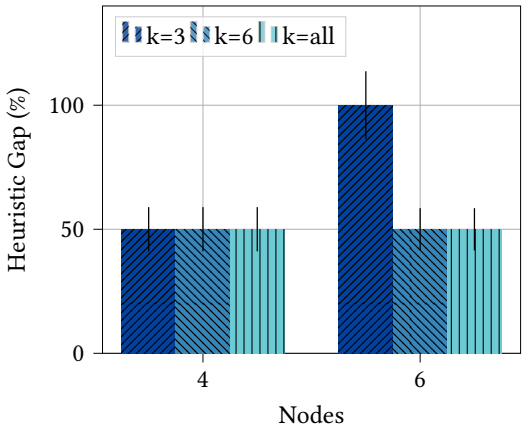
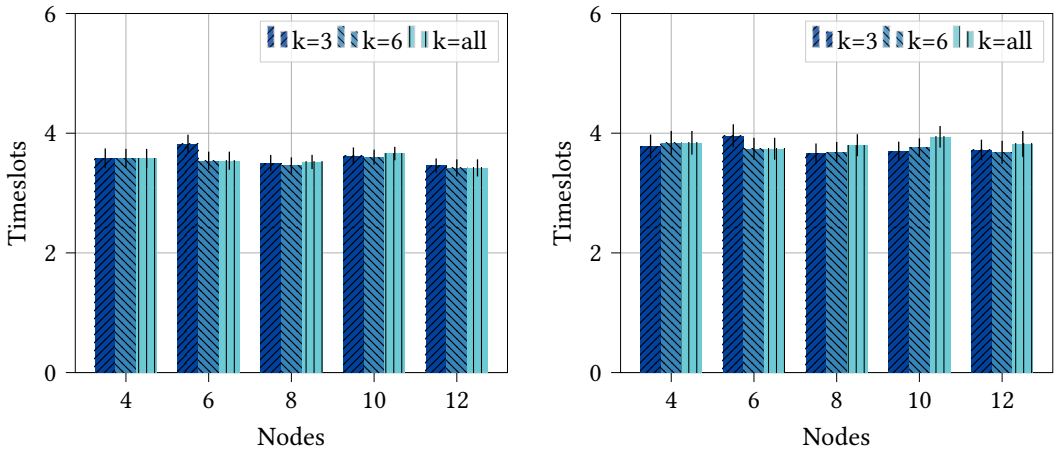
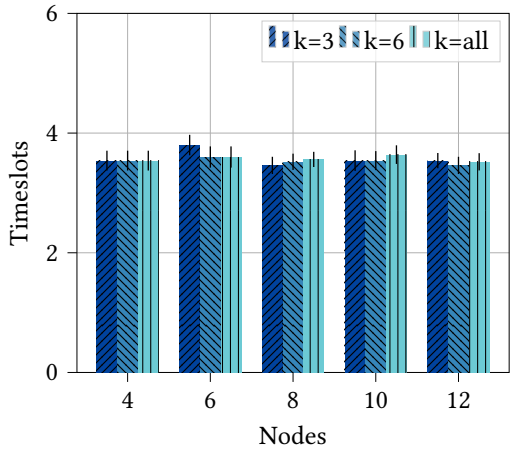


Figure 3.11: 3-*level* gap



(a) 1-*level* used slots

(b) 2-*level* used slots



(c) 3-*level* used slots

Figure 3.12: Average number of used slots using the heuristic algorithm

3.1.4 Evaluating Backtracking Heuristic

Given the optimal solution, my main objective here is to evaluate the performance of the heuristic solution in terms of execution time and the optimality gap.

Scenario

The exact formulation is solved using Gurobi Optimizer 7.5; the heuristic is implemented in Python. All simulations were executed in single-threaded mode on Intel Xeon X560 cores running at 2.67 GHz. The simulated environment consists of a room with area 25×25 m^2, where nodes are placed uniformly at random, independently from each other. The attenuation between two nodes $v, v' \in V$ is given by $\gamma_{v,v'} = \frac{1}{d^2_{v,v'}}$, where $d_{v,v'}$ is the distance between the two nodes.

I vary the number of nodes and run 50 independent realizations for each number of nodes. In each realization, node capacities c are uniformly distributed $c(v) \sim U(\max(w_p), \sum_{p \in P} w_p)$, ensuring that each node can at least host even the heaviest job. Furthermore, v_{src} and v_{sink} nodes are picked randomly per realization.

I choose a generic algorithm from the field of acoustic signal processing for the application graph [Sch+17a]; a typical algorithm used for synchronizing the clocks of wireless microphones. Figure 3.4 depicts the application graph with 5 jobs equally weighted $w_p = w_o$. I add two artificial jobs p_{src} and p_{sink} to assign to v_{src} and v_{sink}; $w_{p_{\text{src}}} = w_{p_{\text{sink}}} = 0$. The overlay graph has a typical use case for multi-cast routing between blocks 2–3 and 2–4, and a feedback from nodes 5 to 2.

Execution Time

Figures 3.8 (a) and 3.8 (b) evaluate the heuristic's median runtime for different configuration setups. First, in Figure 3.8 (a), I set the lookahead *level* to 1 and vary the search tree degree k between 3, 6, and all neighbours. It is observed that the runtime increases exponentially in the number of nodes. Moreover, limiting the search space to k neighbours reduces the execution time significantly as k decreases.

I investigate the impact of increasing the lookahead *level* in Figure 3.8 (b). I observe that increasing the *level* from 1 to 2 has a higher impact than increasing the number of neighbours k. The median execution time (over 50 runs) of the solver for an optimal solution is 140 seconds for 4 nodes and 2564 seconds for 6 nodes. Therefore, I limit the exact model's evaluation to 6 nodes.

Schedule length: Heuristic vs. optimal solution

I compare the heuristic and the exact model via the approximation ratio $\frac{\sum_{t\in T}\beta_h(t)-\sum_{t\in T}\beta_{opt}(t)}{\sum_{t\in T}\beta_{opt}(t)}$, where $\sum_{t\in T}\beta_h(t)$ and $\sum_{t\in T}\beta_{\text{opt}}(t)$ are the heuristic and optimal (with no gap) solution's schedule length per realization.

In Figures 3.9 and 3.10, I show the 95% confidence interval for the mean gap with 4 and 6 nodes for different k. When using 1-*level* lookahead (in Figure 3.9), 2-*level* (in Figure 3.10) and 3-*level* (in Figure 3.11)

First, when the number of nodes is 4 and for every k, the 1-*level* lookahead has lower (better) gap than 2-*level*. Hence, increasing the *level* number does not necessarily yield fewer time slots (except when *level* = $|L| \leftrightarrow$ brute-force).

Although the heuristic gap increases for 2-*level* lookahead, limiting the number of neighbors $k = 3$ has a lower heuristic gap for 4 nodes. The reason is the random selection during early link mapping, when two or more mappings give the same minimal additional number of time slots. Yet progressing with link mapping may yield higher time slots, (which is not anticipated with low lookahead levels). As the search space shrinks (i.e., decreasing k), such randomness becomes more guided towards nearby nodes. Another way of mitigating such behavior may be increasing the lookahead *level* to anticipate future embedding in the application graph. As seen in Figure 3.11 the heuristic gap decreases for both 4 and 6 nodes compared to Figure 3.10.

Schedule length in large scenarios

I further investigate the schedule length using the heuristic for many nodes. Figures 3.12 (a) and 3.12 (b) show that (a) the average schedule length does not change significantly with the number of nodes, (b) the *level* number has an impact; 1-*level* has smaller mean gap than 2-*level* lookahead. This confirms our hypothesis in Section 3.1.4: increasing the *level* number does not have to yield a better solution. Furthermore, I observe that limiting k yields (in most cases) a shorter schedule. An exception would be in Figure 3.12 (a), $k = 6$ and 14 nodes yields longer schedules than considering all neighbouring nodes.

A **utilized node** is a node that hosts a job or that is just being used for traffic forwarding. Figures 3.13 (a) and 3.13 (b) show the number of utilized nodes for different number k of considered neighbours and lookahead *level* configurations. In Figure 3.13 (a), 1-*level* backtrack is used while considering all, 6 and 3 nearest neighbours. Similarly, in Figure 3.13 (b), I repeat the same experiment, but using 2-*level* backtrack.

As seen in both Figures 3.13 (a) and 3.13 (b), there is no clear interdependency

between the number of used nodes and the available nodes in the substrate network. Moreover, neither parameter, *level* and k, has a significant impact on the number of utilized nodes. Hence, increasing the *degree* number does not have to yield lower number of utilized nodes.

To sum up, I introduced a new formulation for the wireless VNE problem, suitable for multicast multi-hop environments. Since optimally solving such a problem is time consuming, I also proposed a heuristic algorithm that can be controlled using two parameters; level and k nearest neighbours. Changing both parameters can have a significant impact on the execution time, especially for the level parameter. Reducing the search space to the nearest k-neighbours does not have a substantial impact on the number of used time slots.

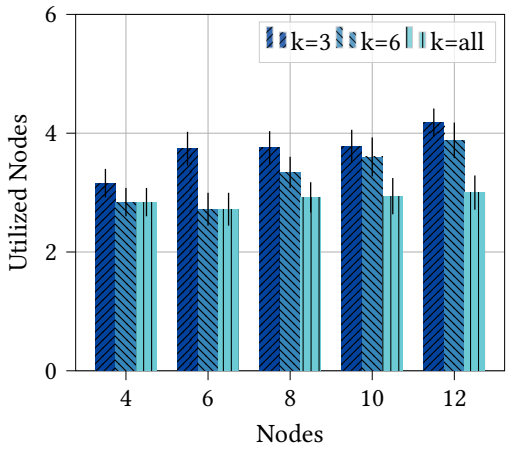
A typical use case of such analysis is for applications with fast embedding requirements. I have shown that the heuristic can get acceptable results using a setup that requires low execution time. In this case, a quick solution can be found for the wireless VNE problem. Then, it can be optimized by changing the level and k-neighbours parameters if more time is provided

3.2 Power Allocation

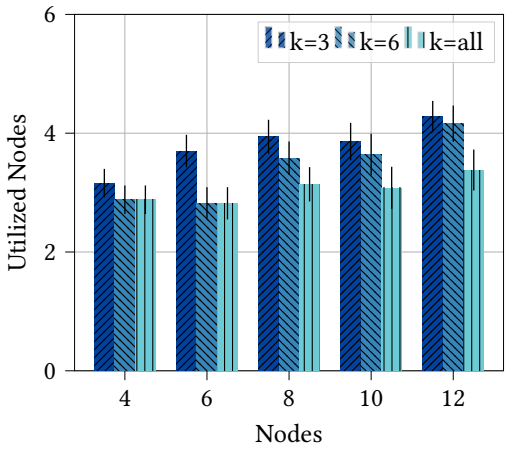
Earlier, I assumed that all nodes operate in the same collision domain. Meanwhile, using power control may allow to achieve more feasible results [AK19b]. Here, I reformat the optimization problem to be simpler and include power allocation. The simplicity comes from removing the continued fraction method and introducing two more decision variables h and λ, and redefining f (see Table 3.2), so that the resulted formulation is an MILP instead. The latter offers less complexity. The continued fraction solution requires less number of variables and more constraints, created by the continued fraction algorithm. This can find faster solution when using search techniques such as branch and bound. But, I need to highlight that creating this constraints also take some time which increases the complexity of creating the problem itself, especially with high number of nodes and links. Accordingly, the proposed formulation in this chapter is better to save resources. To include power control, the decision variable ω is introduced, yet the formulation then reverts back to an MIQP.

3.2.1 Constraints

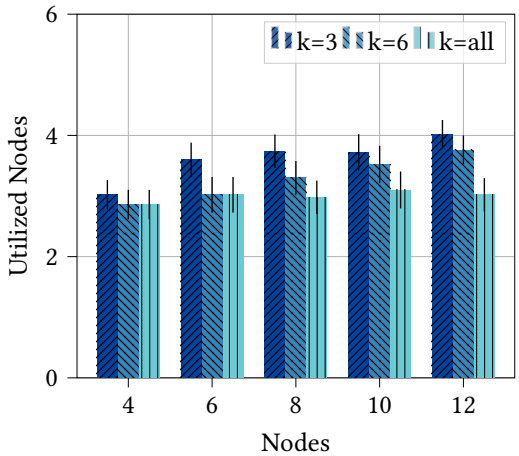
I group the constraints into three main groups so that I highlight here only the substantially new ones compared to Section 3.1.1. **First**, I define variable interde-



(a) 1-*level* backtrack



(b) 2-*level* backtrack



(c) 3-*level* backtrack

Figure 3.13: Number of used nodes using n-*level* backtrack

Table 3.2: Summary of Variables.

Variable	Description
$f(v,t)$	$\in \{0,1\}$ to determine if node v is transmitting at time slot t
$h(v_1, v_2, b_p)$	$\in \mathbb{N}$ to determine how many blocks needs to receive the b_p when node v_1 sends it to v_2
$\lambda(v, v', t)$	$\in \mathbb{Z}^+ \bigcup 0$ a McCormick helper variable to linearize the quadratic interference between (v, v') at t
$\omega(v,t)$	$\in [0,1]$ defines the transmit power for v at t

pendency, so I show here the interdependency between h and s in constraints (3.13) and (3.14).

$$h(v_i, v_j, b_p) - \sum_{t \in T} s(v_i, v_j, b_p, t) \geq 0, \qquad \begin{smallmatrix}\forall (v_i, v_j) \in V \times V \\ \forall b_p \in B_P\end{smallmatrix} \tag{3.13}$$

$$h(v_i, v_j, b_p) - \mathcal{M} \cdot \sum_{t \in T} s(v_i, v_j, b_p, t) \geq 0, \qquad \begin{smallmatrix}\forall (v_i, v_j) \in V \times V \\ \forall b_p \in B_P\end{smallmatrix} \tag{3.14}$$

Second, I check flow constraints in the infrastructure graph. In these constraints, I consider the mapping of an application link $l = (b_p, b')$ to a path between nodes. Constraint (3.15) checks that a source node v_{src} has virtually sent the traffic of b_{src} as often as the number of successor jobs. Similarly, constraint (3.16) ensures that the virtually received traffic by the sink node is at least equal to the number of required traffic from the predecessors' jobs of b_{sink}. Then, I give the flow balance for any other node in (3.17); for a given node, the incoming traffic is equal to the outgoing traffic unless jobs are mapped on this node.

$$\sum_{v' \in V} h(v_{\text{src}}, v', b_{\text{src}_p}) - \sum_{(b_{\text{src}_p}, b') \in L} \theta(b_{\text{src}}, v_{\text{src}}) = 0\,, \qquad \forall b_{\text{src}_p} \in b_{\text{src}_P} \tag{3.15}$$

$$\sum_{v' \in V} \sum_{(b_p, b_{\text{sink}}) \in L} h(v', v_{\text{sink}}, b_p) - \sum_{(b_p, b_{\text{sink}}) \in L} \theta(b_{\text{sink}}, v_{\text{sink}}) \geq 0 \tag{3.16}$$

$$\sum_{v' \in V} h(v, v', b_p) - \sum_{(b_p, b')} \theta(b, v) = \sum_{v' \in V \setminus v} h(v', v, b_p) - \sum_{(b_p, b')} \theta(b', v), \; \begin{smallmatrix}\forall v \in V \\ \forall b_p \in B_P \setminus \{b_{p_{\text{sink}}}, b_{p_{\text{sink}}}\}\end{smallmatrix} \tag{3.17}$$

Keeping track of the multicast requirements of a job via h is the key technique that allows a linearised multicast formulation, without using continued fraction. In other words, h virtually represents how often a job's output needs to be forwarded or multicasted, so that it allows a node to physically duplicate the traffic (via variable s), even if the traffic is received only once. In other words, h is a helper variable that follows the flow conservation rule (Figure 3.3 (a)), while s is physical and does not follow the flow conservation (Figures 3.1 (b) and 3.2 (b)).

The **Third** group concerns wireless interference constraints and depends on the transmission power.

Uniform transmit power – MILP

$$\sum_{b_p \in B_P} s(v,v',b_p,t) \cdot \mathrm{SINR_{th}} \leq \frac{f(v,t)}{N_o + I(v,v',t)} \gamma_{v,v'}, \qquad \substack{\forall \{v,v'\} \in V \\ \forall t \in T}$$

$$\text{where } I(v,v',t) = \sum_{\substack{u \in V \\ u \neq v}} f(u,t) \cdot \gamma_{u,v'} \tag{3.18}$$

$$\frac{f(v,t) \cdot \gamma_{v,v'}}{\mathrm{SINR_{th}}} \geq \lambda(v,v',t) + N_o \sum_{b_p \in B_P} s(v,v',b_p,t), \qquad \substack{\forall \{v,v'\} \in V \\ \forall t \in T} \tag{3.19}$$

$$\lambda(v,v',t) \leq I(v,v',t), \qquad \substack{\forall \{v,v'\} \in V \\ \forall t \in T} \tag{3.20}$$

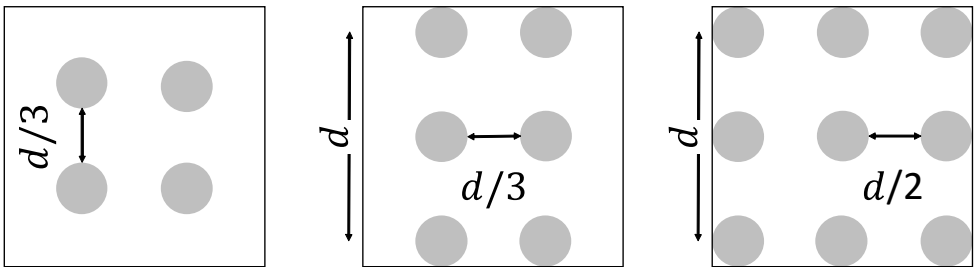
$$\lambda(v,v',t) \leq \sum_{b_p \in B_P} s(v,v',b_p,t) \cdot \gamma_{v,v'} |V|, \qquad \substack{\forall \{v,v'\} \in V \\ \forall t \in T} \tag{3.21}$$

$$\lambda(v,v',t) \geq I(v,v',t) - |V| \left(1 - \sum_{b_p \in B_P} s(v,v' b_p,t)\right), \qquad \substack{\forall \{v,v'\} \in V \\ \forall t \in T} \tag{3.22}$$

In (3.18), I formally control interference $I(v,v',t)$. This constraint allows transmissions from node v to v' if the SINR at v' is bigger than or equal to $\mathrm{SINR_{th}}$. It will, however, yield a quadratic constraint. Thus, I reformat it in (3.19) to be linearly constrained, and do not need to use (3.18) any more. The linearisation of (3.18) is inspired by the McCormick method for 0/1 quadratic problems [McC76]. First, λ defines the effect of interference on an active path as $\lambda(v,v',t) \leftrightarrow I(v,v',t) \cdot \sum_{b_p \in B_P} s(v,v',b_p,t)$. Given that the interference $I(v,v',t)$ is bounded between $[0, |V|]$, while $\sum_{b_p \in B_P} s(v,v',b_p,t) \in \{0,1\}$, I ensure in Equations (3.20) and (3.22) that $\lambda = I$ when $s = 1$ is active, else $\lambda = 0$ using Equation (3.21). Accordingly, I define the conditional boundaries of the integer variable $\lambda(v,v',t)$ through constraints (3.20) – (3.22).

Power allocation – MIQCP

I rewrite the SINR constraint to consider power allocation as in (3.23). I ensure in (3.24) – (3.25) that power allocation takes place if and only if the node is transmitting.



(a) 4 Nodes. **(b)** 6 Nodes. **(c)** 9 Nodes.

Figure 3.14: Examples of network distribution.

$$\sum_{b_p \in B_P} s(v, v', b_p, t) \cdot \mathrm{SINR}_{\mathrm{th}} \leq \frac{\omega(v,t)}{N_o + I(v,v',t)} \gamma_{v,v'}, \qquad \forall \{v,v'\} \in V \; \forall t \in T$$

$$\text{where } I(v,v',t) = \sum_{\substack{u \in V \\ u \neq v}} \omega(u,t) \cdot \gamma_{u,v'} \tag{3.23}$$

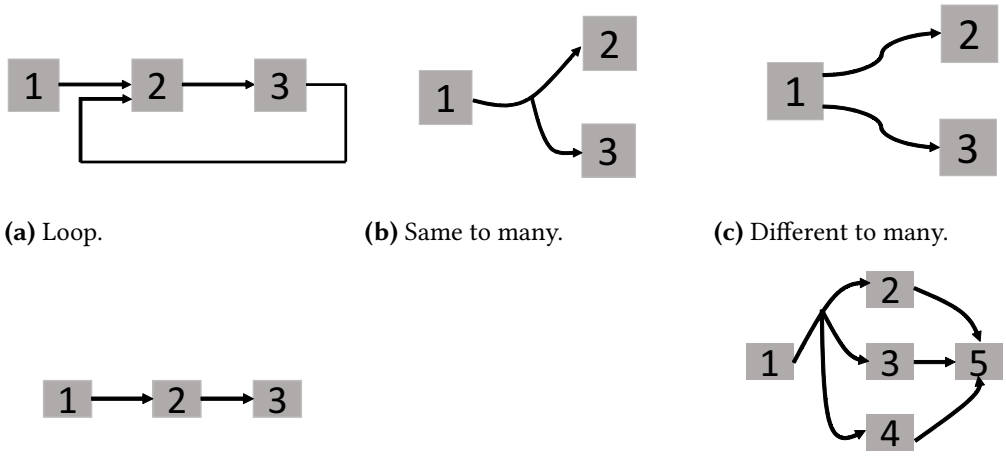
$$\omega(v,t) - f(v,t) \leq 0, \qquad \forall v \in V \; \forall t \in T \tag{3.24}$$

$$\mathcal{M} \cdot \omega(v,t) - f(v,t) \geq 0, \qquad \forall v \in V \; \forall t \in T \tag{3.25}$$

The **objective** is same as in 3.1.2, to minimize the number of used time slots within a time frame; min $\sum_{t \in T} \delta(t)$. This reflects latency requirements of typical signal-processing or real-time applications, so that traffic is received within the first few time slots in a time frame (or it can also be used for slicing in a multi-tenant network).

3.2.2 Fixed vs. Flexible Power Results

As an example, I consider a seminar room where the nodes are uniformly distributed over a grid (Figure 3.14). The attenuation between nodes $\gamma_{v,v'}$ is given by $\frac{1}{d^2_{v,v'}}$. The simulation is repeated for a network of 4, 6, and 8 nodes, and the nodes' capacities are changed to be proportional to the weight per job; $\frac{c_v}{w_b}$ ranges between 1 and 5. For each simulation, the source and sink nodes are changed. The jobs' weights are fixed over all simulation runs, while the number of jobs ranges between 3 and 6. I evaluate the computation time required with and without power allocation and focus on the evaluation for the number of used time slots for different scenarios. I select a subset of our results when, first, I choose the structure of application graphs



(a) Loop. **(b)** Same to many. **(c)** Different to many.

(d) Linear. **(e)** Parallel.

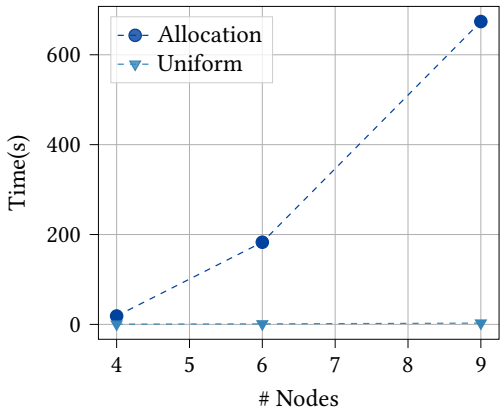
Figure 3.15: Examples of overlay graphs.

to be linear or parallel. Second, I increase the number of an application's jobs. Third, I increase the number of nodes. Fourth, I change $\frac{c_p}{w_p}$. I use the Gurobi solver, integrated with Pyomo modelling language [HWW11], to solve the optimisation models.

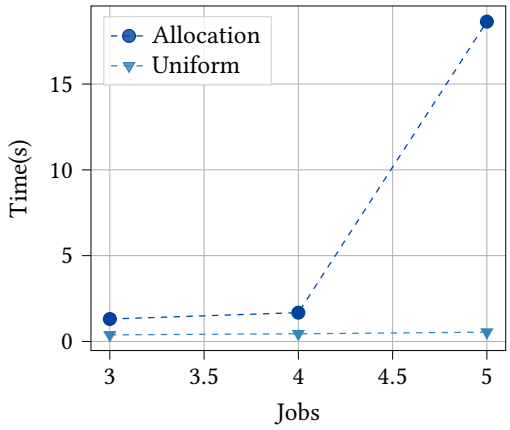
Execution Time

First, I set $\frac{c_v}{w_b}$ to 1 and fix the number of jobs to 5 in Figure 3.16 (a). We observe that increasing the number of nodes has a high impact on the computation time and, in addition, considering power allocation increases the computation time significantly due to the quadratic constraint. It ends up being on the order of hundreds of seconds (Figure 3.16 (a) compared to mere seconds for uniform transmit power.

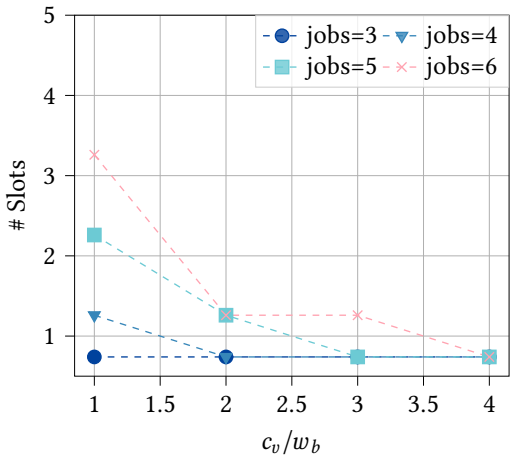
In contrast to the optimal solution in Section 3.1.1 and the used application graph, we observe that the proposed (MILP) solution here can solve the same problem for 4 and 6 nodes in 2 and 4 seconds, respectively, compared to 140 and 2564 in Section 3.1.1; a clear tribute to the improved problem formulation as an MILP rather than a quadratic problem.



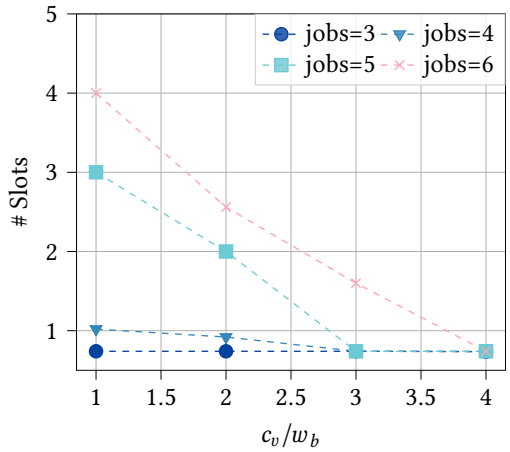
(a) #Jobs=5.



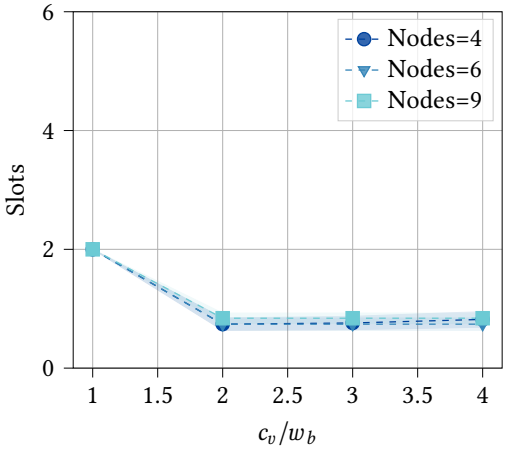
(b) #Nodes = 4.



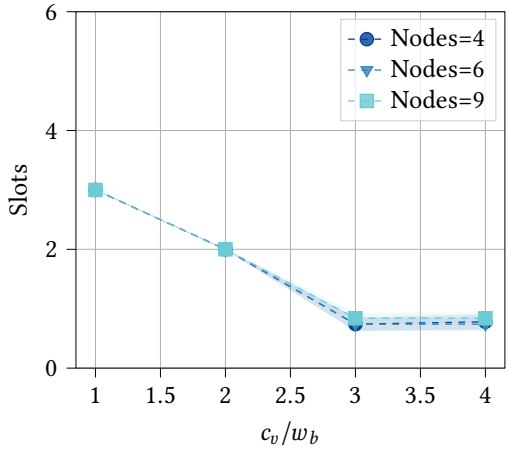
(c) Linear.



(d) Parallel.



(e) Uniform Power.



(f) Power Allocation.

Figure 3.16: Summary of results

Used Time Slots

I fix the number of nodes to 6 and observe that increasing the number of jobs (Figure 3.16 (c)) will increase the number of used time slots, due to the need for extra nodes, and consequently extra transmissions.

Additionally, we observe that for all scenarios, increasing the nodes' capacity (given by $\frac{c_v}{w_b}$) decreases the number of required time slots, until it reaches a point where the capacities are no longer a constraint. In other words, increasing the capacities of nodes allows more jobs to be placed per node, until we can place all the jobs on only two nodes; v_{src} and v_{sink}. Hence, the problem reduces to a mere routing problem where the objective is to find a route between v_{src} and v_{sink}.

Moreover, increasing the number of nodes (Figures 3.16 (e) and 3.16 (f)) has barely an impact on the number of required time slots: without (Figure 3.16 (e)) and with power allocation (Figure 3.16 (f)). This behaviour is due to the fact that additional nodes are neither used for placement nor for routing. These results are similar to those presented in Section 3.1.1. However, using power allocation can achieve lower number of time slots compared to using a uniform power, especially for scenarios where nodes have low capacities; when many nodes need to transmit data, power control can be used to allow simultaneous transmissions. But as the nodes' capacities increase, power control becomes insignificant since not many nodes are transmitting data (i.e., used for placement)

3.3 A Greedy Heuristic and Approximation Ratio

Here, I reapply the McCormick method to linearize the power allocation constraint in Section 3.2. Additionally, I simplify the heuristic of Section 3.1 and derive an upper bound of the output with respect to the optimal solution. Furthermore, I compare my solution to the proposed formulation in the literature when the interference is independent of the allocation. The following results are based on the outcome of [AK19a].

3.3.1 Linearize Quadratic Power Constraint

The following equations formalize the power allocation to control wireless interference in a *linear* formulation. Such linearisation offers tight boundaries that relaxes the problem, decreases the computation overhead of the quadratic constraint and still guarantees convexity. Additionally, it results in the same optimal solution.

$$\sum_{b_p \in B_P} s(v,v',b_p,t) \cdot \mathrm{SINR_{th}} \leq \frac{w(v,t)}{N_o + I(v,v',t)} \gamma_{v,v'}, \qquad \forall \{v,v'\} \in V \; \forall t \in T$$

$$\text{where } I(v,v',t) = \sum_{\substack{u \in V \\ u \neq v}} w(u,t) \cdot \gamma_{u,v'} \tag{3.26}$$

$$\frac{w(v,t) \cdot \gamma_{v,v'}}{\mathrm{SINR_{th}}} \geq \lambda(v,v',t) + N_o \sum_{b_p \in B_P} s(v,v',b_p,t), \qquad \forall \{v,v'\} \in V \; \forall t \in T \tag{3.27}$$

$$\lambda(v,v',t) \leq I(v,v',t), \qquad \forall \{v,v'\} \in V \; \forall t \in T \tag{3.28}$$

$$\lambda(v,v',t) \leq \sum_{b_p \in B_P} s(v,v',b_p,t) \cdot \gamma_{v,v'} |V|, \qquad \forall \{v,v'\} \in V \; \forall t \in T \tag{3.29}$$

$$\lambda(v,v',t) \geq I(v,v',t) - |V| \, (1 - \sum_{b_p \in B_P} s(v,v' b_p,t)), \qquad \forall \{v,v'\} \in V \; \forall t \in T \tag{3.30}$$

Recall that to guarantee a successful transmission, constraint (3.26) controls the interference $I(v,v',t)$ between nodes (v,v'). This also ensures that the SINR at node v' is bigger than or equal to $\mathrm{SINR_{th}}$. Nevertheless, this is a quadratic constraint. I linearize the constraint using McCormick's method for 0/1 quadratic problems [McC76]. First, I define the effect of interference on an active path as $\lambda(v,v',t) = I(v,v',t) \cdot \sum_{b_p \in B_P} s(v,v',b_p,t)$. Then, I define the conditional boundaries of the integer variable $\lambda(v,v',t)$ through constraints (3.28) – (3.30); it translates the quadratic constraint to a linear one, so that $\lambda(v,v',t) = I(v,v',t)$ if and only if there is an active transmission between nodes v,v' at time slot t, so that $\sum_{b_p \in B_P} s(v,v',b_p,t) = 1$.

3.3.2 Greedy Heuristic

Finding a zero-gap optimal solution for an optimization problem is time-consuming and, thus, heuristic solutions can be used for finding a fast sub-optimal solution, especially when the size of the problem is big.

I propose a constructive heuristic (i.e., starts with an empty solution and continuously extends the current solution) that can find a solution by following a sequence of pre-ordered constraints. It is a simplified version of the heuristic algorithm proposed in Section 3.1, which considers only 1-level lookahead and turns off the back-off feature.

Figure 3.17 shows how and when the power allocation decision is taken. First, a

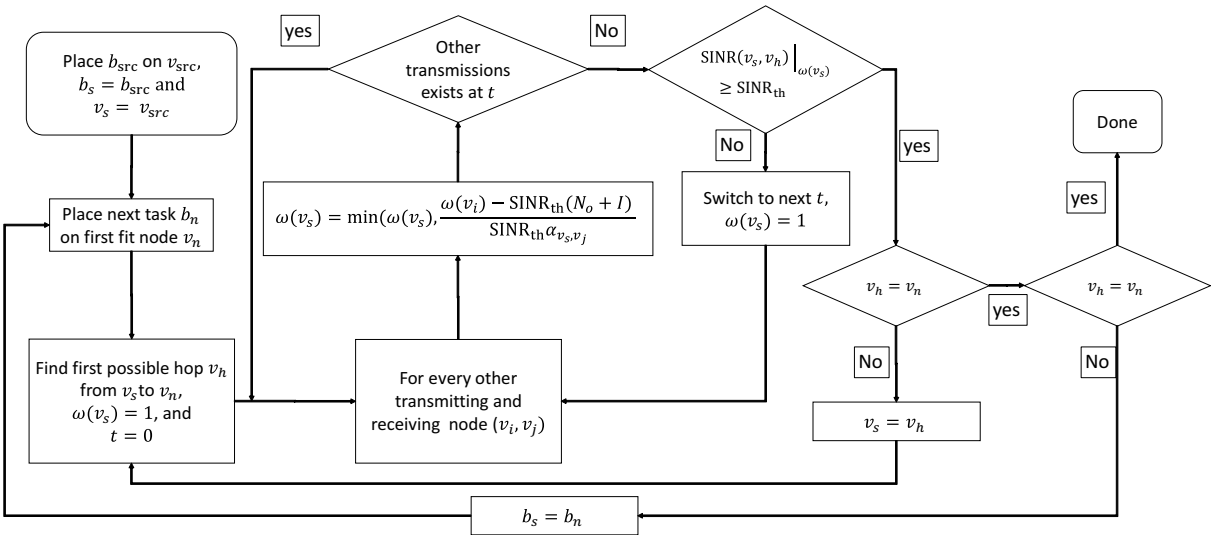
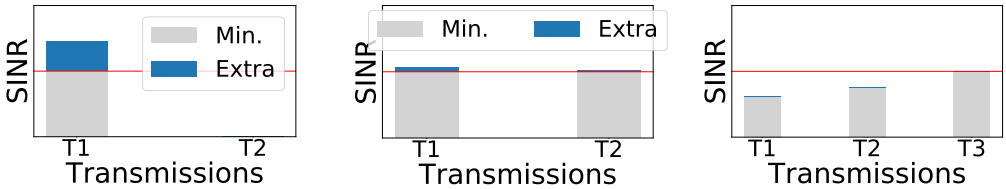


Figure 3.17: Visualization of the heuristic power allocation with VNE.

job is assigned to a node (while ensuring the capacity constraints are met), then a search process starts to find a route between two assigned jobs, using a breadth-first search (BFS) algorithm. Hence, the runtime complexity is $O(n^2)$. For a feasible solution, all routes need to have a minimum SINR_{th} across each of its paths.

To ensure this SINR threshold on a path, we look at candidate links in the topological order as given in the application graph. For this link's path, I choose the maximum possible transmission power to best protect against interference, while not interfering with other active transmissions. Then, for the same time slot, I continue looking for further paths to schedule, giving each path the largest transmission power that fulfills its own SINR requirement without violating the requirements of already scheduled paths. I continue that search per time slot until no more paths can be scheduled. Once it is no longer possible to find a power allocation for another path, the transmission takes place in a new time slot.

To have a better understanding of the power allocation procedures, I show a mock-up example in Figure 3.18, which represents the transmissions taking place at a specific time slot. At the beginning, only one transmission (T1) takes place in this time slot. In Figure 3.18 (a), the red line represent the SINR_{th}, while the blue region (above the red line) is the extra SINR that is above the threshold value. In Figure 3.18 (b), I show that it is possible to pack another transmission (T2) in the same time slot and observe that the blue region for T1 has decreased. Trying to pack another transmission in the same time slot (Figure 3.18 (c)) will decrease the SINR of the other two transmissions to be below the threshold. Therefore, only



(a) 1 Transmission (accepted).

(b) 2 Transmissions (accepted).

(c) 3 Transmissions (rejected).

Figure 3.18: Visualization heuristic decisions.

two transmissions can run in this time slot, and following transmissions must be packed in the next time slot. This approach of allocation is inspired by the inverse water filling algorithm [Pro01].

Adding power control to the SINR constraints reopens the question of how well the heuristic performs compared to the optimization approach. Therefore, I discuss in Section 3.3.3 the bounds of the heuristic when calculating the required number of time slots for data traversing.

3.3.3 Theoretical Analysis

Proposing VNE suboptimal solutions with bounds on their performance was proposed, but only for wired networks, as in [Ban+11; ERS16]. For the special case of a linear infrastructure and a linearly connected application graph, I derive a tight lower bound for the required number of time slots.

▶ **Theorem 3.1.** In an infrastructure with a linear topology where nodes are equidistantly separated by distance d and transmit in a free-space environment so that the SINR threshold is bounded by $\frac{1}{d^2 \cdot N_o} > \mathrm{SINR}_{\mathrm{th}} > \frac{1}{4N_o \cdot d^2}$, the optimal number of time slots δ_{opt} for a linear application graph is bounded from below by $\delta_{\mathrm{opt}} > \frac{(M_{\mathrm{tot}-1})(2 \cdot \mathrm{SINR}_{\mathrm{th}}-1)}{3 \cdot \mathrm{SINR}_{\mathrm{th}}-1}$, where M_{tot} is the number of nodes used for transmissions and running the jobs.

◀

Proof. The infrastructure network consists of $|V|$ nodes that are linearly separated by distance d. After distributing the jobs in the infrastructure network, only $V_{\mathrm{tot}} \subset V$ nodes are used for running the jobs and forwarding the data. The number of these nodes is given by $M_{\mathrm{tot}} = |V_{\mathrm{tot}}|$. A set with the maximum number of nodes that can transmit simultaneously is given by V_{max}, where $|V_{\mathrm{max}}| = M_{\mathrm{max}}$.

We want to find a lower bound on the number of time slots in which one source-to-sink communication can be completed. The idea is to find the largest number of

nodes that can communicate simultaneously in one time slot ($M_{\max}$) and divide the total number of used nodes M_{tot} by this number.

The worst interference that can be generated by these $M_{\max}$ nodes is at the middle receiving node for $V_{\max}$, such that the maximum inter-distance between two consecutive transmitting nodes is $2d$ and the minimum is d.

To better visualize this interference problem, Figure 3.19 depicts 5 transmitting nodes ($V_{\max}$), each sending to its direct neighbour. Since $\text{SINR}_{\text{th}} > \frac{1}{4N_o \cdot d^2}$, only transmissions to the immediate neighbours are possible, but a node cannot be sending and receiving at the same time (duplex constraint). Then, we select a (shaded) node, which is the middle receiving node for $V_{\max}$ and suffers from the highest interference (4 interferers given as I_A, I_B, I_C and I_D). Accordingly, the lowest possible SINR is given by

$$\text{SINR}_{\text{th}} \leq \text{SINR} = \frac{\text{Received Power}}{N_o + \sum_{n=1}^{M_{\max}-1} I_{v_n}} \tag{3.31}$$

$$\text{SINR}_{\text{th}} \leq \text{SINR} = \frac{1}{d^2} \frac{1}{N_o + \frac{1}{d^2} \sum_{n=1}^{M_{\max}-1} \frac{1}{n^2}}, \tag{3.32}$$

Consequently, we will have $M_{\max} - 1$ interfering nodes at this time slot. Using the integral test [W B12], we recall that

$$\sum_{n=1}^{M_{\max}-1} \frac{1}{n^2} < 2 - \frac{1}{M_{\max} - 1} \tag{3.33}$$

Thus, substituting (3.33) into (3.32), we assume that the following equation still holds

$$\text{SINR}_{\text{th}} < \frac{1}{N_o \cdot d^2 + 2 - \frac{1}{M_{\max}-1}} \tag{3.34}$$

Since typically $N_o \cdot d^2 << 1$, we can rewrite (3.34) as

$$M_{\max} < 1 + \frac{\text{SINR}_{\text{th}}}{2 \cdot \text{SINR}_{\text{th}} - 1} \tag{3.35}$$

Note that SINR_{th} is bounded by $\frac{1}{N_o \cdot d^2}$ as per (3.32). Therefore, in case of relatively high values of SINR_{th}, we cannot ignore $N_o \cdot d^2$

Given that nodes are operating in a half-duplex mode and a maximum $M_{\max}$

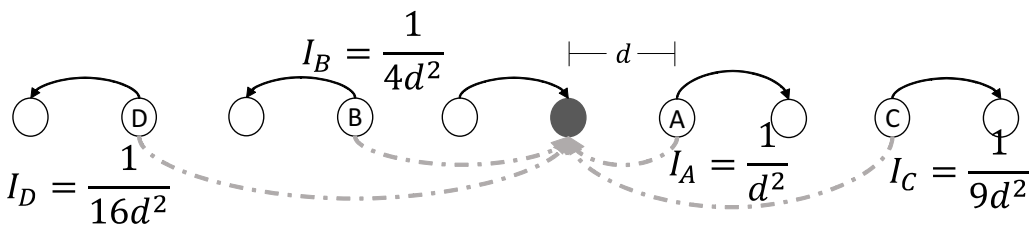


Figure 3.19: Wireless nodes placed in a linear topology with equidistant d separation. I_x represents the interference by node x on the selected (shaded) receiving node, while w_x is the transmitted power.

nodes are sending at a time slot, then δ_{opt} will be at least

$$\delta_{\text{opt}} \geq \frac{M_{\text{tot}} - 1}{M_{\text{max}}} \tag{3.36}$$

As the number of M_{max} tends to be large enough [W B12], we can rewrite (3.33) as Riemann zeta function

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} \tag{3.37}$$

Obviously the difference between (3.33) and (3.37) is less than 1, while δ_{opt} is an integer. Therefore, our assumption in Eq. (3.34) holds. Then, we substitute (3.35) in (3.36)

$$\delta_{\text{opt}} > \frac{\mathrm{M}_{\text{tot}} - 1}{1 + \frac{\mathrm{SINR}_{\text{th}}}{2 \cdot \mathrm{SINR}_{\text{th}} - 1}}$$

$$\delta_{\text{opt}} > \frac{\mathrm{M}_{\text{tot}} - 1}{3 \cdot \mathrm{SINR}_{\text{th}} - 1} \cdot (2 \cdot \mathrm{SINR}_{\text{th}} - 1) \tag{3.38}$$

■

▶ **Corollary 3.2.** Given the previous conditions but for a generic wireless propagation where the path-loss exponent $\alpha \geq 2$ and the minimum allocated transmit power is w_{min}, the relation between δ_{opt} and M_{tot} is given by

$$\delta_{\text{opt}} \geq \frac{\mathrm{M}_{\text{tot}} - 1}{1 + \frac{\mathrm{SINR}_{\text{th}}}{(2 + N_o \cdot d^{\alpha}) \cdot \mathrm{SINR}_{\text{th}} - w_{\text{min}}}} \tag{3.39}$$

◀

Without loss of generality, we assume that all services have the same processing weights w_b and all nodes have the same processing capacity c_v. Then

$$\mathrm{M}_{\mathrm{tot}} \geq \frac{|B| \cdot w_b}{c_v} \tag{3.40}$$

▶ **Corollary 3.3.** The value δ_{opt} is bounded by the available and required resources so that

$$\delta_{\mathrm{opt}} \cdot c_v \geq \frac{|B| \cdot w_b - c_v}{1 + \frac{\mathrm{SINR}_{\mathrm{th}}}{(2+N_o \cdot d^{\alpha}) \cdot \mathrm{SINR}_{\mathrm{th}} - w_{\min}}} \tag{3.41}$$

◀

▶ **Theorem 3.4.** The proposed heuristic has an approximation $\frac{4}{c_v^2} \frac{|B|^2 \cdot w_b^2 - c_v^2}{1+\frac{\mathrm{SINR}_{\mathrm{th}}}{(2+N_o \cdot d^{\alpha}) \cdot \mathrm{SINR}_{\mathrm{th}} - w_{\min}}}$ for the optimization problem ◀

Proof. The total number of required time slots relies on two dimensions; the number of nodes used for mapping and the number of time slots used for transmissions between nodes. Thus, the product of both upper bounds results in the overall upper bound.

Relying on the topological order for assigning jobs to nodes yields a first-fit behavior for the bin-backing problem [JG85]. Thus, the total number of used nodes is bounded from above by

$$2\frac{|B| \cdot w_b}{c_v} = 2(\#\mathrm{optNodes}) \tag{3.42}$$

Similarly, the number of transmissions rely on the first-fit transmission. By replacing Eq. (3.40) with (3.42), we can rewrite Eq. (3.41) so that the number transmissions is upper-bounded by

$$\delta_{\mathrm{opt}} \leq \frac{4}{c_v^2} \frac{|B|^2 \cdot w_b^2 - c_v^2}{1 + \frac{\mathrm{SINR}_{\mathrm{th}}}{(2+N_o \cdot d^{\alpha}) \cdot \mathrm{SINR}_{\mathrm{th}} - w_{\min}}} \tag{3.43}$$

■

We observe from Eq. (3.43) that as the ratio $\frac{w_b}{c_v}$ decreases, the required number of time slots decreases. But we need to bear in mind that in Eq. (3.40), $\mathrm{M}_{\mathrm{tot}}$ is at least equal to 2 since we predefine v_{src} and v_{sink}. Accordingly, in our special case,

the upper bound in Eq. (3.43) is rewritten as:

$$\delta_{\text{opt}} \leq \max(|V| - 1, \frac{4}{c_v^2} \frac{|B|^2 \cdot w_b^2 - c_v^2}{1 + \frac{\text{SINR}_{\text{th}}}{(2+N_o \cdot d^\alpha) \cdot \text{SINR}_{\text{th}} - w_{\min}}}) \tag{3.44}$$

3.3.4 Evaluating the Greedy Heuristic and Estimating Symbol Error Rate

During the simulations, I assume that the applications have a linear topology. To find an optimal solution, I used a Gurobi solver [Gur18] with Pyomo [HWW11] interface. I show two types of simulations to demonstrate the wireless VNE solution. In the beginning, I compare between the optimal (with zero optimality gap) and heuristic solutions to evaluate the latter's sub-optimality for the required number of time slots. Then, I compare the proposed interference-aware formulation with fixed-data rate assumptions (in related work) for wireless VNE in terms of symbol error rate.

Required Time Slots

The simulations are divided into two scenes. **First**, I distribute the nodes in linear (hallway-style) scene to validate the output from Section 3.3.3. **Second**, I consider a more generic (lattice-style) distribution of nodes and observe the changes.

The focus is mainly on the relation between the number of used time slots as a metric and the number of nodes, number of jobs, and ratio between the node's capacity and required computation ($\frac{c_v}{w_b}$) as parameters. When changing the number of nodes, I fix the number of jobs to 4. When changing the number of jobs, I fix the number of nodes to 6. In both cases, I set the ratio ($\frac{c_v}{w_p}$) to 1. When changing the ratio $\frac{c_v}{w_b}$, I fix the number of nodes to 6 and the number of jobs to 4.

Figure 3.20 shows the number of time slots required by the heuristic algorithm compared to the optimal solution as well as the lower and upper bounds formulas.

On one hand, I observe that the lower bounds and optimal solutions are close; the difference is less than 1 time slot. On the other hand, I observe a large gap between the heuristic's solutions and the upper bound, when changing the number of jobs or nodes in Figure 3.20 (b) and Figure 3.20 (a)), respectively. This is due to the fact that the upper bounds assume the worst scenario where transmissions between two jobs go through all nodes, which did not happen in these simulations. Nevertheless, increasing $\frac{c_v}{w_p}$ (Figure 3.20 (c)) tightens the upper bound to overlap

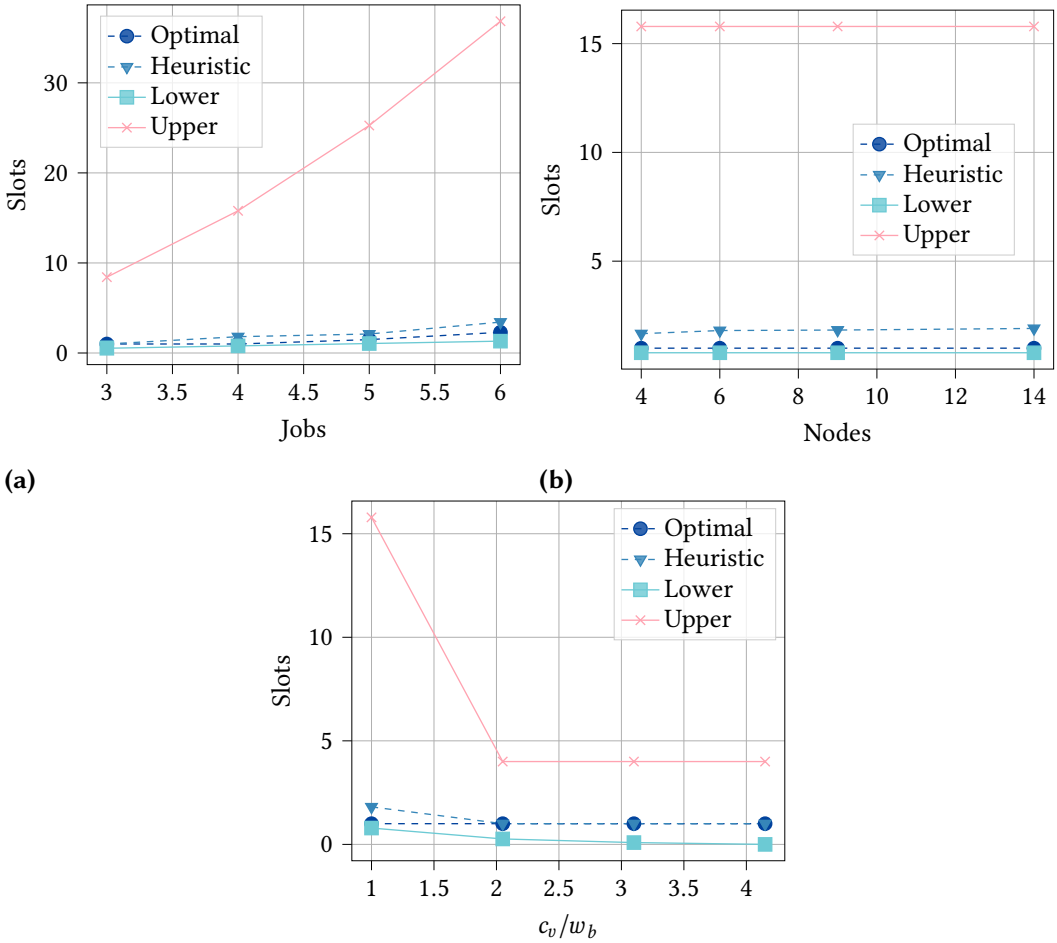


Figure 3.20: Results for nodes with linear distribution.

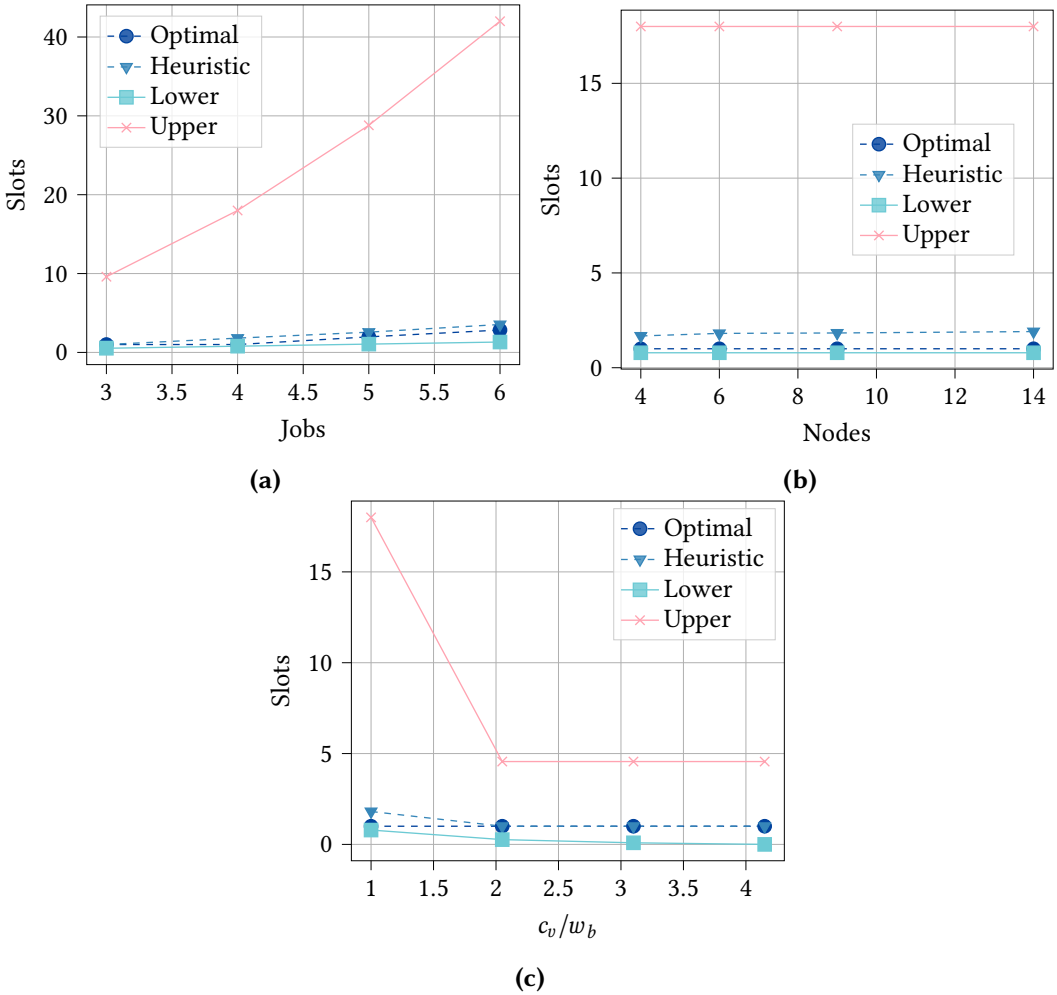


Figure 3.21: Results for nodes with lattice distribution.

with the heuristic's results. This is explained by Equation (3.44), increasing $\frac{c_v}{w_p}$ will relax the problem to be multi-hop routing upper bound; when $\frac{c_v}{w_p}$ is very high, all jobs can run on one node and I may rewrite Equation (3.44) as $\delta_{\text{opt}} \leq |V| - 1$.

Additionally, I observe, in Figure 3.20 (b), that the upper and lower bounds are independent of the number of available nodes as long as $\frac{c_v}{w_p}$ is constant and the number of jobs does not change (see Equations (3.41) and (3.44)). Nevertheless, the heuristic performance shows a slight correlation between the number of time slots and the number of nodes, due to following the topological order during the assignment process. Such behaviour is expected as it is inherited from the first-fit bin-backing solution [JG85].

Figure 3.21 shows the output for nodes distributed on a lattice when changing the number of nodes (Figure 3.21 (b)), number of jobs (Figure 3.21 (a)) and the ratio $\frac{w_p}{c_v}$ (Figure 3.21 (c)). A straightforward derivation for the upper bound in a lattice distribution is $\sqrt{2}\times$ upper bound δ.

Considering the nodes being distributed on a lattice leads to longer computation time for the optimizer (due to more possible paths), especially for a high number of jobs. Therefore, I limit the investigation to five jobs. I observe, in Figure 3.21 (a), that the gap does not change much compared to the linear distribution (Figure 3.20 (a)). This shows that the heuristic's performance is independent of the number of jobs as long as the number of nodes does not change.

Again, in Figure 3.20 (b) I observe that increasing the number of nodes increases the number of time slots used by the heuristic. Nevertheless, we do not get the exact results as in 3.1 because in the lattice distribution new routes are found through the diagonal paths.

In Figures 3.21 (a) and 3.21 (c) (as well as Figures 3.20 (a) and 3.20 (c)), the graphs are reciprocal, which means that decreasing the number of jobs is equivalent to increasing $\frac{c_v}{w_b}$. Because increasing the ratio $\frac{c_v}{w_b}$ allows more jobs to be running per node, this is equivalent to decreasing the number of jobs that need to be distributed. Therefore, we observe such reciprocity.

I need to highlight that the computation time required to solve the problem with zero gap optimality was tens of minutes (40 minutes in extreme cases) while the heuristic solution required in most cases 10s of seconds.

Symbol Error Rate

Considering the broadcast property in the proposed model introduces a lot of complexities. In order to demonstrate the importance of considering SINR in the formulation, I compare the results of the proposed model to other models that do

not manage interference explicitly [Abd+16b]. By ignoring the interference, only paths that have an SNR (note the absence of "I", so no interference considered) larger than $\mathrm{SINR}_{\mathrm{th}}$ have their nodes connected.

For simplicity, I assume that no channel coding is used and all transmissions have the same modulation (16-QAM). Accordingly, I use a high $\mathrm{SINR}_{\mathrm{th}}$ (17 dB) and evaluate the symbol error rate as [FS94]

$$P_s(e) = \frac{3}{2}\mathrm{erfc}\left(\sqrt{\frac{\mathrm{SINR(e)}}{10}}\right), \quad \forall e \subset V \times V \tag{3.45}$$

where e represents the path used for transmission between two nodes. Consequently, the worst-case symbol error when having at least one error at any transmission is given by

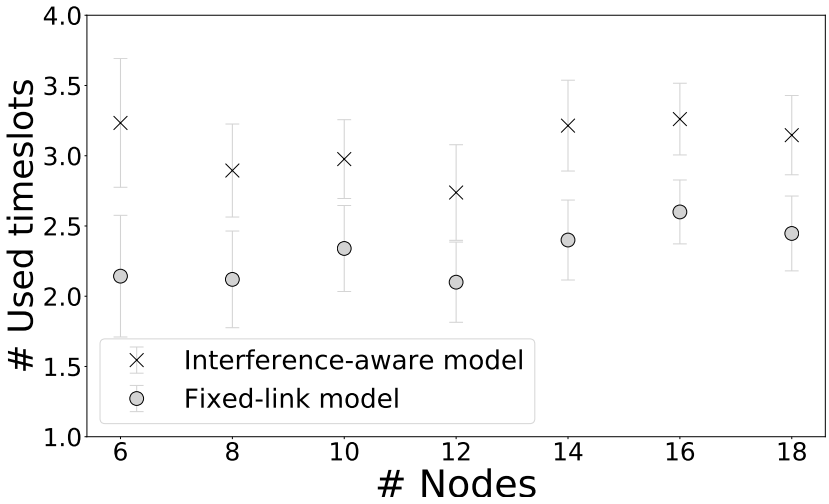
$$P_s = 1 - \prod_{e \subset V \times V} (1 - P_s(e)) \tag{3.46}$$

where P_s represents the worst-case symbol error rate for the multi-hop embedding solution.

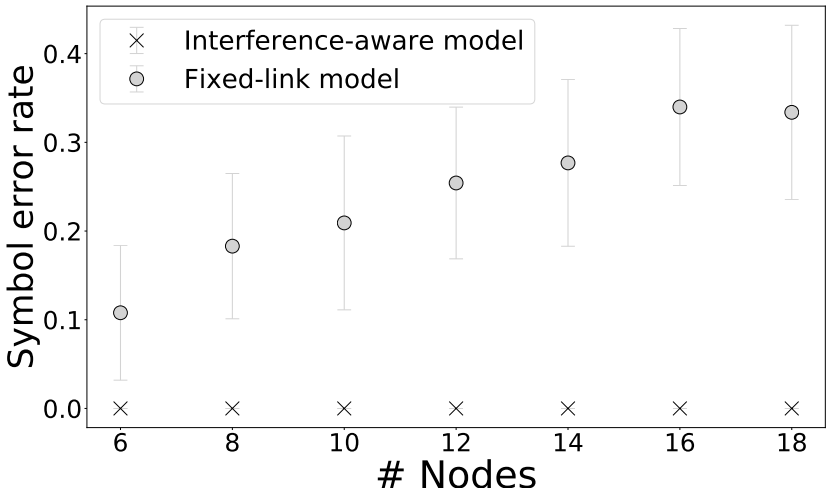
In the evaluation, I use, on one hand, the term *Fixed-link model* to represent the output when paths are assumed to have a fixed data rate, regardless the interference from other nodes. On the other hand, I use *Interference-aware model* to represent the output of the proposed model, which controls the interference over different time slots. In other words, each path will have a different data rate at each time slot. The data rate depends on the SINR, which is controlled by the transmission power of sending nodes.

I show the results with 95 % confidence intervals for the time slots and symbol error rate in Figure 3.22. As expected, the Fixed-link model required fewer time slots than the Interference-aware model (Figure 3.22 (a)), because the former assumes that the data rate is fixed between the nodes and thus, it requires the time slots only for multi-hop transmissions, while the latter may not send at a time slot and needs extra time slots with lower interference.

The additionally required time slots range between 1 to 2 time slots. Moreover, when considering overall symbol error rate (Figure 3.22 (b)), the interference-aware model outperformed the fixed-link one. I observe that the proposed approach had almost negligible symbol error rate, while fixing the paths can have a severe symbol error rate that ranges from 3 % to 40 %.



(a) Average used number of time slots



(b) Upper-bound for symbol error rate

Figure 3.22: Comparison between fixed-link and interference-aware models

3.4 Summary

In this chapter, the focus was on formulating wireless VNE and showing the improvement compared to related work. Different optimization problems were proposed, where each had different number of variables and constraints; which formulation

to use highly depends on the solver. Meanwhile, finding exact solutions for optimization problem takes long time. Hence, I proposed a greedy heuristic that can find a feasible solution relatively fast and I analysed the gap between the heuristic and optimal solutions: theoretically via derivation and empirically via simulations.

Furthermore, I have shown that considering the interference as a variable in the VNE formulation is essential for handling wireless transmission errors. In the presented scenarios, the improvements in symbol error rate were between up to 40%.

4 Meta-Heuristics for Resource Allocation

In the previous chapter, I provided an optimization problem for wireless virtual embedding as well as a greedy heuristic with its lower and upper bounds. Yet the provided heuristic is problem-specific and cannot be applied to other problems. In other words, changing or adding new variables to the problem (e.g., battery life or motion control) will not be easy to integrate into the heuristic. Additionally, greedy heuristics in general can get stuck in a local optimum. Accordingly, I apply two generic meta-heuristic solutions: Genetic Algorithm (GA) and RL), which can be easily modified to adapt to any changes in the problem. Other meta-heuristics are also applicable, yet I choose those two, since their formulation were for me pretty much straight forward.

Genetic algorithms belong to a popular family of population-based stochastic algorithms. These algorithms are also known to be inter-life [Kar20] algorithms, where an individual interacts with the others to define their ranking and their chance of survival. In fact, such algorithms have been applied to similar VNE problems. The authors in [Faj+11] used an ant-colony algorithm to maximize the number of applications running in parallel inside a wired network. In such an approach, ants explore different solutions over multiple iterations and at the end, the best solution over all iterations is selected. Similarly, the work in [Zha+13] uses particle swarm for the same objective; solutions are encoded as particles that keep moving to improve their positions based on their current experience and the group experience. Moreover, the work in [BMH19; MKA21] use genetic algorithms for the same objective. In Section 4.1, I use genetic algorithm to solve the wireless VNE problem.

In contrast to genetic algorithms, RL belongs to intra-life algorithms [SC18], meaning that an individual learns through its existence, e.g., it learns how to communicate and walk. Essentially, genetic algorithms can be applied to any problem, while RL is used to find optimal strategies that maximize the output. This means that RL is suited for problems, where a sequence of actions are required. Nevertheless, RL has not been well investigated for wireless VNE in particular. Most of the RL work, related to wireless networks, focused on building paths between the nodes [Guo+19], without considering the placement aspect. The work in [Fu+20] reconfigured existing placement solutions under changing resource requirements, but in our problem, no initial placement exists nor do resource

requirements change. The authors in [Wan+19] considered the placement aspect in mobile edge computing (MEC); unlike the presented case, they consider the placement of separate functions rather than a graph of functions.

To sum up, existing RL solutions do not address the same facets of the addressed problem here and cannot be directly applied to the given input assumptions of wireless networks. In Section 4.2, I integrate RL with a heuristic solution for a complete VNE solution. Then, I compare the RL framework to the optimization problem.

4.1 Genetic Algorithm

The results in this section are based on the work in [AHK19]. A GA encodes a solution as a chromosome [Mit96]. In the beginning, a set of chromosomes, called population, is generated randomly. Then, the GA starts an iteration process, where over each iteration the population evolves to a new one by using crossover, mutation, and selection. The final solution is the fittest chromosome from the last population. In this section, I show how I match the GA to the wireless VNE problem.

4.1.1 Chromosome

A chromosome represents a joint solution for three problems: placement, routing and scheduling. Since the initial population consists of randomly generated chromosomes, a chromosome may represent an invalid solution as well. I do not ignore these chromosomes for a reasonable diversity among the initial population, lest we converge prematurely. In this context, a chromosome consists of three genes:

1. placement: an array to represent the node running each job

2. routing: a 2D matrix that represents a random cost of every wireless edge to calculate routing table

3. scheduling: a 2D matrix that represents a priority to every wireless edge to assign a time slot

The process of evolving a population is described as follows:

Placement

The crossover of two chromosomes swaps the jobs running on a node. In a mutation, one randomly chosen job will be assigned to a randomly chosen node. The randomness, for both crossover and mutation, increases the diversity for the new generation; however, this may also lead to invalid chromosomes, which can be alleviated by the selection for the new population (see Section 4.1.3).

Routing

Each direct path between two nodes is assigned a random cost $\in [0, 1]$, except for those with low SNR (i.e. $< \text{SINR}_{\text{th}}$); they are assigned an infinity cost. During crossover, the path's cost for the new chromosome is set to the mean of parent paths, while in the mutation, a few links are selected randomly and get new random costs, except for paths with infinity costs.

These costs are used to compute a routing table using Dijkstra's algorithm. Next, the routes between nodes running jobs $b, b' \in L$ are computed via *getTransmissions* (Algorithm 2)), to determine transmissions (sending and receiving nodes), as well as the transmitted data (i.e., from which job). So far, the transmissions have not been yet scheduled.

Scheduling

In the beginning, each transmission is assigned a random priority between 0 and 1. The crossover of two chromosomes assigns the average priority to the new transmission and the mutation selects a new random priority. These priorities represent, which transmission should be scheduled first.

The output of *getTransmissions* is scheduled in a time slots $s \in S$ via *RandomScheduler* (Algorithm 4), as follows. Two priority queues, Q_s and Q'_s are used to sort the transmissions, where Q_s represents transmissions that will try to be scheduled in slot s and Q'_s represents transmissions that failed to be scheduled in time slot s.

Whether a transmission can be scheduled in a time slot or not, this is determined by the algorithm *placeable* (Algorithm 3). It checks if $\text{SINR} \geq \text{SINR}_{\text{th}}$, checks for duplex constraints (a node cannot transmit and receive at the same time slot), and allows multi-cast scheduling so that a node can send the same data to multiple nodes in the same time slot.

For all transmissions in Q'_s that failed to be scheduled in time slot s, they try again in the next time slot $(s + 1)$ so that $Q_{s+1} = Q'_s$. The algorithm *RandomScheduler* terminates if Q_s and Q'_s are empty (successful termination) or if no single transmis-

sion can be scheduled at two consecutive time slots; $Q_{s+1} = Q_s$ and no transmission exits at time slot s (no feasible solution is found).

After handling every link of the application graph, the algorithm returns S and terminates.

Algorithm 2: getTransmissions

Input: Application graph G_O, Chromosome c

```
1  T ← empty list of transmissions
2  foreach link (b, b′) in G_O do
3      t ← new transmission
4      t.job ← b
5      t.sender ← placement(b)
6      t.receiver ← c.routingtable(t.source,placement(b′))
7      T ← T ∪ {t}
8      while t.receiver ≠ c.placement(b′) do
9          prev ← t
10         t ← new transmission
11         t.job ← b
12         t.sender ← prev.receiver
13         t.receiver ← c.routingtable(t.source,c.placement(b′))
14         T ← T ∪ {t}
15 return T
```

4.1.2 Fitness Function

A high fitness value represents a good solution. The objective is to minimize the number of time slots, given by *RandomScheduler*. Hence, I design the fitness to be inversely proportional to the number of time slots used per chromosome. Nevertheless, some chromosomes, having very low number of time slots, may represent infeasible solutions.

Accordingly, I prefer chromosomes that do not violate the constraints, by adding a binary variable $\alpha(c)$ to the fitness function. $\alpha(c) = 1$ if the chromosome c satisfies all the constraints and 0 otherwise. Also, if the scheduler failed for a chromosome, its fitness is set to 0.

$$\text{fitness(c)} = \frac{1}{|S| + 1} + \alpha(\text{c}) \tag{4.1}$$

Algorithm 3: placeable

Input: Transmission t, slot s

```
foreach t' ∈ s do
    I ← interference at t'.receiver
    SINR ← γ_{t.sender,t.receiver} · P / (I + N_0)
    if SINR_th < SINR then
        return false
if t.receiver is receiving or sending in s then
    return false
if t.sender sending data of t.job in s then
    return true
if t.sender is receiving or sending in s then
    return false
return true
```

4.1.3 New Generations and Selection

As a result of mutation and crossover within a population, new chromosomes are generated that create a new generation. Out of this generation, I select a subset of chromosomes to be the new population. Each new population receives the chromosomes with the highest 10 % fitness from the previous population, to guarantee that the so-far best solutions are not lost in the course of the evolution. Then, each other chromosome is selected with a probability equal to their relative fitness weight:

$$\mathbb{P}(\text{c is selected}) = \frac{\text{fitness}(\text{c})}{\sum_{\text{c}' \in \text{population}} \text{fitness}(\text{c}')} \tag{4.2}$$

This gives a chance for low-fitness chromosomes to be selected in the new population, which increases the diversity and may later find better chromosomes by means of crossover and mutation.

4.1.4 Evaluation of Genetic Algorithms

Different choices of the population size (p), crossover rates (r_c) and the mutation rates (r_m) are investigated. After that, I evaluate the GA when changing the number of network nodes as well as the number of jobs.

Algorithm 4: RandomScheduler

Input: Infrastructure graph G_I, Application graph G_O, Chromosome c

```
1  S ← empty schedule
2  Q ← empty priority-queue of transmissions
3  Q′ ← empty priority-queue of Transmissions
4  i ← 1
5  Q ← getTransmissions(G_O, c)
6  while Q ≠ {} do
7      t ← Q.pop()
8      b ← false
9      if placeable(t,s_i) then
10         S.add(s, t)
11         b ← true
12     else
13         Q′.put(t)
14     if Q = {} then
15         if b = false then
16             return null
17         i ← i + 1
18         Q ← Q′
19         Q′ ← empty priority-queue of transmissions
20 return S
```

I consider power-plugged wireless devices as seen in home appliances and factory setups; the sensor nodes have fixed positions. The nodes of the infrastructure graph are distributed uniformly at random in an area of 10 by 10 meters. The attenuation between two nodes v and v' is set to $\gamma_{v,v'} = K/d^2_{v,v'}$, where $d_{v,v'}$ is the distance between the nodes and K is a power constant.

When varying the number of nodes and jobs, two different scenarios are considered. First, I change the number of nodes while the application graph is fixed to represent an algorithm from the field of acoustic signal processing [Sch+17a] (Figure 3.4), which consists of hybrid topologies (linear, parallel, and loop). Second, I fix the number of nodes to 15 and vary the number of jobs, where the links between the jobs are generated at random, so that each pair of jobs has a probability of being connected by a link. Unless stated otherwise, I see the priority to $\frac{1}{8}$.

For each analysis, there are 50 independent realizations of the application and infrastructure graphs. I set a maximum of 2000 generations for each scenario. Although 2000 generations seem to be an overkill – all scenarios converges not later than the 1000th generation (Figure 4.1, 4.5) – but I want to avoid constraining the final result of the algorithm by setting the generation limit too low.

Population Size, Crossover Rate and Mutation Rate

I evaluate the performance of GA with different configurations for the population ($p \in \{50, 100, 200\}$), crossover ($r_c \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$), and mutation ($r_m \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$), where I experiment with low, middle and high probabilities. For the sake of clear visibility in the figures, I select the most significant configurations to show the dependency for the number of time slots (Figure 4.1) and the runtime (Figure 4.2).

In Figure 4.1, it is clear that the GA converges faster for bigger populations since it contains more chromosomes and has better chances of convergence, but of course this comes at the expense of computation time (Figure 4.2). When sweeping over the mutation probabilities, I observe that convergence speeds up with increasing probabilities, (Figure 4.1) at only a slight increase in computation time (Figure 4.2). The crossover probabilities have barely any impact on the convergence rate. At the end, all simulations converge to the same number of time slots as found by Section 3.1.

For further simulations, $p = 100$, $r_c = 1.0$ and $r_m = 0.25$ are selected. Using these values, the best placement, routing and scheduling is on average already computed in generation 144, and the median runtime is 5.89 seconds; the median is selected to prune outliers due to runtime glitches or similar problems.

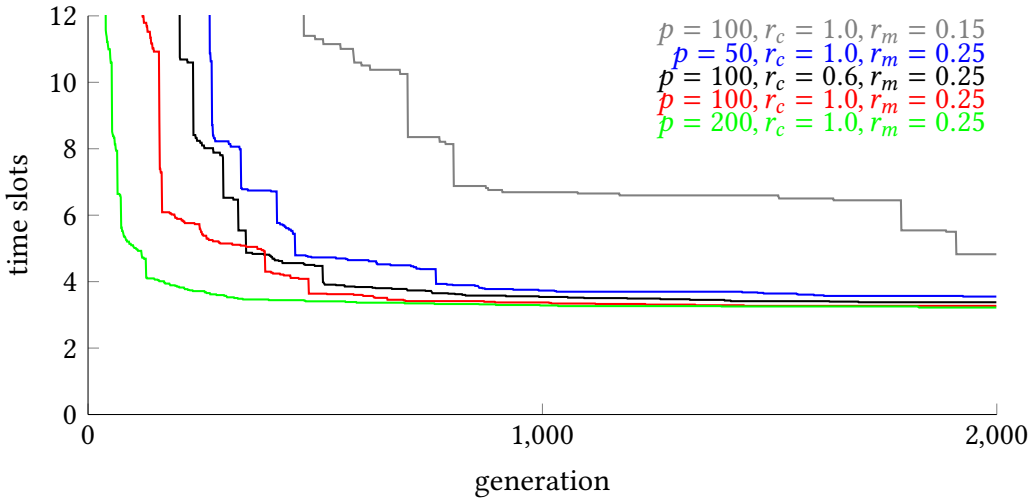


Figure 4.1: Average number of time slots depending on the generation of different combinations of population size p, crossover rate r_c and mutation rate r_m

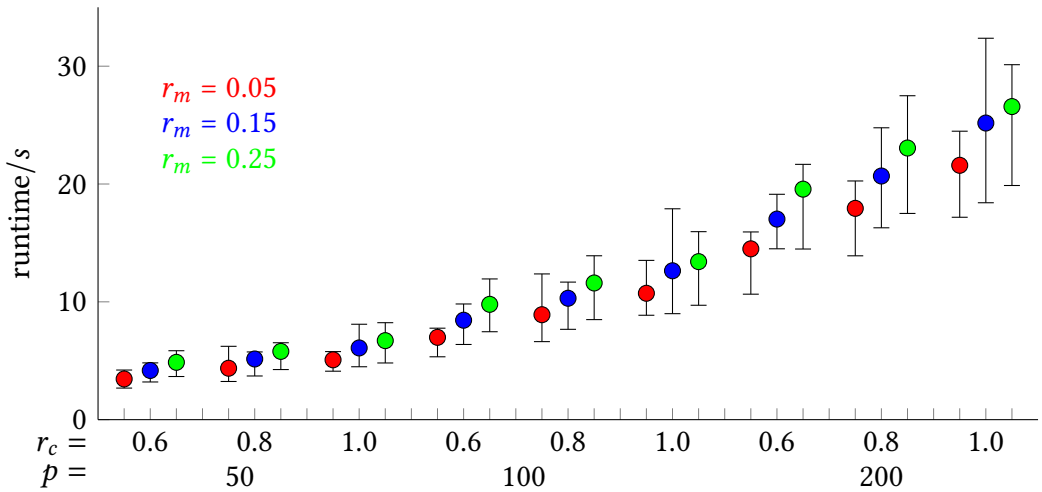


Figure 4.2: Median runtime of different population sizes, crossover rates and mutation rates with a 95% confidence interval

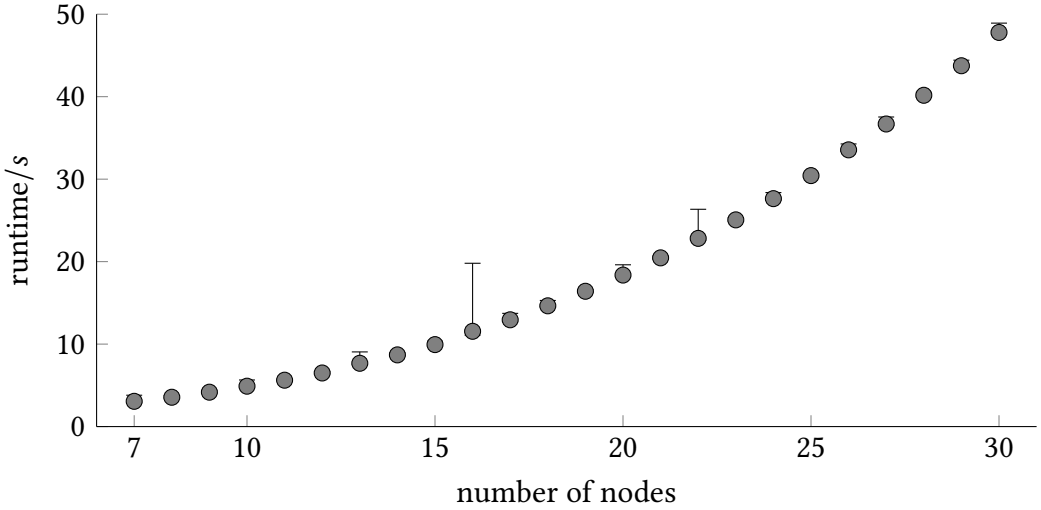


Figure 4.3: Median runtime of different node numbers with 95% confidence interval

Runtime

To consider the bound on largest possible runtime of the GA, I assume a full mesh application graph, so that the runtime is given by $O\big(g \cdot p \cdot \big(|V|^3 \cdot |B|^6 + |V|^4\big)\big)$, where p is the population size and g the number of generations.

Using simulations, the polynomial dependency is evaluated in time units (/s) for the number of nodes (Figure 4.3) and for the number of jobs (Figure 4.4), where I show the median runtime as a function of the number of nodes and blocks with 95% confidence intervals. Again, the median is selected instead of the average, to ignore the outliers impact. Note that the application graph topology is no longer a full mesh graph, jobs are rather connected with a probability $\frac{1}{8}$. In comparison, the runtime increases with a higher gradient with respect to the number of jobs than to the number of nodes.

For Figure 4.4, I generate different topologies at random for the application, where the probability of having a link between two jobs is $\frac{1}{4}$ (red) or $\frac{1}{8}$ (blue). I observe that the runtime increases not only with the number of jobs, but also with the number of links in the application graph.

Impact of application graph topology

Here, I investigate the impact of application topologies on the resulting end-to-end delay. In this experiment, the number of nodes is fixed (to be 15) and I fix the GA configuration ($p = 100, r_c = 1.0, r_m = 0.25$). I change, however, the application

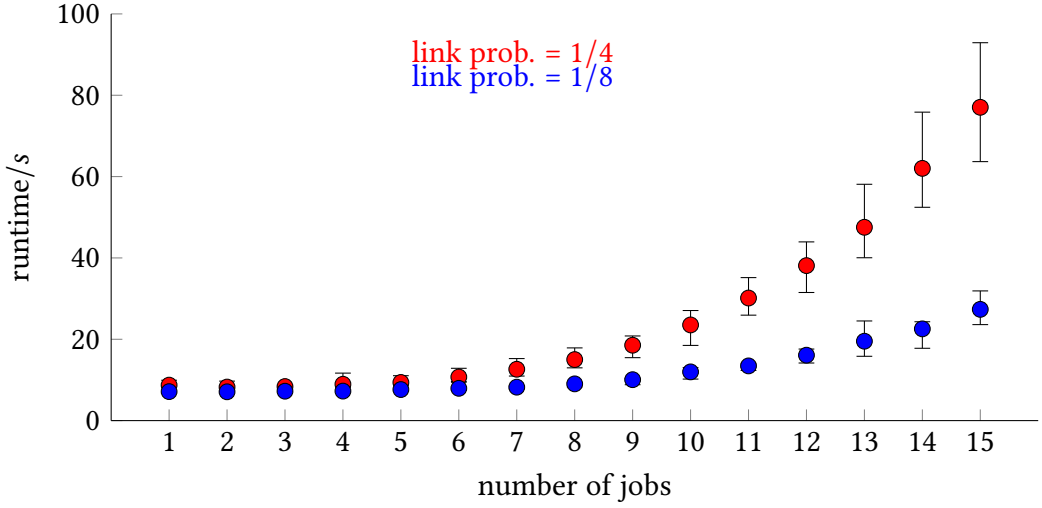


Figure 4.4: Median runtime and 95% confidence interval of random generated application graphs with different number of jobs and link probabilities

graphs to have linear (Figure 3.15 (d)) and parallel (Figure 3.15 (e)) topologies. Note that the objective here is not to evaluate GA, but to find a relation between the application graph topology and the number of used time slots.

I observe, in Figure 4.5, that even when the linear topology has more jobs than the parallel one, the former needs lower number of time slots, because the parallel topology has more links in the application graph. This highlights the importance of considering the links in the application graphs rather than considering only the number of jobs that need to be distributed as in [Din18], especially when considering latency requirements. I highlight that the output from GA was the same as from the optimization model (Section 3.1), using the setup in Section 3.1.4.

4.2 Reinforcement Learning for Placement

In this section, I propose a solution for the VNE problem using a modular implementation. In other words, the solution consists of 3 modules, each solving a different problem: RL for placement, shortest-path-first for routing and mixed-integer linear programming for scheduling. The results of this section are based on the work in [AK20]

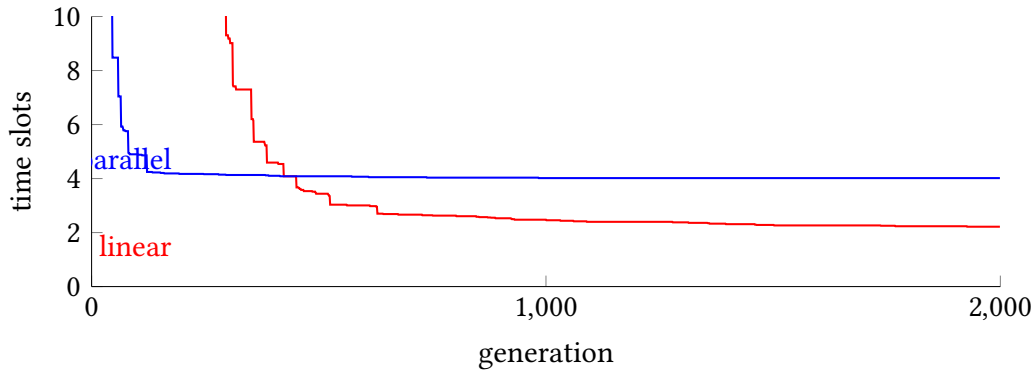


Figure 4.5: Average number of time slots depending on the generation of the linear application graph (red) and the parallel application graph (blue)

4.2.1 States and Actions

The state space represents the placement of jobs on the infrastructure nodes. Consequently, a state is given by an array of length $|B|$, where each element of this array has a value $\in V$; the total number of states is given by $|V|^{|B|}$.

The placement solution per state is not necessarily always valid (e.g., the capacity constraint may be violated as in Figures 4.7 (b) and 4.7 (c)). Therefore, the agent takes an action to move the placement of a virtual function from one node to another (Figures 4.7 (c) and 4.7 (d)); the total number of possible actions per state is equal to $|V| \cdot |B|$. Note that actions are only taken, if and only if the placement solution is invalid.

4.2.2 Reward Function

The main objective of RL is to find a solution as fast as possible, i.e., with minimum number of steps, while still minimizing the number of used time slots. Hence the reward for every state-action is based on two main criteria: first, it checks whether the new state's solution satisfies all constraints (Figure 4.7 (d)). Second, it calculates the number of timeslots required to send the data from source to destination, based on the placement solution (Figure 4.7 (f)) and the resulting route.

Each state-action pair yields a reward according to its placement solution. If the solution violates any of the constraints (Section 3.1.1), a negative value ($-r_\zeta$) is given every time the constraints are violated (Eq. (4.3))

$$R_t = -r_\zeta \tag{4.3}$$

If the state has a valid placement solution, the shortest path between the nodes used for placement is calculated, where a path is determined for each link in the application graph (Figure 4.7 (e)). Then, I calculate the number of timeslots needed to send the data along the paths, by solving for the duplex (Eq. (3.7)) and interference (Eq. (3.8)) constraints. Finally, I terminate the episode and calculate the reward using Eq. (4.4)

$$R_t = \frac{|B| \cdot r_\zeta}{\sum_{i \in T} \delta_i} \tag{4.4}$$

4.2.3 Exploration Rate

I investigate two variants for exploring the environment; Greedy epsilon and Epsilon decay (Section 2.3.3).

Greedy epsilon

I fix the epsilon parameter over all steps for all episodes to explore and exploit the environment. Accordingly, an agent selects a random action with a probability equal to ϵ.

Epsilon decay

Epsilon is initially set to 1, meaning that the agent chooses actions at random to explore the new environment and update the Q-table. After each episode, to ϵ is reduced to slowly transition from exploration to exploitation.

4.2.4 Evaluation of Reinforcement Learning for Placement

I define two different infrastructure graphs with 4 and 6 nodes. For each one, I change the source and destination nodes, the attenuation between the nodes, and the nodes' capacities, to have in total 50 different scenarios per each graph. I fix the application graph to a linear topology of five jobs, where all links in the application graph require the same data rate.

I set the hyper-parameters for the RL algorithm to $\alpha = 0.1$ and $\gamma = 0.6$. When using Greedy epsilon, I set $\epsilon = 0.1, 0.5, 0.9$ and fix it later to $\epsilon = 0.1$ (Section 4.2.4). For the Epsilon decay variant, I decrease the value of ϵ by $\frac{1}{\frac{|\text{episodes}|}{2}-1}$, and stop decaying epsilon after $\frac{|\text{episodes}|}{2}$ episodes. The maximum number of episodes is set to triple the number of states, so that we have 3072 episodes and 23328 episodes for 4- and 6-nodes networks, respectively.

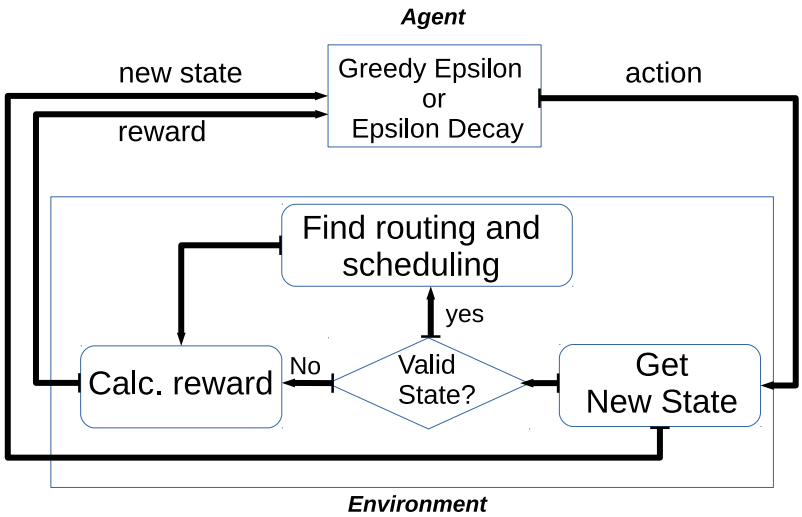


Figure 4.6: RL framework describing the role of agent and the process of environment as described in Section 4.2.1

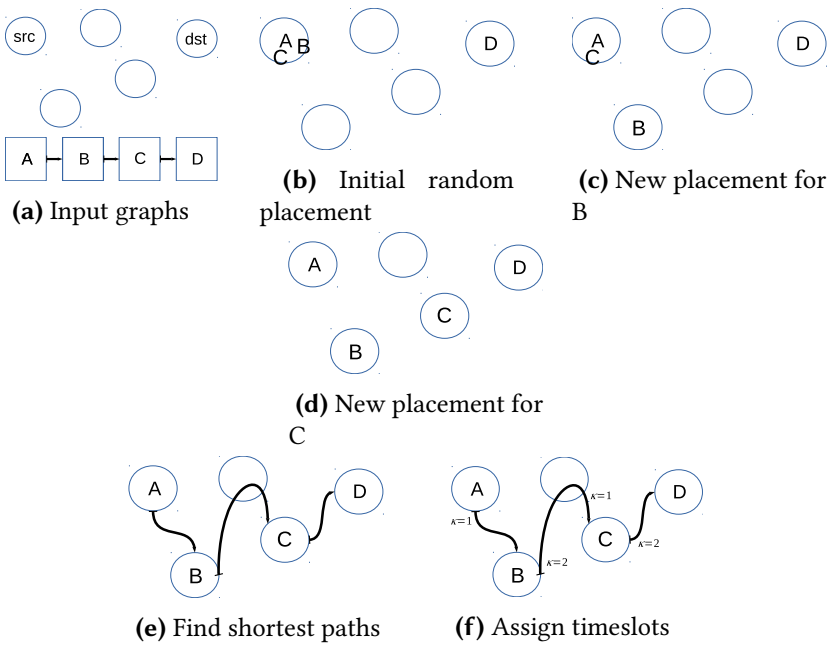
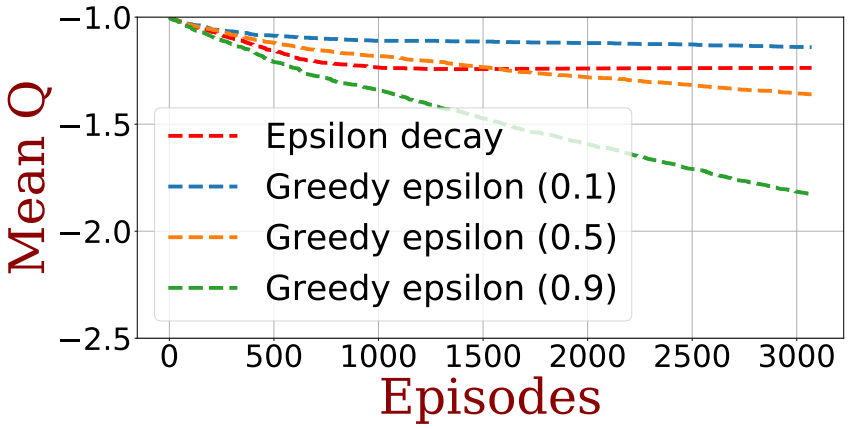
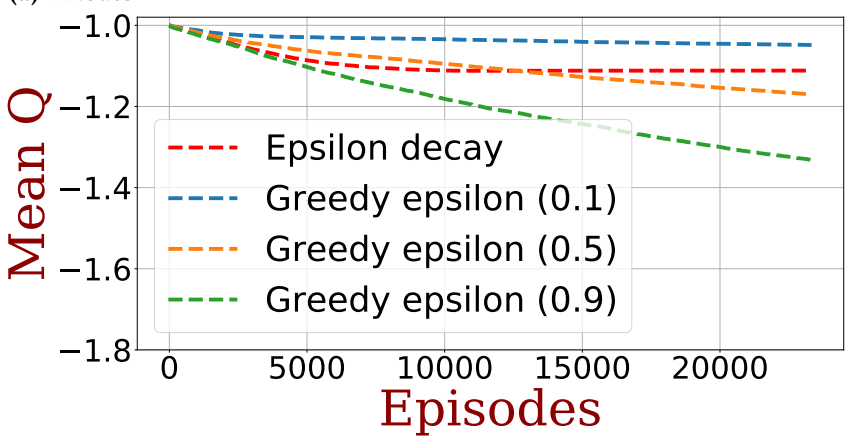


Figure 4.7: Visualizing the RL framework steps; assume all nodes can run one process at maximum.



(a) 4 Nodes



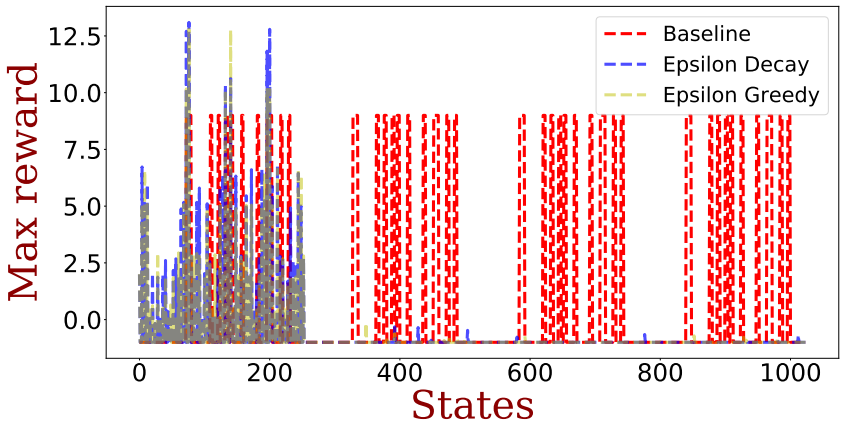
(b) 6 Nodes

Figure 4.8: Mean Q-value convergence

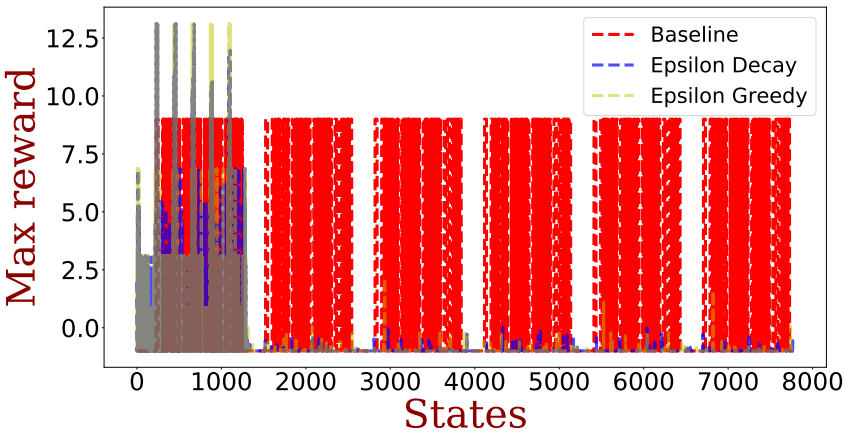
Value based convergence

I first investigate the Q-table's convergence when using Epsilon decay and Greedy epsilon with different values for ϵ (0.1, 0.5 and 0.9). In Figure 4.8, we observe that when using $\epsilon = 0.1$ or using Epsilon decay, the mean value for the Q-table seems to converge within 2000 episodes for 4 nodes network (Figure 4.8 (a)) and 10000 episodes for 6-nodes network (Figure 4.8 (b)). However, when $\epsilon = 0.5$ or 0.9, the mean value does not converge neither for a 4-nodes network nor for a 6-nodes network.

It is expected that the higher the value for ϵ is, the more episodes it needs to converge. Nevertheless, if we choose a very low value for ϵ, we will converge

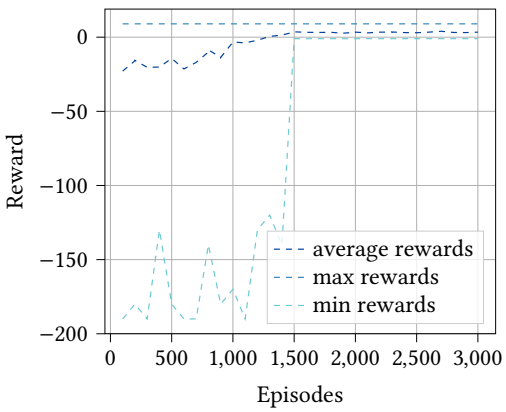


(a) 4 Nodes

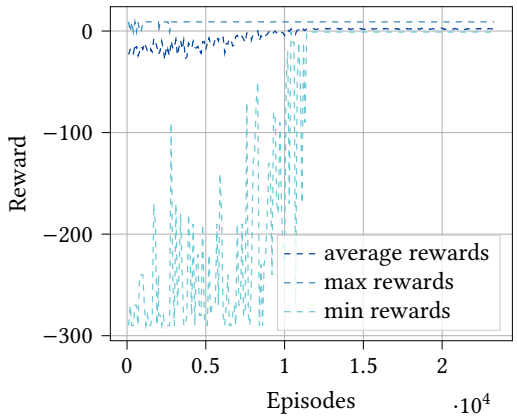


(b) 6 Nodes

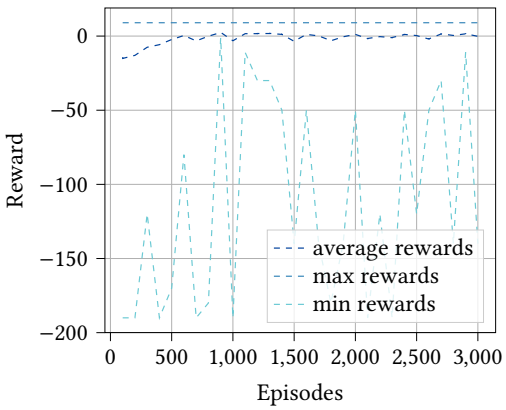
Figure 4.9: Explored maximum reward per state



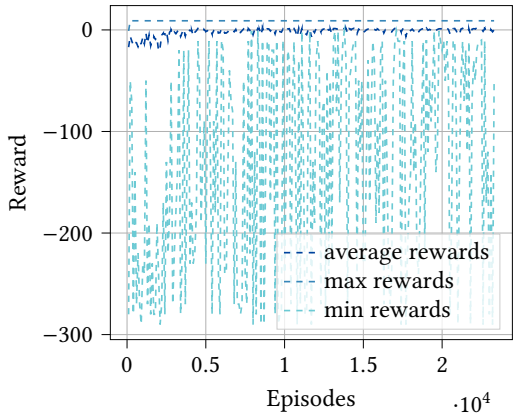
(a) 4 Nodes, Epsilon decay



(b) 6 Nodes, Epsilon decay



(c) 4 Nodes, Greedy epsilon



(d) 6 Nodes, Greedy epsilon

Figure 4.10: Average reward over episodes.

fast but also prematurely to a misleading or suboptimal value, because we do not explore all possible good choices. Accordingly, I fix Greedy epsilon to $\epsilon = 0.1$ for the rest of the evaluation and compare the goodness in Section 4.2.4.

Exploration and Exploitation

To investigate the exploration for Greedy epsilon and Epsilon decay, I show in Figure 4.9 the resulting maximum reward from all actions for each state. The baseline represents the maximum ground truth for each state. For 4 nodes (Figure 4.9 (a)), we have 1024 states where both Greedy epsilon and Epsilon decay tend to give higher weights for the lower numbered states. We observe a similar behavior for a 6-nodes network (Figure 4.9 (b)) with 7776 states.

This behavior is due to the greediness of both approaches: The baseline has the same reward for different states, therefore, a greedy approach will select an action that exploits the first (i.e., the lowest number) state, out of all possible actions with the same reward.

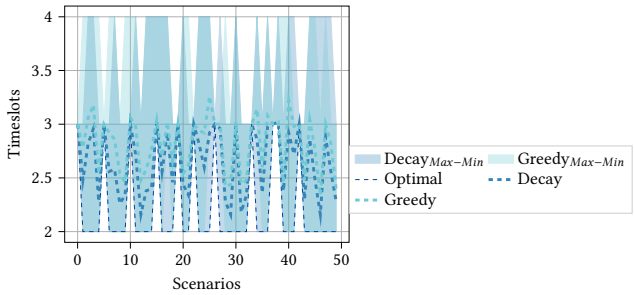
When taking a closer look at the reward and see how it evolves with increasing number of episodes (Figure 4.10), over the last 100 episodes, I show the maximum, minimum and average rewards for the selected actions.

On the one hand, we observe that for the Epsilon decay method (Figure 4.10 (a) and Figure 4.10 (b)), the minimum rewards converge and become close to the average reward after 1600 and 11000 episodes for 4 and 6 nodes networks, respectively. This is due to the decaying property of *epsilon*, where the agent becomes greedier as *epsilon* approaches zero.

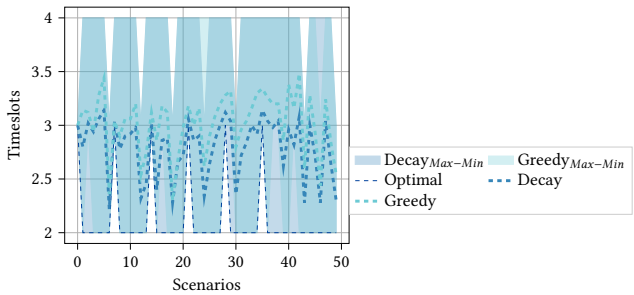
On the other hand, the minimum reward for the Greedy epsilon (Figure 4.10 (c) and Figure 4.10 (d)) has a high variance, because the greediness of the agent is fixed ($\epsilon = 0.1$) and it does not always aim at maximizing the reward, but also try other random actions. Meanwhile, the average values converge similar to the Epsilon decay method.

Comparison to the Optimization Model

Next, I focus on the resulted timeslots using Greedy epsilon and Epsilon decay (Figure 4.11). I simulate 50 different scenario, where each scenario has different source and sink nodes, different attenuation between the nodes, and different capacities per nodes. Each scenario is solved 50 different times using Greedy epsilon and Epsilon decay methods. Out of the 50 solutions, I calculate the mean, maximum and minimum achieved number of timeslots. Additionally, I calculate



(a) 4 Nodes



(b) 6 Nodes

Figure 4.11: Resulted number of timeslots

the optimal number of timeslots per each scenario using the optimization model in Sections 3.1.1 and 3.2.1.

As expected, the optimal number of timeslots (blue) acts as lower bounds for both Epsilon decay (red) and Greedy epsilon (yellow) methods. Moreover, I observe that in worst case scenarios, for both methods, the additional number of timeslots (compared to the optimal solution) is 2 timeslots, for both 4- (Figure 4.11 (a)) and 6-nodes (Figure 4.11 (b)) networks. Meanwhile, the mean number of timeslot of the Epsilon decay is always less than or equal to that of Greedy epsilon.

I show in Figure 4.12 the confidence interval for the average number of timeslots using both methods for all scenarios per each network. Although the Epsilon decay method has lower mean for both networks, the confidence intervals do not really allow to conclude that the decay method is always better than the greedy one. Nevertheless, the results from Figure 4.11 (a) and Figure 4.11 (b) show that the decay method is at least as good as the greedy one in all 50 scenarios. It is worth mentioning that the number of steps required to find a valid solution are almost the same: on average 1.8 steps using the Epsilon decay and 1.98 steps using the Greedy epsilon.

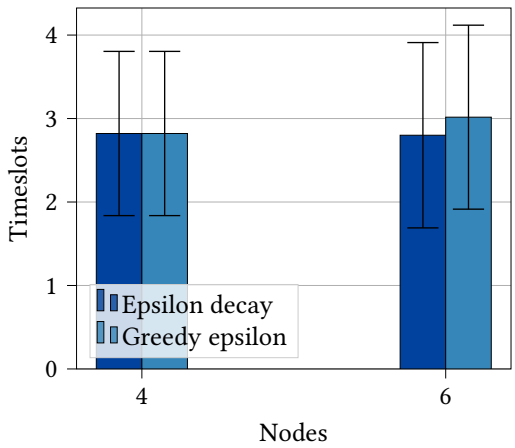


Figure 4.12: Epsilon decay vs Greedy epsilon

4.3 Conclusion

Here, I used meta-heuristics to solve the wireless VNE problem. First a genetic algorithm was defined and used to solve the problem. Then, I showed that the feasible solutions found by the genetic algorithm are close to the optimal solutions. Another RL-based meta-heuristic is also used to solve the wireless VNE problem, where the results are also good compared to the optimal ones.

Estimating the exact gap between the meta-heuristics and the optimal solutions is not easy since both solutions rely on initial random seeds. They have the same objective. however, they are recommended in different scenarios. The RL-based heuristic aims at finding a good solution as fast as possible and then it terminates, making it suitable for handling network failures with minimum number of migrations. Meanwhile, the genetic algorithm aims at finding a solution, but if it is given more time, it may find a better one, making it suitable for problems with distinct due dates. Hence, depending on the user's preference, any of the solutions is chosen.

5 Admission Control for Incoming Jobs

In general, acoustic applications require high Quality of Service (QoS) deliveries (e.g., low end-to-end delays) as seen in streaming services [Drä+18] . In previous chapters, I showed how an application is divided into multiple fine-grained jobs, where each job has some processing requirement (e.g., CPU resources) as well as the communication between the jobs (e.g., data rate). To embed this application within the network, I use VNE but without over-utilizing any of network resources, with the objective of having high QoS.

In this context, a controller receives a request to embed an application, i.e., Virtual Network Request (VNR), and then it decides how to embed the VNR. The objective would then be to maximize the QoS. If the controller cannot find a feasible VNE solution, it rejects the VNR.

A naive controller simply embeds VNRs as first-come-first-serve. But this may limit the revenue/efficiency from utilizing the physical network, especially if the VNRs have different resource requirements. Admission control is one way of handling this issue; a controller may reject a VNR with high requirements to accept later on more VNRs or to later accept VNRs with higher revenues. In this chapter, I focus on WSN and use two parameters to describe incoming VNRs:

1. duration of leasing the network resources
2. priority of the VNR to express the revenue/importance

The main objective of the admission control is then to embed as many VNRs as possible (or more VNRs with higher revenues) while minimizing unnecessary rejections. An important challenge to consider, when designing such a controller, is the uncertainty of future incoming VNRs. For example, this could be related to dynamic arrival rate or the incoming VNRs parameters. I focus here on the latter, so that future VNRs have different lease duration and different priorities.

Instead of using the naive admission, RL may be a promising candidate that can better deal with this challenge. In the RL framework, an agent interacts with the environment (i.e., VNE solution) to optimize its policy of accepting or rejecting a VNR, without getting information about future VNRs.

Generally speaking, using an RL approach in the context of WSNs has been used to solve many problems as in MAC [Mus+17], energy saving [CLZ16], and many

other similar problems [PAA19]. Only some work focused on VNE, admission control and RL together. The authors in [BTS98; RTL20] assume that the QoS is a constraint; hence, their proposed solutions reject VNRs that are likely to violate QoS bounds. In contrast to their assumption, I assume that QoS is an objective for the VNE problem, while the admission control maximizes the embedding revenue. Furthermore, the work in [RTL20] assumed arriving VNRs wait in a queue for a decision to be embedded or to be rejected. Meanwhile, in this chapter, and similar to [BTS98], I have a queue length of only one. Therefore, queuing issues have been ignored to emphasize other features of the admission control problem.

I use a VNE solution to check the feasibility of embedding a VNR. To combine both admission control and VNE problems, the work in [Ble+16] uses recurrent neural networks to reject VNRs that are likely to fail the QoS constraints, which saves the computational time needed by the VNE solution to check the VNR implementation feasibility. Hence, the admission control was trained to have high accuracy of detecting feasible accepted VNR embedding. In this chapter, the admission control is trained to maximize the network's revenue, meaning that it can reject feasible VNRs in order to accept more VNRs later.

Combining both admission control and VNE decisions has been solved using RL in [Ble+18; Yao+20]. Similar to the work presented here, the objective is to maximize the network's revenue. Unlike those references, I assume a modular implementation, where admission control and VNE solutions are two separate modules that interact with each other. This allows reusing different modules in similar problems. Additionally, the simplicity of this problem will probably lead to less training time (i.e., time for the models to converge). For example, the mix of VNE and admission control into a single module, as shown in [Yao+20], is valid only for wired networks and cannot be applied or reused in the wireless network, due to the differences between wired and wireless VNE problems. The presented work is from [ASK21].

5.1 Problem formulation

The objective is to maximize the acceptance rate of incoming VNRs while also prioritizing some VNRs over others. Hence, this would require rejecting VNRs, even if they could be embedded resource-wise, to allow more future VNRs to be embedded instead. In the following subsections, I describe the features of the VNRs and the wireless network for formulating our problem.

5.1.1 Virtual Network Requests

Each job requires a processing capacity c_{req}, while all links have the same minimum required data rate r_{req} to guarantee an upper-bound delay. Similar to previous chapters, I assume that the propagation and processing delays are negligible and the medium access delay is constant.

I assume that we have a predefined set of VNRs, whose topologies are known in advance. Hence, each VNR is labeled with an Identifier (ID), where the duration and priority of each VNR can change, but the topology and required resources do not.

Additionally, I assume that I have a discrete time environment where at each time slot a new VNR v arrives, whose duration δ^v is uniformly distributed between $[\delta_{\text{min}}, \delta_{\text{max}}]$ time steps. By altering the value of δ_{min} and δ_{max}, the average VNR duration can be adjusted to simulate different loads. Similarly, each VNR has a priority $\lambda^v \in [\lambda_{\text{min}}, \lambda_{\text{max}}]$. The priory can express the revenue of running the application or how important it is compared to other applications.

δ^v and λ^v play an important role in the admission decision. For example, if the goal is to maximize the number of admitted VNR, it is very likely to admit only VNR with short duration. In this chapter, I alter the objective between, maximizing the acceptance rate, having a high revenue and a mix between both objectives (see Section 5.2.3).

5.1.2 Wireless Sensor Network

The wireless sensor nodes operate in half-duplex communication mode; a node can be in one of three states: send, receive, idle. Each node $p \in P$ is defined using this set of properties: position in network X^p, computational capacity C^p, actual transmit power S^p and noise floor N_0.

I assume that all sensor nodes share the same collision domain with bandwidth BW. To allow multiple channel access, I use TDMA, where a time step is divided into T time slots. Consequently, two nodes can transmit data of different links simultaneously.

I use a simple distance-based model (5.1) to define the attenuation $\gamma^{i,j}$ based on the distance d between two nodes $i, j \in P$

$$\gamma^{i,j} = \frac{1}{d(X^i, X^j)^2}. \tag{5.1}$$

Accordingly, the maximum achievable data rate $r_{\text{max}}^{i,j,t}$ between two nodes $i, j \in P$ at time slot $t \in T$ is given by

$$r_{\max}^{i,j,t} = \frac{\mathrm{BW}}{|T|}\log_2(1 + \frac{S^i \gamma^{i,j}}{I^{i,j,t} + N_0}) \tag{5.2}$$

where $I^{i,j,t} = \sum_{\substack{p \in P^t \\ p \neq i}} \gamma^{p,j} S^p$ is the interference at node j from other nodes P^t that are simultaneously transmitting with node i at time slot t.

5.1.3 Constraints

A successfully embedded VNR v will decrease its remaining duration $\delta^v = \delta^v - 1$ in each time step; v runs as long as $\delta^v > 0$. Second, let us assume that we have a list of VNRs V^+ which are currently running inside the network (i.e., $\delta^v > 0, \quad \forall v \in V^+$). Then, we need to follow the typical VNE constraints in Section 3.2: capacity, flow conservation, duplex constraints.

5.1.4 VNE Heuristic Solution

The VNE solution is a straightforward, first-fit constructive heuristic as described in Section 3.3; it finds a solution by following a sequence of pre-ordered constraints. First, wireless nodes are chosen in topological order of the jobs, while still checking the capacity constraint. If any node does not satisfy the capacity constraint, another node is selected at random. Next, shortest path routes are computed between the nodes running the jobs. At the end, time slots are allocated to the transmissions between the nodes in a topological order. We start with one time slot. If simultaneous transmissions cannot take place within the available time slots – i.e., due to duplex or SINR constraints – new additional time slots are used for transmissions.

5.2 Reinforcement Learning

To train the RL agents, the Proximal Policy Optimization (PPO) [Sch+17b] RL algorithm is used, because it is recommended for accelerating the training process, thanks to its multiprocessing implementation[Hil+18]. Other RL algorithms (e.g. Q-Learning) would have been possible, but the training time would have been longer. In the following subsections, I define the observation space, the action space and the reward function of the RL environment.

ID^1	ID^2 ID^3 ..... ID
$l_1 = (b_1, b_2)$	...
$l_2 = (b_2, b_3)$	...

VNR example

ID^1	ID^2	ID^3
(λ_1, δ_1)	.....	

VNR properties

Sending nodes

receiving

	ID^1, l_1	–
–		ID^1, l_2
–	–	–

η link activation example

Figure 5.1: Example of activation link for 1 time slot and how it relates to VNR properties.

5.2.1 Observation Space

All observations are stored in a multi-dimensional discrete vector containing the following information:

- Node capacities: $\longrightarrow \mathbb{R}^P$
- Link activation $\equiv \eta \longrightarrow \mathbb{R}^{P \times P \times T \times 2}$
- New VNR's properties $\longrightarrow \mathbb{R}^3$
  - ID
  - Duration
  - Priority

Node capacities contain the available capacities of each node at the current time step. Link activation is a multi-dimensional matrix containing the activation status of all currently embedded links. The first and second dimension encode the sending and receiving nodes (Figure 5.1). The third dimension stands for the time step at which the transmission takes place. Each transmission is labeled by 2 IDs representing which VNR and which link within the VNR. For newly incoming VNRs, the VNR ID, duration δ^v and priority λ^v are added.

5.2.2 Action Space

The RL agent decides whether an incoming VNR is rejected or accepted (binary space). In case of acceptance, the VNE algorithm will try to embed the VNR into the network.

5.2.3 Reward Function

The reward function is chosen to train the agent towards the desired behavior described in Section 5.1.

Table 5.1: Reward function

Agent decision	VNE solution	Label	Reward
accepted	feasible	*true positive*	+6
accepted	infeasible	*false positive*	-2
rejected	feasible	*false negative*	-1 + r_{extra}
rejected	infeasible	*true negative*	±0

Table 5.1 describes the combinations of agent decisions and VNE algorithm solutions with their corresponding reward. An incoming VNR is labeled as *true positive* if the agent decides to accept and the VNE algorithm successfully embedded the VNR. The other labels describe the remaining combinations of agent's decisions and the VNE solution feasibility.

During training, *False negatives* compute VNE, although the agent rejects the VNR. This is used so that *False negatives* receive an extra reward r_{extra} calculated using Eq. (5.3). It relies on the relative delay ($f_\delta^v = \frac{\delta^v}{\delta_{\max}}$) and the relative priority ($f_\lambda^v = \frac{\lambda_{\max}-\lambda^v}{\lambda_{\max}-1}$) of the rejected VNRs v.

$$r_{\text{extra}}^v = c_\delta \cdot f_\delta^v + c_\lambda \cdot f_\lambda^v \tag{5.3}$$

The control parameters c_δ and c_λ are used to tune the *false negative* behavior on the extra reward. The extra reward is used to recompense for rejecting VNR, when a VNR has a very long duration, low priority or both.

5.3 Simulation Setup

The number of time slots per time step $T = 8$ for all simulations in this chapter. 5 nodes are placed in a small-sized room (e.g., a seminar room) with dimensions

5 m × 5 m, ×3 m. The wireless channel bandwidth is set to BW = 20 MHz. The minimum time duration per VNR $\delta_{\text{min}} = 2$, while the VNR's priority is uniformly distributed between $\lambda \in [1, 10]$. All trained agents are compared to an *always accept* baseline agent. This baseline agent will accept each incoming VNR and forward it to the VNE algorithm. Hence, the embedding of a VNR depends only on the VNE solution's feasibility. With respect to Table 5.1, the output of this algorithm corresponds only to *true positives* and *false positives*.

To measure the impact of the control parameters (c_δ and c_λ) and the sensitivity of the trained agent to incoming VNR properties, 3 simulation setups are defined:

1. fix c_λ and change c_δ
2. change c_λ and fix c_δ
3. fix both c_λ and c_δ, while evaluating different agents trained on different δ_{max}

The VNR used in this simulation is made of three processing jobs $B = \{\text{K}, \text{L}, \text{M}\}$ connected via two links ($E = \{\text{KL}, \text{LM}\}$) as shown in Figure 5.2.

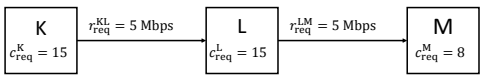


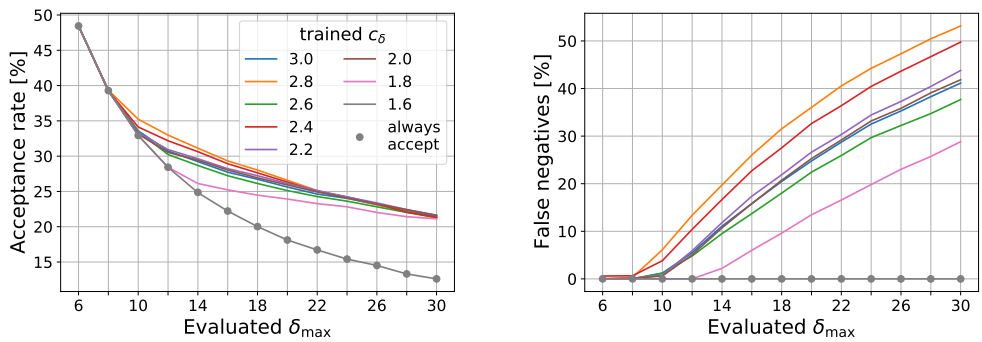
Figure 5.2: VNR overlay graph

5.4 Simulation Results

In general, for each configuration setup there are 100 different runs, where each run has 1000 time steps (i.e., 1000 incoming VNRs). The median of these runs is compared to that from the baseline solution (gray dots). Note that the acceptance rate in the following subsections is calculated after the RL agent and the VNE heuristic.

5.4.1 Duration Control Parameter

As stated in Section 5.2.3, c_δ controls the agent's decision (i.e., accept/reject a VNR v) with respect to the VNR's duration δ^v. To illustrate the sensitivity of agent's decision to this parameter, eight agents are trained, where I manually c_δ between



(a) Acceptance rate

(b) False negatives

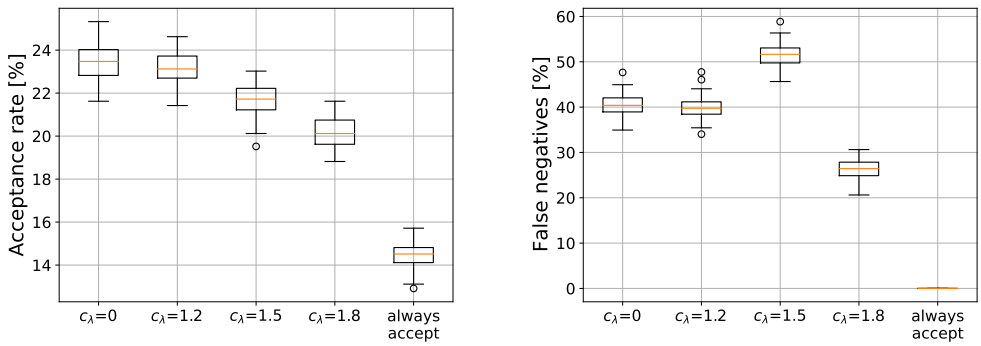
Figure 5.3: Results of c_δ parameter analysis

$[1.6, 3.0]$. All agents have the same number of training time steps 10^6 and $\delta_{\max} = 30$ (Figure 5.3)

Figure 5.3 (a) shows the median acceptance rate ($\frac{\text{number of accepted VNR}}{\text{total number of incoming VNRs}}$) values of all different trained agents and the *always accept* baseline agent in different offered load environments. Figure 5.3 (b) shows the corresponding relative number of *false negatives* created by those agents.

The agent trained with $c_\delta = 1.6$ does not increase overall acceptance rate compared to the *always accept* baseline agent and does not make any *false negatives* decisions. In other words, the agent learns to behave the same way as the baseline. Meanwhile, agents with higher values for c_δ show an increase in overall acceptance rate especially in higher offered load environments, i.e., better than the baseline.

This increase comes with the cost of creating more *false negatives*, i.e., unnecessary computations for the heuristic. The difference between each other is, as expected, most present for medium offered loads. The agents with $c_\delta = 1.8, 2.0, 2.2, 2.4$ show a gradual increase in acceptance rate compared to each other when evaluating $\delta_{\max}$ between 10 and 22. For incoming VNRs with higher $\delta_{\max}$, the agents perform very close to each other, i.e., diminishing return. Hence, the agents with $c_{\mathrm{d}} = 2.6, 2.8, 3.0$ do not further increase the acceptance rate compared to agents trained with smaller values for c_{d}. Meanwhile, in Figure 5.3 (b), increasing c_δ tends to have more *false negatives*. Accordingly, at some point, increasing c_δ does not increase the acceptance rate and just increase the false negatives.



(a) Acceptance rate **(b)** False negatives

Figure 5.4: Results of c_λ parameter analysis with $c_\delta = 1.8$ and $\delta_{\max} = 26$

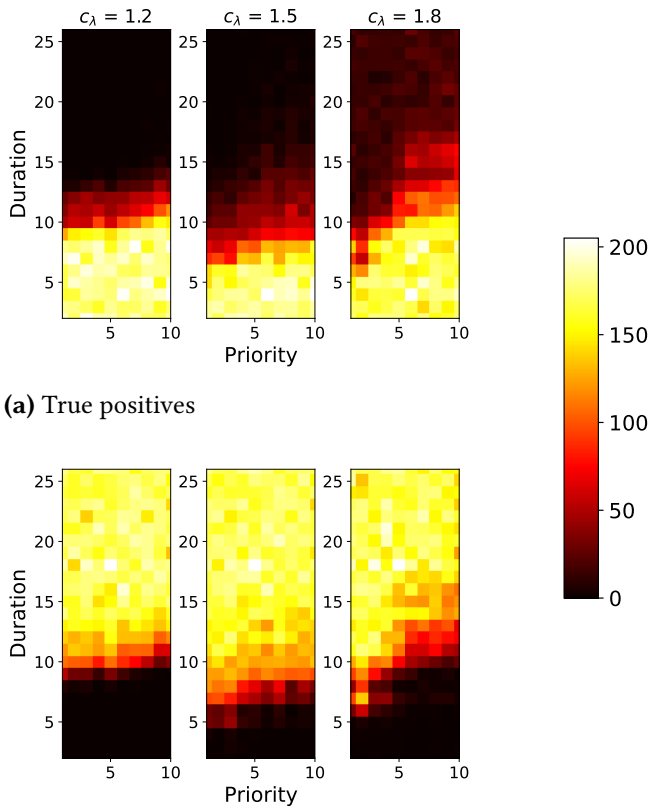
5.4.2 Priority Control Parameter

All previous analyses focused on maximizing the acceptance rate by rejecting long VNRs. This may not be ideal because it prevents longer VNRs from being embedded at all. for a different goal, c_λ can be used to give higher priority for longer VNRs (e.g., higher revenue). Figure 5.4 shows the evaluation results of the trained agents with different values for c_λ and $\lambda \in [1, 10]$, while all agents use $\delta_{\max} = 26$ for training and evaluations.

In Figure 5.4 (a), it is observed that the higher the value of c_λ during training, the lower the acceptance rate of the resulting agent. This is due to accepting VNRs with high $\delta_{\max}$ and high priority. Meanwhile, the number of *false negatives* tends to decrease as c_λ increases.

To extend the analysis, in Figure 5.5, I investigate which VNRs are getting accepted or rejected during evaluation. Figure 5.5 (a) shows the number of *true positives* while Figure 5.5 (b) shows the number of *false negatives* using different values for c_λ during training. Each point represents the number of VNRs with their corresponding duration and priority values. For the agent with $c_\lambda = 1.2$, most *true positives* are short VNRs of different priorities. As the VNR duration increase, they are being rejected by the admission control; very long duration VNRs are not embedded at all, which can be seen in the dark area. The results for *false negatives* in Figure 5.5 (b) confirm this behaviour. Mainly, long VNRs are being rejected, even if they can be embedded.

Higher values for c_λ have two obvious impacts on the acceptance rate (Figure 5.5 (a)). First, the slope of the red boundary increases – the agent starts to accept longer VNRs with high priorities rather than short VNRs with low priority.



(a) True positives

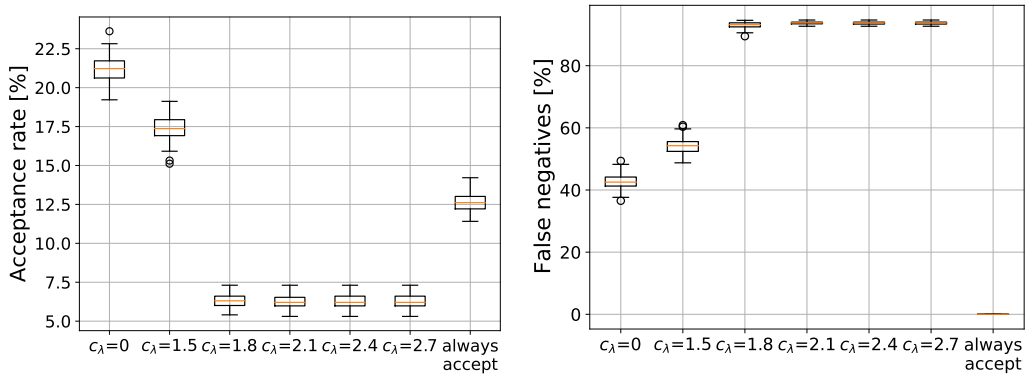
(b) False negatives

Figure 5.5: Impact of c_λ on *true positives* and *false negatives* with $c_\delta = 1.8$ and $\delta_{\max} = 26$.

Second, the dark area in the *true positive* graphs gradually changes to red. That means the agent becomes more flexible toward accepting long VNRs, given that these strict constraints no longer exists.

Depending on the use case (e.g., seeking higher revenues) this behaviour can be beneficial – when increasing c_λ ,long VNRs get a chance to be embedded rather than being always rejected.

Meanwhile, c_λ should be carefully tuned, otherwise it will lead to an undesired agent behaviour. In Figure 5.6, I retrain the environment for $c_\delta = 30$ and, again, with increasing c_λ. It is observed that agents trained with high values for c_λ eventually start to prioritize *false negatives* (Eq. (5.3)). Hence, the results could be even worse than the baseline solution: lower acceptance rate (Figure 5.6 (a)) and higher false positive (Figure 5.6 (b)).



(a) Acceptance rate **(b)** False negatives

Figure 5.6: Results of c_λ parameter analysis with $c_\delta = 1.8$ and $\delta_{\max} = 30$

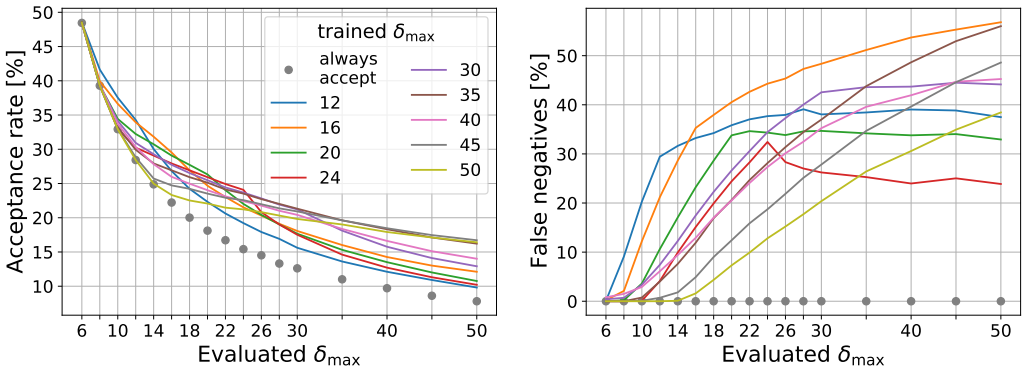
5.4.3 Maximum Trained Duration

In the previous evaluations, agents are trained with a fixed $\delta_{\max}$ and evaluated for different incoming $\delta_{\max}$. *What if the agents are trained for a shorter $\delta_{\max}$? How would that impact the acceptance rate for incoming VNRs with longer, shorter, or equal to the trained $\delta_{\max}$?*

To answer these questions, in total 9 agents are trained for $\delta_{\max}$ between 12 and 30 and evaluated, in addition to the baseline, for different $\delta_{\max}$ (Figure 5.7). All training runs are performed with $c_\delta = 1.8$, which is a compromise between increasing acceptance rate and limiting the number of *false negatives* as described in 5.4.1, and $c_\lambda = 0$.

In Figure 5.7 (a), each agent shows the highest increase in median acceptance rate compared to the *always accept* baseline agent in the exact scenario it was trained for. For higher offered loads, the performance of agents drops but still stays above the baseline level. This behaviour results from agents not having encountered any larger VNR duration during training; thus, they do not know how to properly deal with them during evaluation.

This can also be interpreted from Table 5.2. Agents perform mostly the best when the trained and evaluated $\delta_{\max}$ are the same. As the difference between the trained and evaluated $\delta_{\max}$ increase, the acceptance rate decrease, but it is still better than the baseline solution. Meanwhile, Figure 5.7 (b) does not show consistent patterns for the relation between trained and evaluated $\delta_{\max}$ with respect to *false negatives*. Hence, it depends more on the environment and should be tuned with respect to other environment parameters (e.g., distribution of arriving VNR duration).



(a) Acceptance rate **(b)** False negatives

Figure 5.7: Impact of trained δ_{max} on agent performance in different δ_{max} enviromnets

Table 5.2: Comparison between agents with different trained δ_{max} in multiple δ_{max} environments

eval δ_{max} / trained δ_{max}	12	16	20	24	30	35	40	45	50
12	**34.3**	26.8	22.3	19.2	15.6	13.6	12.1	10.9	9.8
16	33.9	**29.5**	24.7	21.5	18.1	16.0	14.3	13.0	12.1
20	32.2	29.1	**26.3**	22.0	17.7	15.3	13.5	12.0	10.8
24	30.2	27.9	25.9	**24.1**	17.5	14.6	12.7	11.3	10.2
30	30.9	27.7	25.5	23.7	21.2	18.1	15.8	14.1	12.9
35	29.9	26.9	25.1	23.5	**21.3**	**19.6**	18.3	17.1	16.2
40	30.3	26.0	24.0	22.4	20.4	18.4	16.6	15.1	14.0
45	28.8	24.7	23.5	22.6	20.9	19.6	**18.5**	**17.5**	**16.7**
50	28.4	23.3	22.1	21.2	19.8	19.0	17.9	17.1	16.4
baseline	28.4	22.2	18.1	15.4	12.6	11.0	9.7	8.6	7.8

5.5 Summary

This chapter uses RL for admission control of VNRs in wireless VNE. RL agents are trained to maximize the revenue (e.g., acceptance rate or number prioritized VNR) by rejecting some VNRs to be able to accept more/better future VNRs. The agent's behavior can be tuned by modifying the parameters c_δ and c_λ. On the one hand, low values of c_δ lead to a behavior with little to no increase in overall acceptance rate while keeping the number of *false negatives* small. High values of c_δ yield more *false negatives* aggressively, resulting in a higher overall acceptance rate, but with a diminishing return. On the other hand, the higher c_λ the more likely the agent consider priority over duration. Hence, longer VNRs with a high priority value can be accepted. This should be carefully tuned, otherwise the results can be even worse than a first fit baseline. Meanwhile, training an agent with a predefined $\delta_{\max}$ yields a better solution than the first fit baseline, even if there is a mismatch between the trained and deployed/evaluated $\delta_{\max}$.

6 Impact of MAC Protocols

In the previous chapters, I assumed that the number of wireless nodes are fixed and the nodes do not move. Additionally, when fixing the application, the data rate is known in advance. For example, the data rate generated from microphones for acoustic applications are fixed and given. Accordingly, it makes sense to use a TDMA MAC. However, TDMA requires signalling protocols that can create substantial overhead, especially when there are external factors (e.g., heat) influencing the clocks used for wireless transmissions. This makes TDMA less attractive even though it offers tight bounds on medium access delay and guaranteed data rates. As an alternative, typical ad hoc standards (as seen in IEEE 802.11 family) rely on CSMA/CA or listen-before-send protocols. They are, however, subject to collisions, high delays and fluctuating data rates[2].

In much the same way, data synchronization is a critical issue that appears for multimedia applications with distributed sensors/actuators, where audio and video devices are separated in space. In this context, data synchronization means synchronizing the timing of data collected from different sensors (e.g., microphones and cameras). This is indeed crucial for many applications such as lip synchronization, video animation, and real-time audio streaming [Ban+09]. As a concrete example, let us consider acoustic source localization in acoustic sensor networks using audio synchronization [DSB01] via a set of distributed microphone nodes. Without synchronizing the audio streams of the nodes, the localization suffers, for example, from the problem that the Time Differential of Arrival (TDoA) of the signals changes over time [GSH21]. Consequently, the acoustic source seems to be moving even while it actually remains at a fixed position.

In this chapter, I ask how to reduce the synchronization overhead for TDMA protocols and whether they become a competitor to CSMA/CA protocols. To do so, I reuse the results from an application running audio stream synchronization between microphones [GSH21] to synchronize the time slots for wireless transmissions. In a sense, I leverage application synchronization for MAC synchronization. To bootstrap this process, the application starts running using CSMA/CA for an acoustic SRO estimation. Once audio synchronization accuracy is below a certain threshold, I recommend to switch to TDMA for a more stable or a higher network

2 I assume that RTS/CTS is turned off

throughput. In general, I look into the following aspects: How to set the accuracy threshold for switching? Is the resulting audio synchronization good enough for synchronizing TDMA slots? How big do the guard intervals need to be? And overall, what would be the gain of switching between CSMA/CA and TDMA?

I use throughput (number of bits successfully *received* per time unit) as a metric to evaluate the performance of the two MAC protocols and show the relation between audio synchronization error, back-off interval and achieved throughput. On the one hand, throughput is chosen as a metric due to the fact that most multimedia applications exchange a substantial amount of data. On the other hand, I assume that these applications do not have delay requirements, where the accuracy of the synchronization from the application (in this chapter audio synchronization) is independent of network delays. The network delays may, however, impact the time required to estimate the SRO and to synchronize the audio streams.

Generally, several communication protocols have been proposed to synchronize the clocks of wireless nodes. The most famous protocol is Network Time Protocol (NTP) [Mil91], yet it was originally developed to be used over the internet, i.e., wired networks, and rests on assumptions that are usually violated in wireless networks, e.g., constant transmission delays. In NTP, synchronization is achieved using a hierarchical structure of time servers, where the root node is synchronized with the Coordinated Universal Time (UTC). NTP assumes that the transmission delays between two nodes forward and backward are the same, yet this is not always true in wireless networks. Moreover, NTP adds an overhead to the limited data rate (with multi-hopping properties) of wireless networks. To address some of these problems, extensions to NTP [DH04; EGE03; GKS03; GR03] have been proposed.

To specifically address multimedia applications, the authors in [FYD09] proposed combining both TDMA and CSMA/CA MAC protocols for predictive network delays, in which, similar to NTP, the nodes exchange beacons for clock synchronization. The IEEE 1588 standard (a.k.a. precision time protocol [CEP07]) also uses beacons only for estimating time offsets but not for transmissions. A comprehensive survey on wireless MAC protocols and the synchronization protocols was done in [Mah+17; ZMS20].

What is common between the aforementioned works is that they all require additional signalling overhead to achieve synchronization. On the contrary, driving MAC synchronization from an application synchronization process that takes place anyway, and thus reducing signalling overhead, is vastly less investigated. For example, [MR14] uses the time information inside the application's query packets to synchronize the virtual clocks of the nodes. The nodes then alternate between awake and sleep states and adjust the amount of time spent in each phase (duty cycle). Hence, nodes running the same application should synchronize their virtual

clocks to have the same duty cycle. This works, however, best for applications that periodically send queries and works not so well for on-demand applications as in some multimedia applications. This is due to the fact that periodic queries, as well as periodic data traffic, regularly ensure an adequate synchronization between the nodes. But on-demand queries do not exchange packets for some periods in time [Gun+21], in which synchronization errors accumulate. This makes the synchronization problem more complex, because the synchronization error may become arbitrarily large during long periods in time without exchanging queries. Thus, the synchronization needs to be re-started when queries are available again. Here, I assume that the nodes are not synchronized at the begging of sending data flows.

In fact, many multimedia applications, such as acoustic applications, also require synchronization, but between multiple distributed clocks driving the sampling processes of the audio signals [BAK04b; DG15; GBW01]. Accordingly, many algorithms were developed using different assumptions, such as static SROs [CG17a] and time-varying SROs [GSH21]. Such algorithms run independently of the wireless synchronization and are a part of the acoustic application. Here, I leverage the output of an audio synchronization algorithm to synchronize the wireless clocks.

6.1 System Model

I consider a wireless network with $N > 2$ nodes, each equipped with a microphone. All nodes operate in the same collision domain with negligible propagation delays. The nodes exchange audio data over a single hop, i.e., no packet forwarding. I assume that neither the nodes' positions nor their used data rates change. However, the position of the acoustic source (i.e., speaker) changes over time. The buffer size of the nodes is assumed to be infinite, so that packets are buffered (and not lost) in case of wireless channel over-utilization.

In reality, there is a time-varying difference between the rates of the clocks driving the sampling processes of the audio signals of different nodes. This results in a time-varying SRO between the audio signals recorded by different nodes, whose effect accumulates over time, causing a growing SRO-induced shift τ between the audio signals of different nodes [GSH21]. Accordingly, the SRO has to be estimated from the audio signal to adapt the rates of the clocks driving the audio sampling processes before using these clocks to drive the wireless transmission.

Additionally, there are Sampling Time Offsets (STOs) between the audio signals recorded by different nodes, due to the fact that the nodes start recording at different points in time. It corresponds to a constant time shift between audio signals recorded

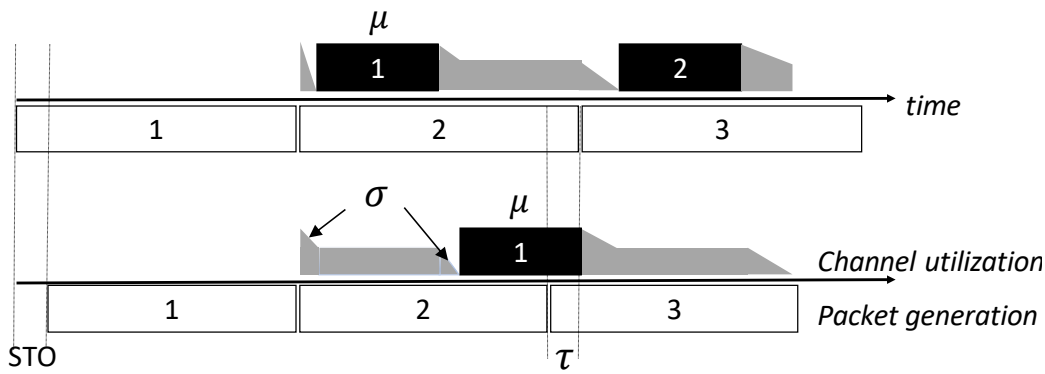


Figure 6.1: Channel access with CSMA/CA

by different nodes. SRO and STO models as well as the algorithms used to estimate these quantities have been discussed in detail in [GSH21]. I use the outcome of this work in the TDMA throughput analysis.

Next, I provide a simple analytic model for the throughput achieved by CSMA/CA and TDMA under the given assumptions.

6.1.1 CSMA/CA

The contribution of this section is to derive the throughput of CSMA/CA for a specific scenario, which will be used later for comparison with TDMA. Accordingly, I apply some modifications to the derivations in the related work [BFO96; Bia00; KSM03], to fit my scenario here.

Since nodes operate in the same collision domain, the RTS-CTS feature of 802.11-MAC [21] is turned off, and for simplicity, I ignore the delays introduced by the acknowledgment time and inter-frame spacing (IFS) [Bia00], which are typically used for carrier sensing and sending/receiving acknowledgements. For fairness, even if a node has multiple packets buffered, it only attempts to transmit one packet and then enters back-off, as specified by 802.11 [WK05].

Additionally, I assume that the contention window size is fixed and does not increase exponentially as in [21], which allows to derive the analysis for a particular contention window size W. This is justifiable for this scenario where the number and positions of nodes do not change and the application generates packets with a constant length L at the same rate (assuming that the remaining SRO is negligible after audio synchronization).

For such a predefined, unchanging setup, I will derive a suitable contention window size. Initially, a node senses the wireless channel and sends a packet if

the channel is idle. Otherwise, a random back-off interval B_n starts when node n attempts a packet transmission after sensing collisions or a busy channel. While the wireless channel is idle, the timer counts down until it reaches zero and the node transmits the packet. If the channel is detected busy before reaching zero, the timer freezes until the channel becomes idle again (Figure 6.1). The time needed to send a packet for node n is μ_n=L/R_n, where R_n is the bit rate given that node n is transmitting. I assume R_n to be below the channel's (Shannon) capacity [Sha48] for all nodes.

In practice, audio data is not continuously transmitted, since there may be time intervals without acoustic source activities. Hence, a channel may be idle but the nodes still have no data to exchange, due to speech pauses or late generated/processed packet. I refer to the fraction of time for a scenario, where there is no packet to send, given that the channel is idle, as $0 < \rho_n \leq 1$, where 1 means that nodes always have a packet to send and 0 means that nodes are not transmitting. Nonetheless, when a node has packets to send, the objective is still to send at a high throughput.

For simplicity, and unless mentioned otherwise, it is assumed that all nodes have the same data rate R, traffic load ρ and average back-off windows $E[B]$. The analysis is based on the capacity formulation in [Bia00], where the focus is on single-hop flows in a single collision domain. The channel throughput is then given by

$$S = \frac{p_{\text{tr}} p_s L}{p_{\text{tr}} p_s \tau_s + p_{\text{tr}}(1 - p_s)\tau_c + \tau_i}, \tag{6.1}$$

such that τ_s is the sending time, τ_c is the collision time, and τ_i is the expected duration of the channel's idle interval (slightly different from [Bia00] where τ_i is the duration of a single back-off slot Chapter A). The content difference is due to different focus; in [Bia00], the author focus on system throughput, while here I focus on the throughout per node.

Furthermore, p_{tr} is the probability that at least one node is transmitting, while p_s is the probability of successful transmission conditioned by at least one node transmitting. When the propagation and inter-frame spacing delays are ignored, i.e., $\tau_s = \tau_c$ [Bia00], we get

$$S' = \frac{p_{\text{tr}} p_s L}{p_{\text{tr}} \mu + \sigma}., \tag{6.2}$$

where μ is, again, the time to send a packet and σ is the average duration of the idle interval when sending a packet.

For a fixed contention window W, the back-off window time has a uniform

distribution $E[B] = \frac{W+1}{2}$. Accordingly, the probability that any node tries to access the channel is [Bia00; KSM03]

$$p_t = \frac{2}{W+1}. \tag{6.3}$$

With the probability that any node tries to access the channel, the probability of at least one transmitting node (p_{tr}) and, respectively, a successful transmission (p_s) are given by:

$$p_{\text{tr}} = 1 - (1 - p_t)^N, \tag{6.4}$$

$$p_s = \frac{p_t(1 - p_t)^{N-1}}{p_{\text{tr}}}. \tag{6.5}$$

To derive the idle interval σ, it is assumed that when a node has a packet to send, every other node also has a packet ready to transmit ($\rho = 1$). Therefore, the idle interval is only due to the back-off count down. When N nodes compete for the channel the idle interval is given by [BFO96]

$$\sigma' = \frac{E[B]}{N} = \frac{W+1}{2N}. \tag{6.6}$$

If the nodes are not fully saturated, i.e., $\rho < 1$, nodes start counting down only for a fraction of time ρ. Hence, as derived in [LK13], the relation between ρ and σ is given by

$$\sigma = \frac{\sigma'}{\rho} = \frac{W+1}{2\rho N}. \tag{6.7}$$

Accordingly, the throughput per node n is

$$x_n^{\text{CSMA-CA}} = \frac{S}{N} = \frac{p_s p_{\text{tr}} L}{p_{\text{tr}} N \mu + \frac{W+1}{2\rho}}. \tag{6.8}$$

6.1.2 TDMA

The derivation is divided into two parts. In the **first part**, TDMA has synchronized time slots. I assume a pre-planned TMDA scheme where N nodes take turns to send a packet. Each node uses a time slot of fixed length in a TDMA frame comprising N slots, separated by N guard intervals (Figure 6.2).

The application requirements set the upper bound on the guard interval ϵ. As-

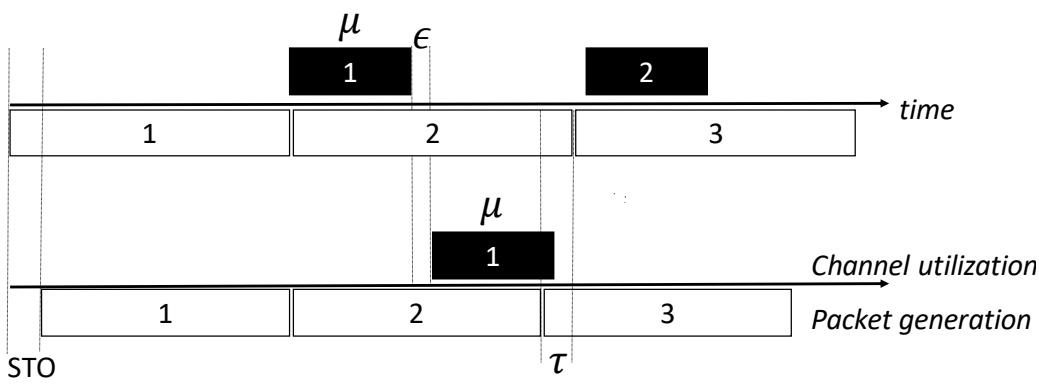


Figure 6.2: Channel access with TDMA

sume that after audio synchronization the remaining SRO is vanishingly small and thus all microphone signals are sampled at a rate f_s (samples/s). The samples are encoded into bits using a code rate of C(bits per sample). Given that the time required to transmit a packet is μ, then the time when a node is ready to send its next packet after every other node has sent a packet and waited a guard interval is t_{send}=$N(\mu+\epsilon)$, while the time needed to generate a packet of size L is t_{pkt}=$\frac{L}{Cf_s}$. Ideally to avoid queuing delays, we would require

$$\underbrace{N(\mu+\epsilon)}_{t_{\text{send}}} \leq \rho \underbrace{\frac{L}{Cf_s^*}}_{t_{\text{pkt}}}, \tag{6.9}$$

where f_s^* is the highest expected sampling frequency across all microphones, depending on the SRO accuracy, and $\rho = 1$ if $t_{\text{pkt}} = t_{\text{send}}$. Given that R=L/μ and solving (6.9) for ϵ it holds:

$$0 \leq \epsilon \leq \rho L\left(\frac{1}{NCf_s^*} - \frac{1}{R}\right). \tag{6.10}$$

As long as ϵ is smaller than this value, the TDMA scheme satisfies the application data rate requirement. Meanwhile, the acoustic application needs to provide a clock synchronization whose remaining SRO-induced shift τ is less than ϵ; otherwise, there are unnecessary queuing delays.

The **second part** is concerned with TDMA throughput with varying accuracy. The challenge is that the accuracy of the synchronization of the acoustic application varies randomly [GSH21]. Therefore, sometimes packets are lost because the guard

interval was chosen too small (clocks are too inaccurate), or there are large delays because the guard interval was chosen too large, wasting throughput. Additionally, when load is too low, a node has to wait $(1-\rho)t_{\text{pkt}}$ for a packet to be ready to be sent. Consequently, from (6.9) the throughput in TDMA is given by:

$$\begin{aligned} x_n^{\text{TDMA}} &= \frac{L}{t_{\text{send}} + (1-\rho)t_{\text{t}_\text{p}\text{kt}}} \\ &= \frac{R/N}{1 + \frac{\epsilon}{\mu} + \frac{(1-\rho)t_{\text{pkt}}}{N\mu}} p(|\tau| < \epsilon + (1-\rho)t_{\text{pkt}}). \end{aligned} \tag{6.11}$$

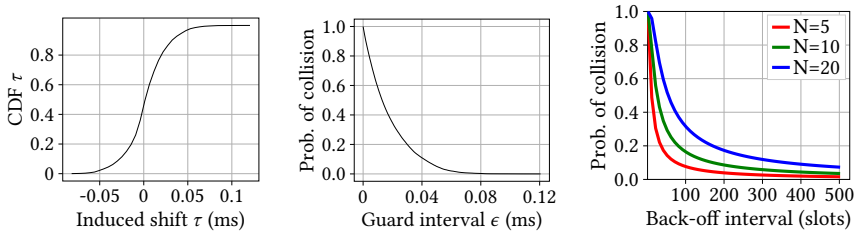
Without training the network, the probability of a successful transmission $p(|\tau| < \epsilon + (1-\rho)t_{\text{pkt}})$ will typically not be available inside a system, hence, it is difficult to find a scheme that adapts the guard interval. But it is easy to evaluate, for an empirically collected distribution of synchronization errors, which impact a fixed guard interval length would have, as shown in the following sections.

6.2 Environment setup

I look into two aspects: 1) obtaining a distribution of synchronization errors based on the simulation of an acoustic application and 2) using this distribution to derive a TDMA-based throughput (and compare it to CSMA/CA throughput). As be would the case in reality as well, I separate out these two aspects in separate simulation systems.

Regarding the first part, I briefly describe the wireless acoustic network setup for getting the synchronization results and refer to [GSH21] for more details. In total, 100 different acoustic sensor networks were simulated, each having different node positions. For each network, the recordings of an acoustic scene are simulated with a single acoustic source being active at any given time. Hereby the source positions vary over time. Such a scene is 5 min long and may contain speech pauses. The audio signals were re-sampled to simulate a time-varying SRO. In total, there are more than 11 million data points of SROs estimation error and the corresponding induced shift τ, from which I derive the empirical cumulative distribution function of the synchronization error (Figure 6.3 (a)).

Regarding the second part of the simulation, I fix the nominal sampling frequency f_s=16 kHz, the coding scheme C=32 bits/sample and the packet length to L=8 kbit. The duration of back-off slot is 20μs. In the following sections, I evaluate the throughput of TDMA and CSMA/CA while varying the number of nodes in



(a) CDF of induced shift **(b)** TDMA probability of collision **(c)** CSMA/CA probability of collision

Figure 6.3: Probability of collision distribution for TDMA and CSMA

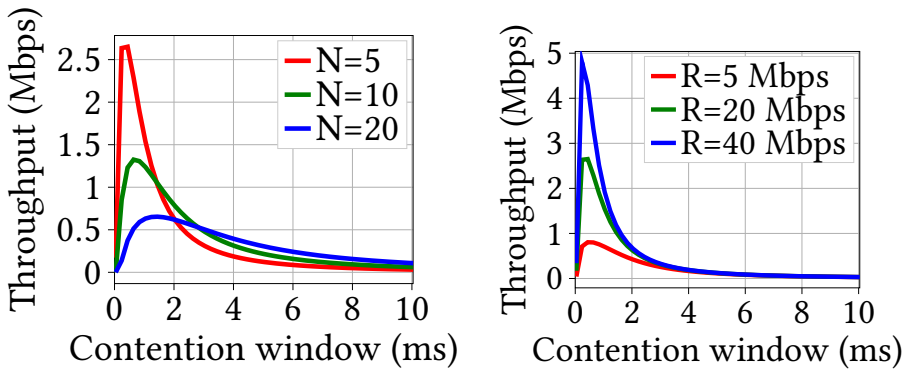
the network and the data rate used for transmission. For CSMA/CA, I use the length of the contention window as a parameter; for TDMA, I use the guard interval length. All throughput results are based on the derivations in Section 6.1. Additionally, I validated the throughput results of CSMA/CA with the data in [Bia00; LK13]. I highlight again that algorithms used for microphone synchronization were developed by my colleagues Tobias Gburrek and Joerg Schmalenstroeer, so I do not discuss these algorithms here.

6.3 Results

In the following sections, I first show the probability of collisions for both TDMA (due to incorrect synchronization) and CSMA/CA protocols. Based on that, I estimate their average throughput per node and compare those. At the end, I show the throughput when there is the same low-to-medium traffic load among all nodes.

6.3.1 Probability of collision

Figure 6.3 (a) visualizes the cumulative distribution function (CDF) of the SRO-induced shift τ that remains after audio synchronization; it was given from the results of [GSH21]. Figures 6.3 (b) and 6.3 (c) depict the probability of collisions for TDMA (when changing the guard interval) and CSMA/CA (when changing the number of nodes and the contention window), respectively. On the one hand, I show the results of CSMA/CA for the sake of convenience to demonstrate how changing the contention window size and number of nodes impact the probability of collision. These results will be used in the next sections when estimating the throughput. On the other hand, and more importantly, it is observed that the probability of collision of TDMA depends solely on the SRO estimation accuracy



(a) $R = 20$Mbps **(b)** $N = 5$

Figure 6.4: CSMA/CA throughput for varying number of nodes and transmission rate; contention window size as a parameter

of the acoustic application and, unlike CSMA/CA, is independent of the number of nodes in the network. Also, it is clear how increasing the length of the guard interval decreases the probability of collisions.

6.3.2 CSMA/CA throughput

Figure 6.4 shows the throughput of CSMA/CA in different parameter combinations. First, we observe that for many nodes ($N = 20$) in Figure 6.4 (a) or low data rates ($R = 5$ Mega bits per second (Mbps)) in Figure 6.4 (b) shows low throughput. But for lower numbers of nodes (Figure 6.4 (a)) or higher data rates (Figure 6.4 (b)) the throughput becomes higher, but more sensitive to the contention window. In particular, for large contention windows, different data rates have similar throughput (Figure 6.4 (b)). Hence, the contention window size in CSMA/CA needs to be tuned carefully; otherwise, the achieved throughput drops steeply, especially for low numbers of nodes and high data rates.

6.3.3 TDMA throughput

I repeat the previous parameter settings for data rates and number of nodes, but I show the throughput using TDMA, where the guard interval is changed instead of the contention window (Figure 6.5). Additionally, I integrate the probability of collision from Figure 6.3 (b) for a convenient translation between throughput and probability of collision, when $\rho = 1$. These probabilities can be used as a threshold for switching between TDMA and CSMA/CA for a given throughput;

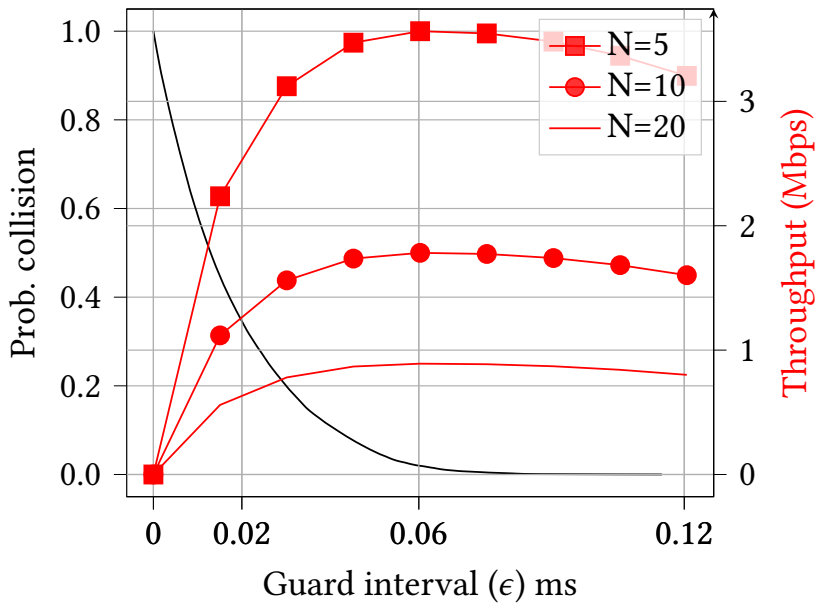
when the resulting probability of collision is higher than the expected one, fall back to CSMA/CA to get a better synchronization and consequently a better estimation of the guard interval. I observe the throughput when changing the number of nodes (Figure 6.5 (a)) and changing the data rates (Figure 6.5 (b)), similar to CSMA/CA, the throughput is low for a large number of nodes and small data rates and is barely sensitive to the guard interval of TDMA.

An interesting fact observed in Figure 6.5, beside not having the steep drop-down behavior, is that two different guard intervals, with corresponding different probability of collisions, will achieve in many cases the same throughput. Although delay is not a metric in the analysis here, this may be useful for an application developer when setting up the network. Hereby, the target could either be to achieve a certain throughput with a small guard interval and high probability of collisions but low delay, or with a large guard interval, small probability of collisions and high delays. This of course also depends on other factors such as how the application handles packet losses, whether re-transmissions are necessary and if there is a maximum queuing delay.

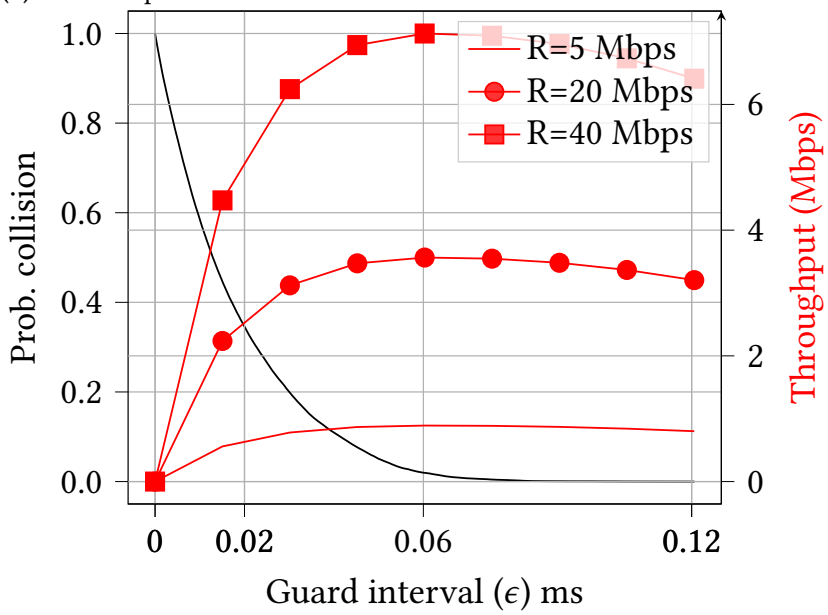
6.3.4 Comparison of CSMA/CA and TDMA throughput

I compare the throughput of CSMA/CA (while changing the contention window) with TDMA (while selecting the guard interval with highest throughput ϵ_{opt} and the corresponding maximum threshold ϵ_{th} from (6.10)). For low data rates and many nodes (Figure 6.6 (a)) CSMA/CA has lower throughput compared to TDMA. For high data rates and small numbers of nodes (Figure 6.6 (b)), CSMA/CA has a higher throughput than TDMA with threshold guard interval but only over a narrow contention window interval. Additionally, the throughput of CSMA/CA decreases fast when increasing the size of the contention window, to be even lower than the throughput of TDMA using the threshold guard interval ϵ_{th}, which is the lowest recommended TDMA throughput.

It is worth noting that TDMA using the optimal guard interval ϵ_{opt} has in both scenarios higher throughput than CSMA/CA. Consequently, TDMA, with ϵ_{opt}, offers a stable behavior (with respect to the number of nodes) and higher throughput than CSMA/CA, while other values may yield in some cases (Figure 6.6 (b)) a lower throughput. Such properties are important for acoustic applications and multimedia ones in general.

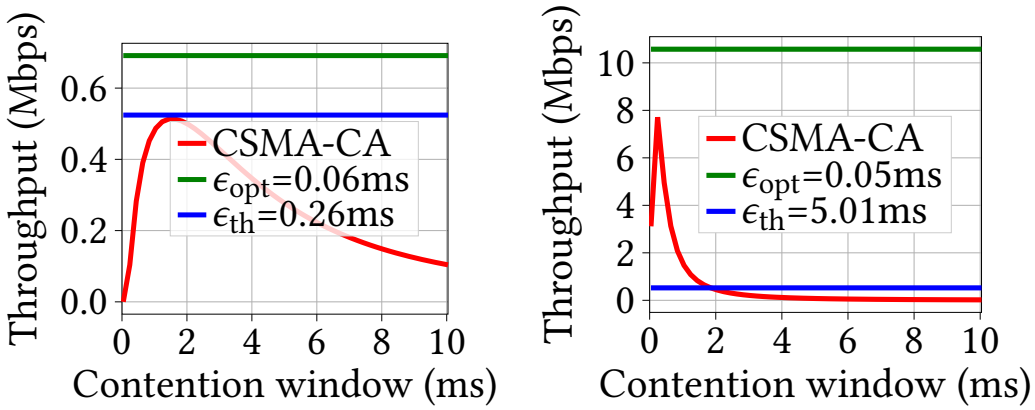


(a) R = 20Mbps



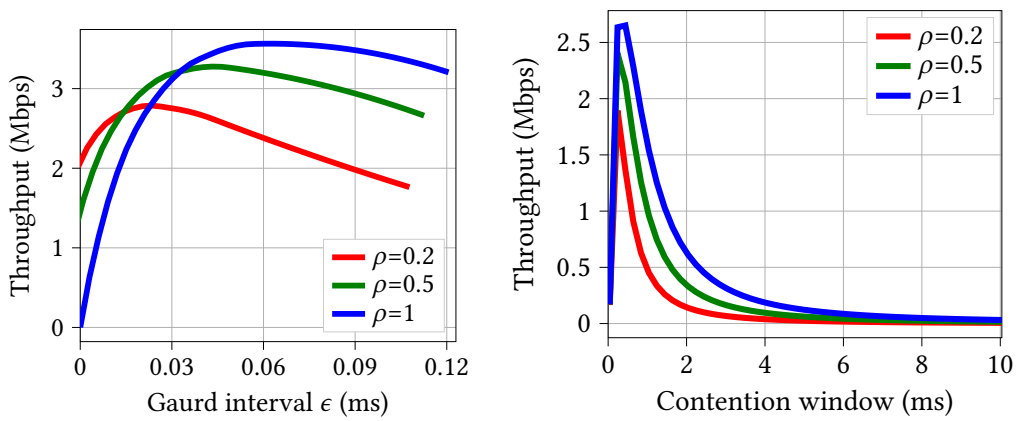
(b) $N = 5$

Figure 6.5: TDMA throughput for varying number of nodes and transmission rate; guard interval length as a parameter



(a) R = 15Mbps –N = 20 **(b)** R = 40Mbps – N = 3

Figure 6.6: TDMA vs CSMA for throughput per node



(a) TDMA **(b)** CSMA/CA

Figure 6.7: Unsaturated nodes for R = 20 Mbps and N=5

6.3.5 Throughput for unsaturated nodes

As mentioned earlier, acoustic applications may have moments of silence (i.e., pauses) where a node is not sending, although the channel is idle. Here, I assume that all nodes have the same traffic load ρ for both TDMA (Figure 6.7 (a)) and CSMA/CA (Figure 6.7 (b)), fixing the number of nodes ($N = 5$) and the data rate ($R = 20\,\text{Mbps}$). An interesting observation is that when TDMA has small guard intervals, a network with low load (e.g., $\rho = 0.2$) tends to have higher throughput than highly load ones (e.g., $\rho = 1$). Since the channel is idle for some time and the nodes have no data to send for some time, this time compensates for the induced shift, decreasing the probability of collision. As the guard interval increases, the impact of the time without having data to send decreases and the impact of the guard interval becomes more dominant: the higher the network load, the higher the throughput.

Similar to TDMA, and as mentioned in [LK13], increasing ρ in CSMA/CA increases the throughput. Yet, I highlight that the throughput for all ρ is higher for TDMA than for CSMA/CA. This is another advantage of using TDMA in wireless multimedia applications where ρ may change.

6.4 Summary

In this chapter, I use the synchronization achieved by wireless multimedia applications to synchronize wireless transmissions. Accordingly, I derive the throughput of TDMA resulting from the synchronization error of the audio stream synchronization and compare it to CSMA/CA, when nodes operate in a single collision domain. A major advantage of the TDMA approach over CSMA/CA is the stability, because the probability of collision is independent of the number of wireless nodes, as well as its higher throughput.

To use the synchronization from the application for TDMA, first a minimum throughput (or other metrics like probability of collision) need to be defined. Accordingly either TDMA or CSMA/CA can be used. If CSMA/CA is used due to to bad synchronization, the application can later switch to TDMA after synchronizing the clocks and optimizing the guard interval, for a higher throughput. It is worth noting that such MAC systems will be application proprietary; TDMA will always need an application running synchronization.

Part III

Subset Selection of Jobs, Nodes and Moves

Admission control is not the only way to handle resource intensive requests; in some cases jobs within a request could be optional, so not all jobs have to be allocated. In [Gün+19], I use an acoustic application as a case study where some of the jobs are optional to allocate. Consequently, a request consists of many optional jobs and the objective is to select at maximum k jobs while having an upper bound on the allocated time slots for transmissions.

Another way to control the load of an application is to control the amount of data being transmitted/forwarded. In [Gun+21], a speaker is assumed to be stationary, where only the best k microphones are selected to maximize the acoustic quality at low network cost. Similarly, my work in [Afi+21; ARK21a] solves a similar problem using RL, yet for a moving speaker, while using different utilities of the acoustic application. Moreover, I add an objective in [Afi+21] to decrease the frequency of changing the selected microphone set.

So far I have assumed that the microphones are stationary, yet moving the microphones adds a degree of freedom that can improve the performance, as seen in theatres and stadiums. In [ARK21b], I show how the performance enhances when using autonomous vehicles pre-installed with microphones compared to using static microphones. Additionally, I propose a centralized single-agent solution that controls the movements of the microphones. Then, I show how it performs better than a greedy heuristic that simply moves all the microphones towards the active speaker.

Because the training of a single agent does not scale up easily when increasing the number of microphones to control, I propose two different multi-agent solutions; with either shared or separate policies. The shared policy uses a the same trained model by all agents. In the separate one, each agent independently trains its model. The performance of this separate model is compared to the single agent as well as to the heuristic; it is found to be close to the single agent and better than the heuristic. Moreover, I show theoretically how the speed of the vehicles and the speaking time of the speaker impact the learning process of the RL agent.

7 Resource Allocation for Optional Jobs

Typical research in WSN applications optimizes the application's output while focusing solely on its parameters [Cob+17; JSH12].In acoustic signal processing, this could be the acoustic Signal-to-Noise Ratio (SNR) or word error rate for speech enhancement. Similarly, optimizing the WSN itself focuses on the network's aspects (such as data rates and delays) while considering arbitrary parameters of the application, hoping it will serve any application to its best. In this chapter, I consider cross-layer optimization where both application (in this work, an acoustic application) and network aspects are jointly considered. The results in this chapter are based on the joint work in [Gün+19], yet I shall describe here only my contribution.

Besides the benefits mentioned in Section 1.1 of distributing signal processing application, an additional benefit is to optimize the application itself. Let us consider an application graph with many jobs, while some of these jobs are optional to run. Ideally with unlimited resources, running all the jobs may yield the best performance, but if there are not enough resources or tight delay constrains, it is beneficial to ignore some of these jobs, while still having the algorithm running.

Consequently, in addition to the research questions regarding node selection, routing and time slot allocation (Chapter 3), two more questions arise:

- Which jobs should be selected to maximize the quality?
- What is the impact of dropping the jobs compared to running them all?

In this context, TRIple-N-Independent component analysis for CONvolutive mixtures (TRINICON) [BBK18], an acoustic signal processing algorithm for Blind Source Separation (BSS), is distributed inside a WASN. Choosing TRINICON here is only to have a representative application with many jobs and it is interchangeable with other algorithms such as Independent Component Analysis (ICA) [MI98] and Independent Vector Analysis (IVA) [Kim+07].

Solving wireless VNE with optional jobs in the application graph has not been investigated before. Nevertheless, scaling the jobs (by placing a job more than once) is very similar to having optional jobs, but it has been considered only in wired networks [DSK18]. The main difference between job scaling and selecting optional jobs, the latter has a given application graph whose topology may change

by dropping some jobs, while the former changes its topology in a predefined manner by adding more jobs. The authors is [Dek+22] considered cross-layer optimization between wireless networks and acoustic application but they have focused only on energy consumption, here I focus on network delay. Additionally, work that focused on the wireless communication cost and microphone selection (i.e., also cross-layer optimization) will be discussed later (Chapter 8).

7.1 Problem Formulation

The *application graph* consists of obligatory jobs P' that have to be placed and optional P jobs that may or may not be placed. Similar to Chapter 3, all jobs have processing requirements and the links between the jobs describe the data rate requirement. The *infrastructure graph* is defined the same as in Chapter 3; a group of wireless nodes $v \in V$ each of which has a processing capacity c_v and the wireless channel is divided into T time slots.

In contrast to the formulation in Chapter 3, here I add and modify constraints, and I have a different objective. Let us recall the binary variables $\theta(p,v)$, which indicate that job $p \in P$ is running on network node $v \in V$, and the binary variable $\delta(t)$, which indicate whether any node is transmitting in a time slot t. The processing of the jobs requires that all calculations, per job, are finished and transmitted within some time duration to avoid dropping samples.

I still assume that the time of computation is negligible and the delay is introduced mainly due to the transmission delay. Thus, the resulting upper bound on the allowable end-to-end delay is determined primarily by the application (here the sampling rate and the number of samples are used for real-time effect), which is reflected by the newly-added constraint Equation (7.1) that limits the number of utilized transmission time slots (i.e., scheduling time frame) to a maximum number of time slots $T_{\max}$:

$$\sum_t \delta(t) \leq T_{\max}. \tag{7.1}$$

While all jobs have to be distributed in Chapter 3, I relax this restriction to allow partial distribution of the jobs in P by replacing the original constraint Eq. (3.5) with the modified one Eq. Equation (7.2). If necessary, jobs can be discarded, particularly to satisfy Equation (7.1), while still ensuring that no job is placed multiple times.

$$\sum_{v \in V} \theta(p,v) = 1, \forall p \in P \quad \rightsquigarrow \quad \sum_{v \in V} \theta(p,v) \leq 1, \forall p \in P \tag{7.2}$$

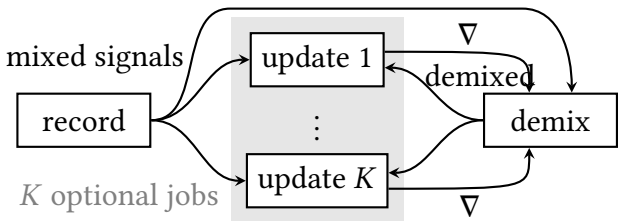


Figure 7.1: Multicast-Aware Routing for Virtual network Embedding with Loops in Overlays (MARVELO) application graph for distributed TRINICON.

Note that for the obligatory jobs P', the old constraint still holds

$$\sum_{v \in V} \theta(p, v) = 1, \forall p \in P' \tag{7.3}$$

Last but not least, the objective is changed. Instead of minimizing the end-to-end delay, i.e., the number of used time slots used for transmission, the modified problem aims to maximize the number of running jobs:

$$\min \sum_{t \in T} \delta(t) \quad \rightsquigarrow \quad \max \sum_{\substack{p \in P \\ v \in V}} \theta(p, v). \tag{7.4}$$

I assume in this formulation that all jobs have the same resource requirements, which is actually the case as in [Gun+21], but they do not have the same contribution. An important part of the solution is to determine the contribution of each of the optional jobs, yet I do not discuss this here, since it is not a part of my contribution and I assume the contribution per job is given. Accordingly, it is enough to abstract the quality as the number of running jobs, instead of summing their contribution, especially when calculating this contribution has a large simulation overhead. For other applications, a natural extension would be to assume different requirements with different contribution for the jobs, yet I focus here on the application at hand, without adding further complexities.

7.2 Simulation Setup

The *application graph* consists of two obligatory jobs, *record* and *demix*, and K *update* optional jobs. The record job produces blocks of audio signals for simultaneously active speakers. The demix job is used for demixing multiple active speakers, using a linear convolution on the input recorded signal blocks and a demixing filter of

Table 7.1: Network parameters.

Parameter	Value
R	2.048Mbps
$t_{\max}$	128ms
N_0	-82 dBm
SINR_{th}	7 dB
$T_{\max}$	8 slots

length H [BAK04a]. The signal blocks are further divided into K signals (let us call them offline signals), so that the *update* jobs calculate the *offline gradient* between a shift of the input signal and the kth offline signal where $k \in K$. The gradients are then forwarded to the demix job to update its demixing weights, which are sent back to the *update* jobs (Figure 7.1). Up to $P = 8$ *update* jobs can be placed.

I simulate a WASN in a room of dimensions $25m \times 25m$ and use its infrastructure graph as embedding target for the TRINICON application graph in Figure 7.1, with different numbers of nodes, where each node can only run one job simultaneously. The minimum payload rate for inter-node communication is given by

$$R = P_{\text{src}} \cdot S_{\text{type}} \cdot f_{\text{s}} = 2 \cdot 64 \cdot 16000\ bps = 2.048Mbps, \tag{7.5}$$

where S_{type} denotes the width of the underlying data type in bits, $f_{\text{s}} = 16kHz$ represents the sampling frequency and P_{src} is the number of microphones. The maximum end-to-end delay due to block processing for a convoluted signal of length $2H = 2048$ is

$$t_{\max} = \frac{2H}{f_{\text{s}}} = \frac{2048}{16000}s = 128ms, \tag{7.6}$$

where H is the demixing filter length[3].

The three remaining parameters in Table 7.1, the transmission noise floor N_0, the minimum required SINR for successful wireless transmission SINR_{th} and the maximum number of transmission time slots $T_{\max}$, are configured with respect to the IEEE 802.11 standard [19] to meet the specified values for R and $t_{\max}$.

3 I ignore packet overheads

Table 7.2: ΔSIR in decibels, averaged over channels and scenarios.

Selection scheme	$K' = 1$	$K' = 2$	$K' = 3$	$K' = 5$
Reference	**6.64**	**6.64**	**6.64**	6.64
Recent	4.55	5.38	5.87	6.29
Random	4.66	5.49	5.96	6.40
Proposed	5.27	6.20	6.56	**6.93**

7.3 Results for Optional Block Selection

The proposed formulation in Section 7.1 outputs the maximum number of optional jobs that can be placed K', but it does not define the subset of jobs that should be selected. In this application, the optional jobs are the gradient jobs, which are ranked based on the magnitude-squared coherence [Gün+19] and the highly ranked ones are selected.

To demonstrate the efficacy of the proposed jobs selection, three baselines are used for comparison:

- **Reference**, where no optional jobs are discarded and the network impact is ignored, hence its performance is independent of K',
- **Recent**, where the K' most recent *update* jobs are selected,
- **Random**, where K' update jobs are randomly selected.

Three pairs of clean speech signals of different gender and speaker age sampled at 16kHz with a duration of 60s are convolved with room impulse responses measured in a low-reverberant chamber and superimposed with additive white Gaussian noise to obtain an acoustic SNR of $20dB$ at two microphones spaced $21cm$ apart. Two speakers are active in 20 scenarios with different combinations of speaker positions. More details about the acoustic parameters are found in [Gün+19]

The demixing performance of TRINICON is quantified using the acoustic logarithmic Signal-to-Interference Ratio (SIR). In each output channel, the extracted speaker acts as the target while the suppressed speaker acts as the interferer. Thus, the SIRs improvement ΔSIR is the difference between the SIRs of the output and input signals, averaged over all scenarios, to yield the values depicted in Table 7.2.

As expected, the algorithmic performance deteriorates with a decreasing number of selected optional jobs K'. Nevertheless, the proposed method outperforms the two baseline methods: random and recent, consistently for all investigated different

Table 7.3: Job distribution results.

#Nodes	#Jobs (= K')	#Slots
4	3	5
6	4	7
8	6	8
9	6	8
10	6	8

number of selected jobs and, due to its ability to discard harmful *update* jobs (jobs with negative contribution due to adding harmful gradients [Gun+21]), $K' = 5$ even outperforms the reference method which exploits all update jobs. Thus, even when sufficient computational power is available, sparse adaptation using subset of the optional jobs does not only save computational power but also improves the performance.

Table 7.3 shows the distribution results in terms of distributed optional update jobs (recall Figure 7.1) and used transmission time slots for different numbers of network nodes. Two limiting factors can be identified. On the one hand, the limited computation capacity of the network nodes only allows distributing 3 and 4 jobs for 4 and 6 nodes, respectively. On the other hand, when increasing the number of nodes above 8, no more than $K' = 6$ of the $P = 8$ available jobs can be distributed due to the end-to-end delay constraint, irrespective of the available number of nodes.

7.4 Summary

In this chapter, the formulation in Chapter 3 is modified to be used for speaker separation, using TRINICON, while the objective is to maximize the number of optional jobs that could be placed.

The job placement results demonstrate the limiting factors of WASN and their implications on an exemplary acoustic signal processing scheme. Additionally, the simulation results show an improved separation performance even when selecting only a subset of optional jobs and not all jobs as it was expected, achieving higher performance at lower computation load. I focused on one example of acoustic application that optimizes application's quality under network constraints, which can be applied to different application with similar properties. In next chapters, I investigate how to jointly optimize both the application and network qualities.

8 Sensor Selection in Acoustic Sensor Networks

In the previous chapters, I assumed that the source nodes (e.g., microphones) are always given. But this does not have to be the case; there may be the opportunity to select a subset of microphones that results in the best quality and at the same time has load on the network.

For example, let us consider a scenario based on hybrid learning [ZB13]; classes take place in-person with a limited number of attendees while others join the classes virtually at the same time. Such scenarios require a camera and multiple wireless microphones distributed inside the classroom.

A naive solution would select all available microphones for recording, but this might over-utilize the wireless network leading to high latency and packet losses. Another solution would activate N microphones with the best signal quality, but they do not necessarily have good wireless connectivity, which again results in problems of latency and packet losses.

In Chapter 7, I showed that selecting a subset of optional jobs can actually improve performance. Similarly, selecting a subset of sensors (in this case, microphones) can also be better than selecting all microphones:

1. It decreases the load on the wireless network for transmitting audio signals

2. Some signals may be noisy (e.g., microphones placed close to a fan or a window) which decreases the overall quality of the received audio signal.

When selecting a subset of microphones, again, two metrics should be considered: Quuality of Service (QoI) and QoS. The former is used to describe the quality of the collected data (e.g., accuracy and SNR), while the latter describes the quality of how the data is delivered (e.g., delay and packet losses) [ASV15; SBB13].

Both QoI and QoS are important factors for end users' the overall Quality of Experience (QoE); there is a lot of work that estimates and maximizes this QoE [Sko+18]. Nevertheless, it is necessary here to separate between both QoS and QoI for the application developer, because the sensed data are not the final results; after the data are collected, they are further processed, hence, QoI should be investigated as a separate metric in decision making.

For example, when audio data are collected from microphones and further processed to classify speech or to localize speakers, based on the quality of the sensed

data, different algorithms can be used with different complexity and performance. Increasing the number of microphones may enhance the QoI and the performance of the algorithm. But for that, the data must be delivered within a bounded time in case of real-time applications or with low packet losses for high reliability. Selecting more and more microphones for data collection will increase the load on the wireless network, causing unwanted delays, which decreases the QoS. Thus, a developer can define the QoE with respect to the application itself, e.g., by setting a minimum required QoI, while maximizing the QoS, or vice versa.

In order to jointly solve for these metrics, let us consider each one separately. On the one hand, choosing a subset of microphones to increase the QoI is thought to be a submodular problem [Bac10; SBV10]. In other words, increasing the subset size of selected microphones increases the QoI, yet with a diminishing return; at some point, the rate of gain when increasing the subset size decreases. On the other hand, choosing a subset of microphones for optimizing QoS is a supermodular problem, so that all microphones need to coordinate together who is going to be active, otherwise they overload the wireless channel and decreases the QoS. In case of wireless networks, two unequal subsets of wireless microphones can have different QoS, even if they have the same size. This is due to the fact the QoS is directly proportional with the wireless signal quality of the microphones and not only the number of transmitting microphones.

The results shown in this chapter are from the joint work in [Afi+21; ARK21a; Gun+21]. Since I focus here only on my results, I note that the QoI problem alone is not always submodular [Gun+21], but it is still convex. In fact, it showed skew-supermodular behaviour [FK09]. This means that there exists one or more subsets (I shall call them parity subsets), when increasing the number of microphones within these subsets, the QoI decreases. However, there exists other subsets that intersects with the parity subsets, increasing the number of microphones of these subsets still increases the QoI.

Most of the work that studied QoI exploits all available microphones for data processing [NM18; WW19]. When it comes to subset selection, some generic approaches have been proposed [CL13; JB09] to optimize QoI using convex and non-linear optimization. When focussing on WASN, there is several work related to synchronization [CG17b] and beamforming [ZH14]. However, they ignore the network aspect.

The work in [Zha+18; ZHH18] selects the most useful microphones for beamformer-based noise reduction, while minimizing the subset's size to save energy. Similarly, the authors in [CKS21] select the most useful microphones for speech enhancement while minimizing the subset size to decrease communication cost. The communication cost is a simple model, based only on the number of nodes. In contrast to

related work, my network cost model for QoS is based on the SINR which is more generic, so that the microphones do not have to be equidistant to the receiver and they do not need to be all in the same collision domain. Hence, I treat QoS as a submodular problem and not just linear in the number of microphones.

In this chapter, I look into solving the problem when the speakers are stationary (Section 8.1) and when they are moving (Sections 8.2 and 8.3).

8.1 Microphone Selection for Stationary Speakers

The motivating scenario here is a meeting room where the speaker is sitting (i.e., not moving). One objective is to find a set of microphones to activate for recording the meeting. More importantly, I investigate the relation between QoS and QoI and how it impacts the set selection, e.g., how many microphones within a set and which ones should be activated.

8.1.1 Network Cost Function

Recall from Section 3.1 the definition of application and infrastructure graphs. Here, the application graph consists of up to $\|V\|$ instances of a job p_{src}, responsible for recording the microphone signals, and one job p_{sink} at the central vertex of the graph that collects the captured data and performs further processing.

The infrastructure graph captures how the wireless microphones transfer their data to the access point. Specifically, the wireless link from node v to the access point is characterized by the radio attenuation $\gamma_v, \forall v \in V$, while the transmission noise floor N_0 is identical for all nodes. The communication between the wireless nodes and the access point adheres to the IEEE 802.11 standards [21]. In general, it relies on L_{SC} sub-carriers for transmitting the modulated signal, Modulation and Coding Scheme (MCS) for setting the number of bits per symbol,number L_{M} of data streams using a MIMO antenna and the symbol time interval T_{SI} for determining the time duration per symbol.

Assuming that all nodes within the selected subset S have the same transmit power of one Watt, then recall the wireless SINR is

$$\text{SINR}(v) = \frac{\gamma_v}{N_0 + \sum_{\substack{u \in S \\ u \neq v}} \gamma_u}. \tag{8.1}$$

Depending on $\text{SINR}(v)$, different MCS can be employed, which in turn affect the achievable data rate. An example of the MCS table is shown in Table 8.1, where

$\mathrm{SINR_{th}}$	2	5	9	11	15
MCS	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3

Table 8.1: An exemplary MCS table with bandwidth 20MHz [21]

$\mathrm{SINR_{th}}$ represents the minimum required SINR to use the corresponding MCS. Then, the wireless data rate between node v and the access point is

$$r_v = \frac{L_{\mathrm{SC}} \cdot L_{\mathrm{M}} \cdot \mathrm{MCS}(\mathrm{SINR}(v))}{T_{\mathrm{SI}}}. \tag{8.2}$$

Consequently, the transmission delay $\tau(S)$ when selecting the channels in S is given by

$$\tau(S) = \sum_{v \in S} \frac{L_{\mathrm{b}}}{r_v}, \tag{8.3}$$

where L_{b} is the number of bits per sample. For Equation (8.3) to hold, two constraints must be satisfied:

$$\sum_{v \in V} \theta(v) \geq 1, \qquad \forall v \in V, \tag{8.4}$$

$$\mathrm{SINR}(v) \geq \mathrm{SINR_{th}}, \qquad \forall v \in S, \tag{8.5}$$

where $\theta(v)$ is a binary placement variable that indicates that an instance of block p_{src} runs on node v. In other words, $S \subseteq V \ni \theta(v) = 1, \quad \forall v \in S$. As the microphone selection pertains only to the blocks p_{src}, block p_{sink} always runs on the gateway. Equation (8.4) ensures that $S \neq \emptyset$ and Equation (8.5) ensures that the wireless transmissions are (semi-) collision free.

Finally, I define the network cost function $\mathcal{J}_{\mathrm{n}}(S)$ as

$$\mathcal{J}_{\mathrm{n}}(S) = 1 - \left(\frac{1}{\tau(S)}\right)^{\eta} \tag{8.6}$$

where $\eta \geq 0$ is a user-specified parameter capturing the sensitivity of an acoustic application to the network delay; small values of η indicate low sensitivity and vice versa.

8.1.2 Joint Optimization

Now let us assume there is another cost function $\mathcal{J}_{\mathrm{a}}(S)$ that mimics the QoI, in this case the acoustic quality. In [Gun+21], there are detailed steps on how to calculate such an acoustic cost function. There is a fundamental trade-off between acoustic and network cost. By emphasizing one of the two aspects, the obtained solutions offer different optimal trade-offs and form a Pareto front [MGS09]. Therefore, I select the joint cost function as

$$\mathcal{J}(S, \alpha) = \min\big(\{\alpha \mathcal{J}_{\mathrm{a}}(S), (1-\alpha)\mathcal{J}_{\mathrm{n}}(S)\}\big) \tag{8.7}$$

with $0 \leq \alpha \leq 1$ controlling the relative weight of the individual cost functions. I consider α to be time-invariant as long as the application does not change. For online processing, α is controlled by the subsequent signal processing algorithm, e.g., a speech enhancement algorithm could emphasize acoustic cost to increase the speech quality. The optimal solution

$$S_{\mathrm{opt}}(\alpha) = \arg\min_{S} \mathcal{J}(S, \alpha) \tag{8.8}$$

is found offline by an exhaustive search over all $2^{\|V\|} - 1$ admissible S. Although I have chosen Equations (8.7) and (8.8) to emphasize the trade-off between the acoustic and network domains, $\mathcal{J}_{\mathrm{a}}(S)$ and $\mathcal{J}_{\mathrm{n}}(S)$ will be also later incorporated into Equation (8.8) as constraints to ensure a minimum signal quality or maximum network utilization (Section 8.2).

8.1.3 Experimental Evaluation

In this section, I show the efficacy of the proposed selection scheme while considering practical acoustic conditions using recorded speech signals. I show empirically that the proposed microphone-set selection S_{opt} outperforms the majority of possible choices with respect to the latency performance for an exemplary practical signal processing algorithm. Note that the acoustic cost is generic and was not specifically optimized for the acoustic application (in this case, Signal-to-Distortion Ratio (SDR) obtained from BSS [FGV05]). I demonstrate how the parameter α impacts the trade-off between acoustic and network cost and how this translates to the ground-truth (i.e., SDR and delay) performance measures.

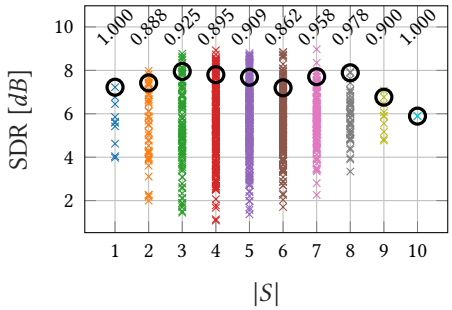


Figure 8.1: SDR for admissible S, grouped by cardinality $|S|$. Utility-based $S_u(|S|)$ outperforms majority of S.

Environment Setup

The experimental setup is illustrated in Figure 8.4. $\|V\| = 10$ microphones, clustered in five pairs with inter-microphone distance $4cm$, are placed in a quarter circle of $2m$ radius surrounding the loudspeaker/speaker inside a room of dimensions $6.26m \times 4.86m \times 3m$. All microphones face the loudspeaker and with sampling frequency $f_s = 16kHz$ via a common clock, which allows computing the ground-truth performance measures. To replicate a realistic scenario with physical obstacles, the first two arrays, representing microphones 1–4, have their line-of-sight to the loudspeaker blocked by a solid object $1m$ wide and $2m$ high, hence suppressing the line-of-sight contribution in the corresponding microphone signal. The loudspeaker signal consists of $22s$ of adult male speech.

The delay sensitivity of the signal processing application is set to $\eta = 0.4$ while the radio noise floor is $\frac{N_0}{\text{transmission power}} = 10^{-4}$. I restrict the calculations to 20MHz wireless bandwidth, i.e., $L_{SC} = 48$ sub-carriers, with symbol time interval $T_{SI} = 4\mu s$ and one omni-directional antenna ($L_M = 1$)[4]. For sample encoding, I set $L_b = 8$.

Figure 8.1 shows the efficacy of the acoustic utility measure proposed in [Gun+21], without considering the network aspects. Therein, the SDR for all admissible S are shown, grouped by the number of selected microphones. For each $|S|$, the corresponding utility-based selection $S_u(|S|)$ is highlighted by black circles, while the numbers above each point cloud indicate the fraction of combinations with the same number of microphones, while not having a higher SDR than $S_u(|S|)$. Again, recall that the acoustic utility is a generic utility and was not specifically designed for SDR. Nevertheless, as shown, calculating the acoustic cost can still

4 These values are needed to look up the modulation and coding scheme in [21]

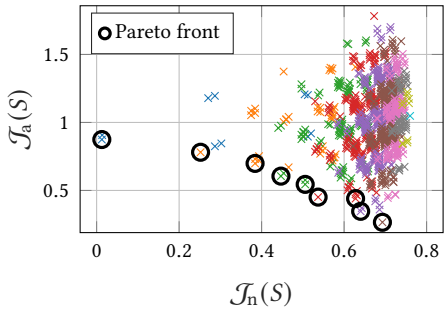


Figure 8.2: Acoustic $\mathcal{J}_a(S)$ vs. network $\mathcal{J}_n(S)$ cost for all admissible S. Pareto front shows trade-off between acoustic and network cost.

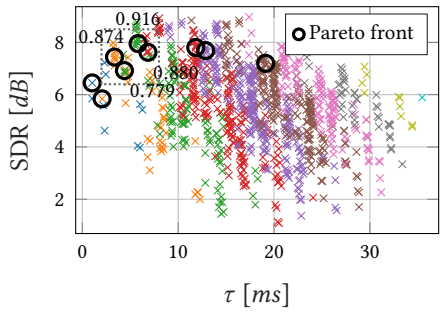


Figure 8.3: SDR vs. network delay τ for all admissible S. Points of Pareto front in Figure 8.2 outperform the majority of admissible S.

rely on this utility metric. According to [Gun+21], selecting microphones with the lowest acoustic cost consistently achieves SDR values in the top 15%.

Results

Here I show the results of joint optimization using acoustic and network cost functions. Figure 8.2 shows a scatter plot of acoustic cost $\mathcal{J}_a(S)$ vs. network cost $\mathcal{J}_n(S)$, where each data point again represents a set selection S. Obviously, there is a trade-off between acoustic and network cost. The black-circled data points form the Pareto frontier, i.e., combinations where neither cost can be any further reduced without increasing the other one. These points offer an optimum trade-off between acoustic and network cost, and are found for the minima of Equation (8.7) by varying the value of $0 \leq \alpha \leq 1$. Thus, minimizing the cost function in Equation (8.7) captures the intention of emphasizing either acoustic or network cost.

Furthermore, Figure 8.3 shows the SDR ground truth vs. network delay. The

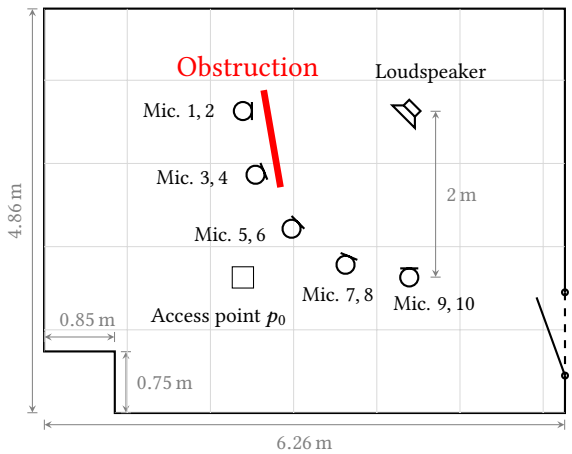


Figure 8.4: Illustration of the experimental setup. Line-of-sight between the loudspeaker and microphones 1–4 is blocked.

circled data points correspond to the Pareto front in Figure 8.2. Similar to Figure 8.1, the optimality of the acoustic cost function does not exactly reflect the optimality with respect to the SDR; the subset of nodes with the highest acoustic utility has a high SDR, but not necessarily the highest.

Accordingly, the four Pareto points framed by the gray box are the most desirable, since they offer high SDR values at low delay. They can be obtained by choosing $\alpha \in [0.42, 0.67]$, but this does not necessarily generalize to other applications. The number next to the chosen points represents the fraction of admissible S which are strictly worse, i.e., achieve lower SDR and incur higher network delay. Obviously, the selected combinations still outperform the majority of other possible ones.

Up to this point, I showed how microphone subset selection relies on both QoI and QoS. By sweeping over different subsets and different α, I have shown the Pareto frontier of the cost functions and found a range of α that yields high acoustic and network gains (or in other words, low costs). The results from the cost functions were good to be in the top 15% of the SDR (ground truth). Meanwhile, sweeping over the parameters is valid for stationary microphones and a stationary speaker. Once the speaker starts moving, it becomes impracticable and in some cases even sub-optimal. Hence, I show in the next subsections other approaches (with slight changes in the formulation) for a moving speaker.

8.2 Microphone Selection for Moving Speakers

Modeling the QoI is, as seen in [Gun+21], complex and computationally challenging, yet modeling QoS is not easier. In fact, the authors in [Liu+10] claimed that QoS is not easy to model because there is no clear interrelation between physical, medium access control, and network layers. Hence, they used a “black box" to generate QoS samples. Next, a value function is derived by interpolating these samples. A major drawback of this approach is the memory required to keep a record of these samples, which may be infeasible for large scale problems, making them less desirable with such constraints.

Similarly, neural network-based models are used to estimate the optimal configuration. In [Akb+19], the authors assumed that the black box is a mixed integer programming model, which requires a long termination time. The objective of the black box is to find the optimal configuration and the corresponding quality. Next, they used samples from the black box to train the neural network and used the trained model to estimate the output without the need for a memory to save the samples (as in interpolating-based models). Neural networks are, however, by nature regression or classification models and the samples are generated at random. Therefore, they may need a large number of samples to achieve good performance, especially for a constraint guarantee.

Alternatively, RL solutions are experience-based models that learn to take best actions based on previous state-actions and their corresponding rewards (e.g., Q-learning). let us consider Deep Q-Learning, it uses neural networks for value function approximation, hence, they require less memory and are very useful when collecting samples is expensive. Unlike neural network-based models, Deep Q-Learning kind of chooses which samples to collect; they rely on epsilon greedy (or epsilon decaying) method to select the samples during the exploration phase.

Based on the above givens, RL is the best candidate to use when collecting samples is expensive. This fits well to the problem at hand, when calculating the acoustic quality is expensive, especially for a moving speaker. In this section, I consider replay-based RL models [HHA19] to update the RL function approximation for high data efficiency. The results in this section are based on the work in [ARK21a]. Although I use here a mock function for calculating the acoustic quality as a proof of concept, later on (in Section 8.3) I use pre-computed acoustic quality metrics.

8.2.1 Problem Definition

The problem at hand is related to hybrid e-learning and hybrid conferences. There is a camera and a set of wireless microphone nodes $\|V\|$ distributed inside a room

to stream the audio data via an access point as the gateway. The main difference compared to Section 8.1 is that the speaker is no longer stationary but moving. In fact, there could be more than one speaker but only on is active at a time. An acoustic application simultaneously analyzes the classroom (speaker localization and speech enhancement), therefore, again the QoI (G_{QoI}) and QoS (G_{QoS}) gains are separated.

The acoustic quality per each microphone is a distance-based function given by $q_v(t) = \frac{1}{\Delta_v(t)}$, where $\Delta_v(t)$ is the distance between the speaker and the microphone v at instant t. The distance can be easily estimated via the installed camera. Then, I select a subset of nodes $S(t) \subseteq V$ for recording, so that $\tau(t)$ is the corresponding total transmission delay. Note that in this scenario, there are no obstacles as in Section 8.1.

Now I define two objectives, which I compare them between later. **First**, as an application developer, I set a maximum number of recording nodes δ_n and an upper bound on the network delay δ_d. Then, the objective is to select a subset of recording nodes $S(t)$ to maximize the *QoI* under *QoS* constraints

$$\begin{aligned} \max_{a} \quad & G_{\text{QoI}}(t) = \sum_{n \in S(t)} \frac{1}{\Delta_n(t)} && \forall t \in T \\ \text{s.t.} \quad & |S(t)| \leq \delta_n && \forall t \in T \\ & \tau(t) \leq \delta_d && \forall t \in T \end{aligned} \tag{8.9}$$

The **second** objective it to select a subset $S(t)$ to maximize the *QoI* while discouraging having the delay higher than δ_d (i.e., soft constraint):

$$\begin{aligned} \max_{a} \quad & \sum_{n \in S(t)} \frac{1}{\Delta_n(t)} - w \cdot \max(0, \tau(t) - \delta_d) && \forall t \in T \\ \text{s.t.} \quad & |S(t)| \leq \delta_n && \forall t \in T \end{aligned} \tag{8.10}$$

Note that the goal from comparing both objectives is not to find out which objective is better, but to understand the impact of moving from hard constraints to soft constraints.

8.2.2 Reinforcement Learning Solution

In the implementation of the RL environment, the continuous state (observation)

space $s(t)$ represents the position of the speaker. The action space is a discrete space that selects a subset of nodes $S(t) \in 2^{\|V\|}$ to stream the data.

Next, I rewrite the objective in (Equation (8.9)) as follows:

$$\begin{aligned} \max_{a} \quad & G_{\text{QoI}}(t) = \sum_{n \in S(t)} \frac{1}{\Delta_n(t)} \\ & -c_1 \cdot \max(0, |S(t)| - \delta_n) \\ & -c_2 \cdot \max(0, \tau(t) - \delta_d) \quad \forall t \in T \end{aligned} \tag{8.11}$$

where c_1 and c_2 are positive penalty numbers. However, this requires calculating the acoustic quality (QoI) and the resulting delay (QoS) at each time step for every chosen action. Recall that calculating these metrics is not easy, computational expensive and may require collecting empirical data first. To avoid unnecessary computation when getting learning samples, I define the reward function $r(t)$ as follows

$$r(t) = \begin{cases} -c_1(|S(t)| - \delta_n)^2 & \text{if } |S(t)| > \delta_n \\ G_{\text{QoI}} - c_2 \max(0, \tau(t) - \delta_d)^2, & \text{otherwise} \end{cases} \tag{8.12}$$

where $G_{\text{QoI}}(t) = \sum_{n \in S(t)} \frac{1}{\Delta_n}$. Consequently, I do not need to sample (i.e., calculate) any of the delay or quality costs, if and only if the maximum-number-of-recording-nodes is exceeded.

For high values of both c_1 and c_2, the solution has hard constraints, which penalizes any violation of delay or number-of-nodes constraints. Meanwhile, when $c_1 \gg c_2$ and c_2 is relatively a small number, the solution converges to a weighted approach for multi-objective optimization. In other words, c_2 denotes a sensitivity parameter (similar to w in Equation (8.10)) that allows compromising the delay constraint when the gain in the acoustic quality is high enough (i.e., soft constraint Section 2.3.6).

In order to understand the use of risk sensitivity, I use an arbitrary QoI function for the sake of illustration (Figure 8.5). QoI increases with network delay (i.e.,decreasing QoS). Yet there is an upper bound delay $\delta_d = P_0$, at the vertical dashed line, which is not recommended to be exceeded (QoS soft constraint). Nevertheless, the QoI value will almost double if the dashed line moves slightly to the left, i.e., compromising the delay constraints to achieve a higher QoI. In many acoustic applications, such compromise may be valid if the QoI gain is high enough. The

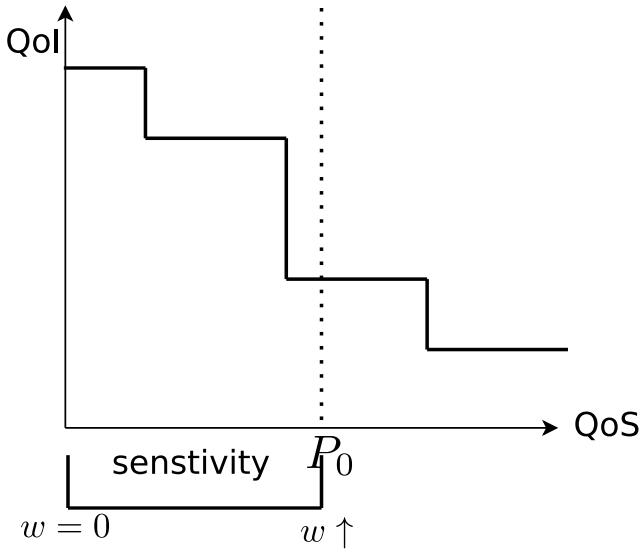


Figure 8.5: Example of risk sensitivity

parameter w is used to control the sensitivity of compromising P_0; the lower the value of w, the less the sensitivity. Note that when $w = 0$, the constraint is ignored.

The reason for having a quadratic max function in Equation (8.12), when calculating the reward is to make our problem differentiable, which eases the process of learning (see Equation (2.7)). For example, if we consider the resulted delay from IEEE PHY MAC [21] (Figure 9.2), the derivative of $\tau(t)$ is a sum of Dirac functions. The max function can be written as

$$\max(0, \tau(t) - \delta_d) = \begin{cases} 0, \text{for } \tau(t) \geq \delta_d \\ \tau(t) - \delta_d, \text{for } \tau(t) \leq \delta_d \end{cases} \tag{8.13}$$

so when differentiating at $\tau(t) = \delta_d$, we will get 0 from left side and $\tau'(t)$, which is a constant Dirac, on the right side, thus it is not differentiable. Meanwhile, when considering $\max(0, \tau(t) - \delta_d)^2$ instead, the function is still differentiable at $\tau(t) = \delta_d$ to be 0.

The RL agent is a DQN agent with experience replay with discount factor set close to 0, since the speaker's position can change arbitrarily. This is in particular useful for our formulation since actions have no impact on next states (speaker's position); therefore, I discourage that the constraint is violated at any state (i.e.,

highly penalized), while ignoring the long-term cumulative reward. More discussion on the impact of discount factor will follow in Section 9.6.1.

8.2.3 Experimental Evaluation

Two similar scenarios were used for evaluating the RL solution. In Scenario 1 (Figure 8.6 (a)), 3 wireless microphone nodes (A, B and C) are equidistant to the sink node (i.e., access point). Accordingly, all nodes have the same delay to the access point. I assume that each node v introduces delay τ_v, which is directly proportional to the total delay $\tau(t)$, i.e., $\tau(t) = \sum_{v \in S(t)} k_v + \tau_v$ where k_v is overhead delay introduced by the wireless MAC protocol (as seen in CSMA/CA) per each node v.

Since all nodes are equidistant to the sink, I simplify the problem and assume that all nodes have the same delay to the access point $\bar{\tau}$. The maximum number of selected microphone nodes is $\delta_n = 2$. Note that in this case, the upper limit delay should be at least $\delta_d \geq \delta_n \bar{\tau}$; otherwise, there is no feasible solution. I set $\delta_d = \delta_n \bar{\tau}$ for this scenario. Note that here, there is no trade-off between the delay and signal quality (as in Equation (8.10)) due to the dependency between the δ_d and δ_n, and only the objective in Equation (8.9) can be used. Therefore, I set high values for c_1 and c_2

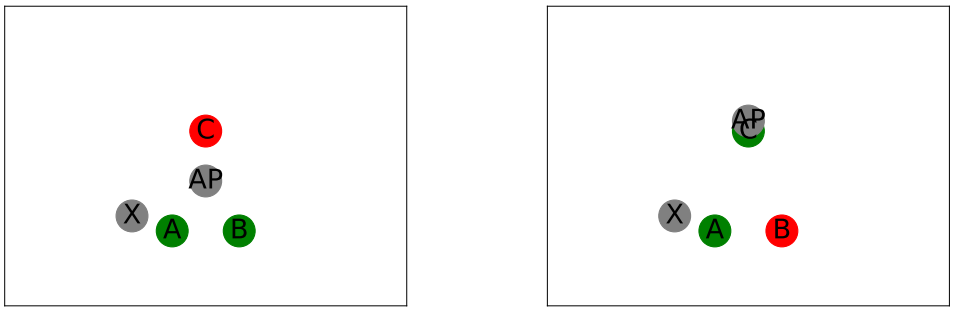
In Scenario 2 (Figure 8.6 (b)), again, there are 3 wireless microphones and $\delta_n = 2$. But the sink is closer to one of the nodes, thus, the delay $\tau(t)$ is no longer equal for all nodes; nodes have different wireless signal qualities and hence different transmission delays. Consequently, I apply the trade-off objective in Equation (8.10) to this scenario, by setting a high value to c_1 and a low one to c_2.

As a baseline solution, I use the optimization model in Equation (8.9) to both scenarios (Figure 8.7) and compare them to their corresponding objectives. The optimization model is solved using the Gurobi solver [Gur18] with the Pyomo interface [HWW11]. Since Equation (8.9) is already the objective of Scenario 1, I do not expect the RL solution to have better results in terms of QoI. Nevertheless, it is possible that the RL solution results in better results in Scenario 2 since the reward function optimizes Equation (8.10).

Before evaluating the two scenarios, I also evaluate the RL solution when the speaker is not moving.

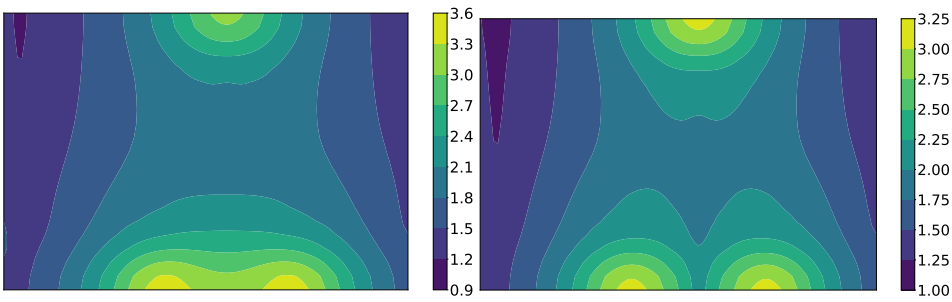
Static Speaker

A sanity check of the RL implementation makes sure that the RL agent learns to take the best action in the simplest case, i.e., when the speaker is not moving. At



(a) Scenario 1 **(b)** Scenario 2

Figure 8.6: Visualization of the layouts



(a) Scenario 1 **(b)** Scenario 2

Figure 8.7: Visualization of the rewards

maximum 3 samples from the real environment (i.e., baseline) are needed and I compare the best solution to the agent's action. In Figure 8.8, the agent's average reward is close the baseline after ≈ 3000 learning steps, which confirms the agent's ability of finding good actions.

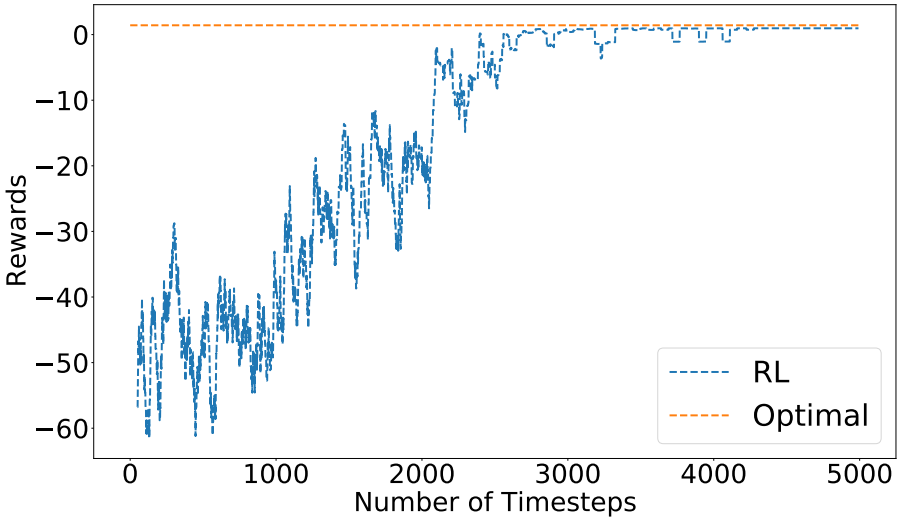


Figure 8.8: Average reward for a static speaker in Scenario 1

Moving Speaker – Scenario 1

I repeat the previous experiment again, but this time the speaker is moving. 200 samples are available from the state space for training RL agents. Two similar agents are used for comparison; they only differ in the number of training steps: 10^4 and 10^6 steps. Thanks to the simplification in (Equation (8.12)), the maximum number of combinations for which an agent needs to calculate the quality per state is ${}^{\|V\|}C_{\delta_n} = 3$. Remember that calculating the quality is expected to be an expensive process.

Supporting speaker mobility requires more training steps, compared to the previous setup. Figure 8.9 shows that the agent with 10^4 training steps obviously behaves worse (lower reward) than the agent with 10^6 steps. Additionally, the RL agent with 10^6 steps has rewards very close to the optimal solution.

I take a deeper look into the results and compare the agent with 10^6 training steps to 1500 samples from the optimization model, where the samples are distributed all over the room. In other words, I compare the agent's solution to the optimal one, in states that it has not seen before. I show the difference between the optimal and

RL solutions in Figure 8.10 (a). It is clear that the RL and optimal solution are very close; the maximum difference, which is 0.048 or ≈ 1% of the maximum reward (see Figure 8.7 (a)) and the average difference is almost 0.

Three spots show differences in reward (Figure 8.10 (a)). At these spots, the agent's solutions have similar rewards to the optimization model, hence, it is expected that after more training steps and more samples, the agent will converge exactly to the optimal solution.

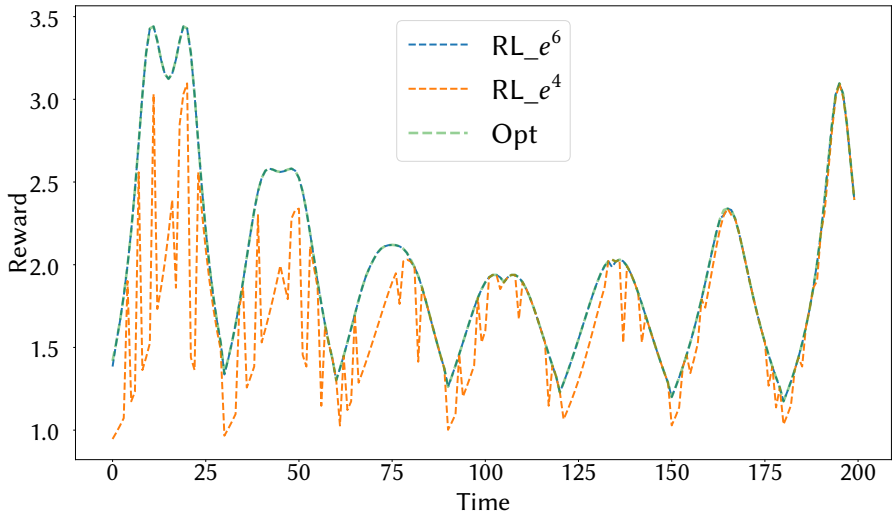


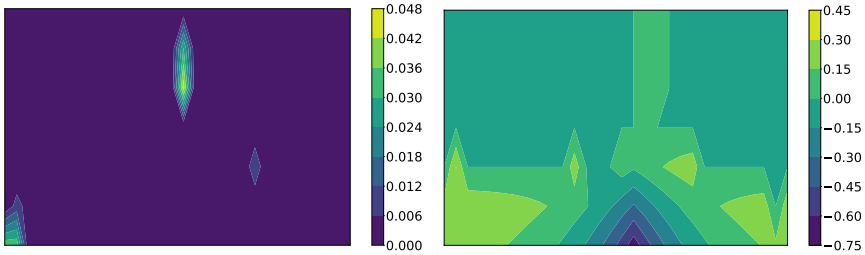
Figure 8.9: Reward for a moving source in Scenario 1

Moving Speaker – Scenario 2

Again, the speaker is moving but this time in Scenario 2 and the agent's solution is compared to the optimization model in Equation (8.9) for 1500 samples. In this case, the RL targets a QoI/QoS trade-off while the optimization model maximizes QoI under QoS constraints. Based on the results from the previous experiment, I train the RL agent with 10^6 steps and compare the differences in rewards (8.10 (b)).

The RL solution learns to compromise between QoI and QoS by achieving higher rewards than the optimization model. Such behavior is noticed when the speaker is between microphones A and B (Figure 8.6 (b) and Figure 8.7 (b)); the acoustic quality is high enough so that the delay constraint is compromised.

Comparing the achieved rewards of both RL and the optimization model in Fig 8.7 (b), the agent achieves a higher reward up to 23% and lower reward till 12%. Nevertheless, the mean average reward is almost the same: -0.04. The performance



(a) Scenario 1 **(b)** Scenario 2

Figure 8.10: Visualization of the rewards difference

of the RL agent can be calibrated by optimizing c_2. Also, increasing the number of training steps and samples may slightly improve the agent's performance, as seen in the previous experiment.

8.3 Minimizing Rate of Changing the Selection

In Section 8.1, I demonstrated the relation between QoI and QoS, then in Section 8.2, I used RL in a small-scale environment to select the best set of microphones for recording or streaming. Here, I update the QoS metric to include the frequency of changing the set of microphones; I assume that these changes (i.e., reconfiguring the microphone set) cause interruptions while adding/removing a new connection. Additionally, I reformulate the problem to be applied to a large-scale environment. The results in this section are based on the output of [Afi+21].

8.3.1 Problem Formulation

In general, selecting the microphone set depends on three metrics:

1. the achieved QoI from the selected microphone set
2. the required delay to collect the audio data from the microphones within the set
3. the frequency of reconfiguring the microphone set while a speaker is active

Audio samples are divided into time frames. Calculating QoI per a time frame k is application-dependent. Hence, I assume that there is a black box that calculates the QoI for a given microphone set. Next, there are different sources of network delay such as queuing, processing and transmission delays. Here, again, I focus

only on the transmission delay $\tau[k]$, i.e., the time needed to transfer data from the microphones to the access point. It is calculated in the same way as mentioned in Section 8.1.1.

I use two weighting parameters, η_Ω and η_τ, for QoI and network delay, respectively. Accordingly, the objective is choosing the microphone set that maximizes the weighted sum of these two metrics, while minimizing the frequency of changing the set when the speaker is active.

8.3.2 Reinforcement Learning Formulation

When everything is static and not changing, it is easy to estimate the best solution for the set of microphones by solving a linear system of equations. But in this problem, the parameters change over time k and the solution at k may not be the best for $k+1$. Moreover, changing the solution (i.e., reconfiguring the microphone set) also impacts quality over time. Hence, I here use RL to find the best decisions to take for each k.

The RL environment is defined by three main components: observation space, action set (i.e., a discrete space) and a reward function. The observation space is continuous, where a vector $s[k] = \left[\mathbf{u}^{\mathrm{T}}[k], S^{\mathrm{T}}[k], \zeta[k]\right]^{\mathrm{T}}$ describes the continuous acoustic utility per microphone $\mathbf{u}[k]$ and has a binary (multi-discrete) indicator vector for the selected set of microphones $S[k]$ at time frame k. For $\|V\|$ microphones, there are $2^{\|V\|} - 1$ different possible combinations for microphone selection, so that I discard empty set selection. Moreover, there is a binary activity detection variable $\zeta[k]$ to indicate speech pauses. Hence, the length of the observation vector is $2\|V\| + 1$.

The action corresponds to selecting at most one microphone to switch its state per time frame k, i.e., changing the selected microphone set by adding or removing a microphone, or not changing the microphone set. Hence, the size of the action set is $\|V\| + 1$. This is better than other solutions, such as picking a set of microphones, because here the action space is smaller and other solutions will take loner time for learning.

Based on the chosen actions, the RL agent receives corresponding rewards. The reward function combines three different aspects: QoI, network delay and number of changes applied to the set selection within a predefined number of time frames κ. I explain in the following the reward modelling.

In order to evaluate the QoI of a certain set of selected microphones relative to all other microphone subsets, I define a goodness metric Ω. All subsets are sorted at each time instant k in ascending order w.r.t. the QoI metric given by the black box . Then, the goodness $\Omega[k] \in [0, 1]$ of the selected subset in $s[k]$ is the

normalized rank (rank of selected subset divided by the number of all possible subsets $2^{\|V\|} - 1$) in ascending order; 0 is the worst and 1 is the best. The network delay is calculated in the same way as Section 8.1.1, hence, the higher the delay the lower the quality of the corresponding subset. Moreover, every time the selected set of microphones changes from $k-1$ to k such that $S[k-1] \neq S[k]$ (i.e., a new microphone is selected/discarded), I reduce the reward by a penalty term

$$\Delta[k] = \begin{cases} k \bmod \kappa + 1, & \text{if } S[k-1] \neq S[k] \\ 0, & \text{otherwise.} \end{cases} \tag{8.14}$$

This ensures network stability and decreases jitters or delays due to reconfiguration within κ duration. Consequently, the reward $r[k]$ is given by

$$r[k] = \begin{cases} \eta_\Omega \Omega[k] + \eta_\tau \tau[k] - \Delta[k], & \text{if speaker active} \\ 0, & \text{otherwise} \end{cases} \tag{8.15}$$

where the constants η_Ω and η_τ are used to normalize the goodness $\Omega[k]$ and delay $\tau[k]$, respectively. Note that the observation space has acoustic utilities which are calculated per microphones (similar to Section 8.1.2), while here the goodness depends on the quality of a set of microphones. As shown in [Gun+21], there is a direct correlation between the acoustic utilities per microphones and the set of selected microphones.

I use the deep-Q-network (DQN) as the RL agent with experience replay and epsilon greedy for training [HS19]. Although other training models would also be plausible, DQN is recommended because the action space is discrete (Section 2.3.2). Consequently, I use neural networks to model the relation between microphone set selection and microphones utilities on the one hand, and the achieved reward of adding or removing a microphone from the selected set on the other hand.

8.3.3 Evaluation

Here I define the setup for data generation and two baselines for benchmarking– quality and frequency of changes of– the RL solution.

A total of $\|V\| = 10$ microphones are placed in a room with random, but fixed, positions, depicted in Figure 8.11. The room has dimensions of 5m × 5.2m × 3m. The microphones' sampling rate is $f_s = 16$kHz, the size of data sent per each microphone is $L = 64$ms long, which is later used for estimating the microphones' utility. The details of calculating a generic microphone's utility is described in [Gun+21]. In

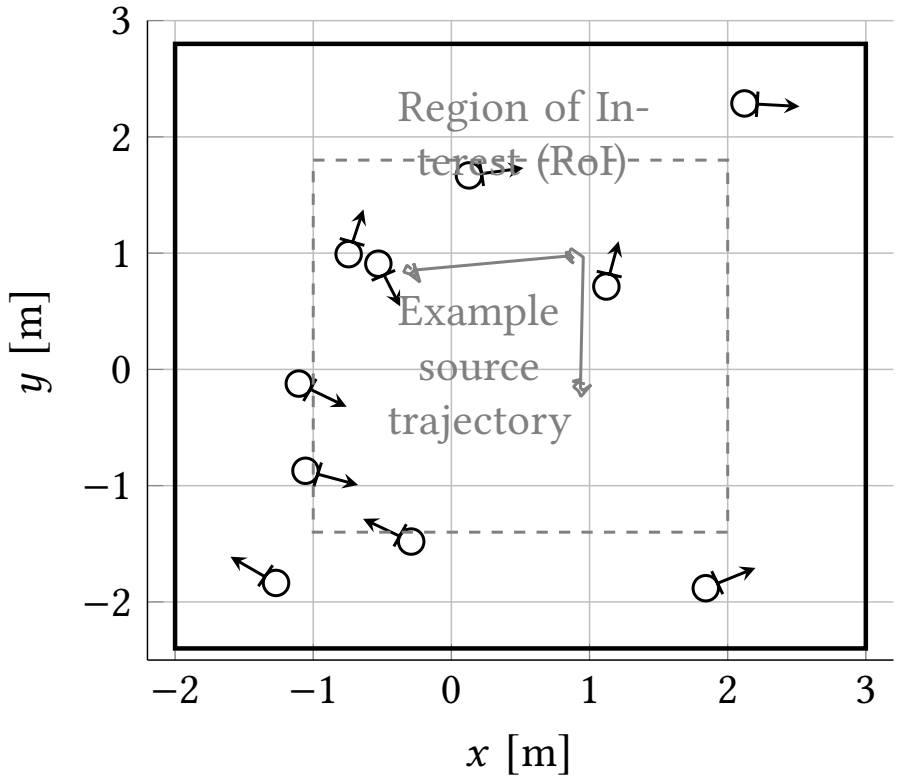


Figure 8.11: Top-down view on an exemplary acoustic scenario.

this setup, the BSS application is considered so that the QoI metric is the Signal-to-Distortion Ratio (SDR) [FGV05].

All microphones and the active speaker are at a height of 1m. An active speaker moves randomly within the region of interest (RoI) of size 3m × 3.2m. To simulate a human movement, the speaker remains almost stationary for 8s, then moves to a new random position, by random uniformly selecting x and y positions, in the RoI within 2s (i.e., low speed). Each experiment trial is 28s long, such that it contains 3 stationary intervals, and 2 movement intervals. An example is shown in Figure 8.12.

Two different speaker signals (male and female speech) and 30 random trajectories are used for simulation, resulting in a total of 60 unique combinations of signal and trajectory. To facilitate computation of the acoustic reward in Section 8.3.2, the SDR values for each possible microphone selection are computed for each trajectory in advance. All microphones are assumed to be synchronized.

The wireless bandwidth is 20 MHz and the wireless attenuation depends only

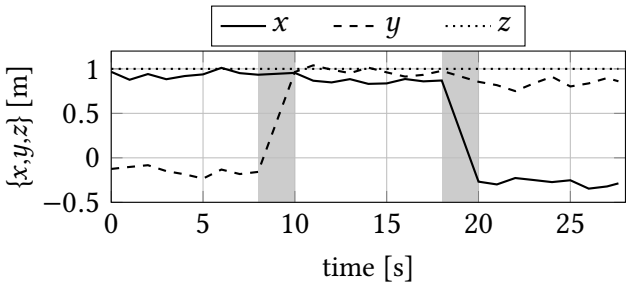


Figure 8.12: Exemplary speaker trajectory. Shaded intervals indicate source movement.

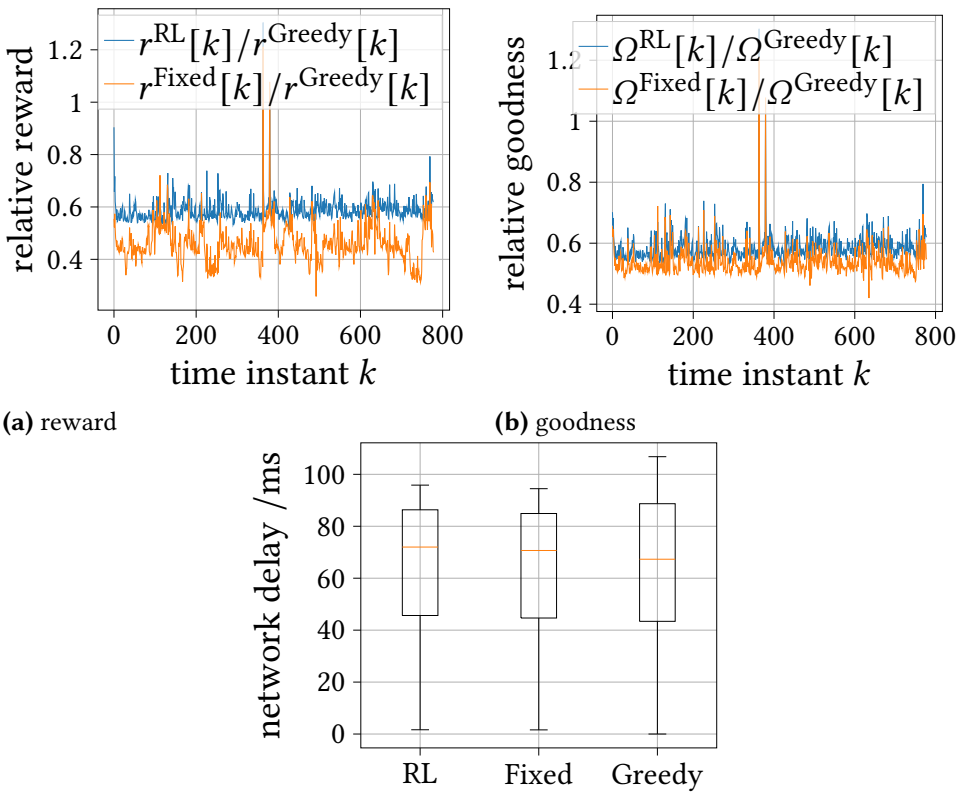
on the distance d_v between the microphone v and the access point. Next, I use 802.11 PHY standards [21] to calculate the transmission delay based on the SNR determined by the distance-based attenuation model $\frac{1}{d_v^{2.5}}$. Since the positions of the microphones are fixed, the delay for each microphone is time-invariant and can be calculated once at the beginning of the simulation (which I do here), but other alternatives are also plausible such as empirically measuring the delays.

Half of the simulated trajectories are used for training while the other half is used for evaluation. All recordings are equally long, resulting in time frames $k \in \{0, 1, \ldots, 842\}$. I choose the normalization factors so that $\frac{\max_k(\eta_\Omega \Omega[k])}{\max_k(\eta_\tau \tau[k])} = 9$. This prioritizes the optimization of the acoustic quality over the minimization of the network delay. Nevertheless, they can be tuned for any application with other values.

During the training phase, both the initial microphone set selection, the trajectory of the speaker and the starting time frame of the trajectory k are randomly selected. The random process of selecting the initial $S[-1]$ is repeated 2×10^5 times during training, where I set $\kappa = 5$ to avoid many changes in microphone selection within few number of time frames.

Baselines

Furthermore, I simulate two baseline methods for benchmarking the proposed RL solution. First, *Fixed*, where the initial selection of microphones is random (in terms of how many and which ones) and does not change as the speaker moves. Second, *Greedy*, where the activity of at most one microphone is changed per k to maximize the instant reward based on oracle knowledge. An oracle is a black box that can pre-compute the SDR values for any microphone set selection. In a real-world scenario, such an oracle does not exist as it requires a reference (original) signal,



(a) reward

(b) goodness

(c) delay

Figure 8.13: Comparing RL to Fixed and Greedy baselines where $\Delta[k] = 0$

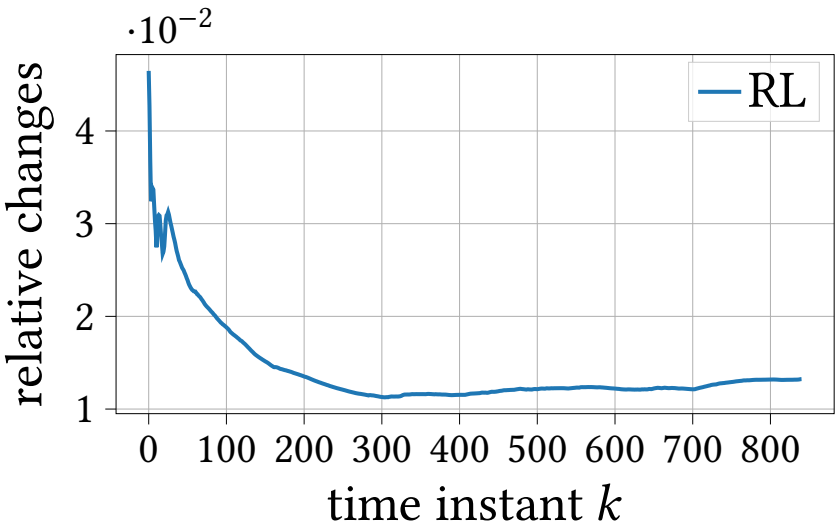


Figure 8.14: Relative number of changes per time frame

which is provided as an input to the oracle during simulation, but it is not needed by the RL solution during inference.

The greedy approach is short sighted, hence, it maximizes the instant reward and ignores the penalty due to changes, i.e., $\Delta[k] = 0, \quad \forall k$.

Results

During evaluation, each trajectory starts from the beginning ($k = 0$) with 50 random initializations for trajectory generation and initial microphone set selection. The focus in Figure 8.13 is on the rewards in terms of QoI and transmission delay, while ignoring the penalty due to reconfiguring the selected microphone set. For a better visualisation in the figures, I ignore the results with speech pauses.

In Figure 8.13 (a), I plot the relative rewards $r^{\text{RL}}[k]/r^{\text{Greedy}}[k]$ and $r^{\text{Fixed}}[k]/r^{\text{Greedy}}[k]$ for the RL and Fixed approaches, respectively. As can be seen, the RL approach shows higher rewards than the Fixed baseline. The fact that the curves corresponding to the relative rewards of the RL and Fixed approach are below one shows that the Greedy approach obtains the highest rewards. However, recall that the Greedy reward values do not include the penalty $\Delta[k]$ due to changing the microphone selection and are based on oracle knowledge, which is not always available or cannot be obtained in real-time operation. Yet, it is still plausible here for simulation purposes.

I also look at the goodness for both RL and Fixed approaches relative to the

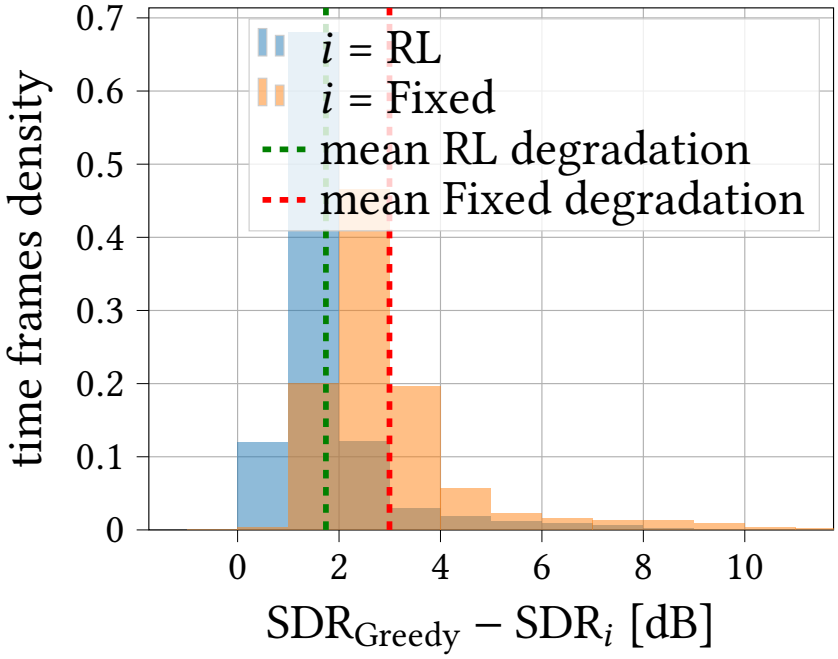


Figure 8.15: Histogram of SDR degradation

Greedy baseline, see Figure 8.13 (b). Similarly to Figure 8.13 (a), the RL approach shows, again, better performance than the Fixed baseline but here with a smaller gap, because I ignore the delay's impact and the weight parameter η_Ω is dropped. Meanwhile, Figure 8.13 (c) shows that both the Fixed baseline and the proposed RL approach show a similar distribution of the resulting network delays. Nevertheless, the Greedy approach has a higher upper bound on the network delays, but this is compensated for by higher goodness (Eq. Equation (8.15)) resulting in the optimum reward per instant time frame.

The RL approach achieves on average 60% of the Greedy rewards (Figure 8.13 (a)). Remember that the Greedy baseline is the best what we can achieve, when ignoring penalties due to network reconfiguration. Hence, I look at the cumulative relative number of changes of the RL approach with respect to the Greedy baseline (Figure 8.14), which lets us focus on network stability. First, I observe that the RL approach seems to apply many changes at the beginning to minimize the number of changes for the remaining part of the trajectories. Second, the Greedy approach changes the set of selected microphones hundred times more than the RL one. Accordingly, the proposed RL approach sacrifices 40% in goodness performance (Figure 8.13 (b)) to achieve more stability for the microphone set selection.

This results in the following question: what is the equivalent of 40% performance difference in goodness to that of SDR? Remember that goodness is only a rank metric and not an absolute number. In other words, I need to quantify the stability in terms of SDR. To answer this question, I show in Figure 8.15 the normalized histogram of SDR degradation for both RL (blue) and Fixed (orange) approaches, w.r.t. the Greedy one. In other words, small values correspond to a performance similar to the Greedy approach, while large values correspond to a worse performance. The vertical lines represent the mean difference of 1.7dB for the RL approach (green) and 3dB for the Fixed one (red). Additionally, the distribution of RL degradation has a skewed distribution, i.e., low differences are more probable than large ones. Also there is a tendency of being heavy tailed (Kurtosis for RL is 13.7 while for Fixed is 8.3).

To sum up, the results show that the RL actions are better than simply keeping the initial microphone set selection (Fixed), while the changes applied by the RL agent achieves on average 60% of the (oracle) Greedy approach's rewards. Nevertheless, this 40% sacrifice is equivalent to 1.7dB in SDR. This compromise is acceptable when considering that the number of changes done by the RL approach is hundred times less than that of the Greedy one, resulting in a more stable solution compared to the greedy approach.

8.4 Conclusion

In this chapter, I studied the relation between QoI and QoS when selecting a subset of microphones. First, I assumed that both microphones and speakers are static and showed that not only QoS benefit from subset selection but also QoI. Then, I used RL to microphone set selection, using a simple QoI/QoS model. Finally, I used RL again yet for a more complex QoI model adopted from acoustic signal processing and considered the frequency of changing the selected subset as a metric that influence both QoI and QoS. The RL solutions showed in general better performance compared to greedy approaches and in particular to a naive approach where the subset selection does not change over time.

9 Movement Selection in Dynamic Environment

Now I take dynamics to the next level, where not only the speakers are moving, but the microphones move as well. In contrast to the dynamic properties in Sections 8.2 and 8.3, here the dynamics from moving microphones can be controlled, a.k.a. autonomous vehicles. Autonomous vehicles – like unmanned ground vehicles (e.g., social robots [AL21]) or unmanned aerial vehicles (UAV, e.g., drones [JMA15]) – have a broad range of applications, such as entertainment, industrial scenarios and ambient-assisted living, all working in dynamic environments.

Such control presents both an opportunity and a challenge. Controlling such vehicles via rule-based systems can be feasible, but is often labor-intensive, rigid and static. The user's needs and preferences or the environment may, also, change over time, while adapting hard-coded rules to changes in application setup or environment is difficult. As an alternative, RL-based approaches allow vehicles to continuously learn from interactions with each other and with their environment to adapt their behaviour.

To illustrate the idea, I consider the scenario of a room (e.g., meeting room or large conference hall) with several human speakers and a couple of such autonomous vehicles, equipped with microphones (Figure 9.1). The objective is to move the microphones to appropriate positions in the room to obtain high-quality recordings of the speakers (e.g., for question-and-answer sessions in conference setups). Hence, there is a need for an algorithm to move these vehicles so that they can best acquire audio signals when speakers take turns (e.g., multiple people asking questions from different locations), under the constraint that only a few vehicles are available. Note that the ideas can be transposed to similar problems in other modalities like video streaming, but for the sake of concreteness and clarity in presentation, I limit the description to audio streams.

Depending on the problem at hand, there are multiple objectives that yield a multitude of solution behaviours. Such a scenario can support different concrete use cases, with different desired vehicle behaviours. For instance, localizing a speaker as best as possible or to record the audio in stereo [BK18; Sch+17a] will require moving multiple vehicles towards the "active speaker" to act as a microphone array. Audio filtering, on the other hand, requires at least one vehicle close to the "active speaker", and data from other vehicles is used for noise cancelling [Xu+18].

Here, there are two different problems that can (and will) be solved using the

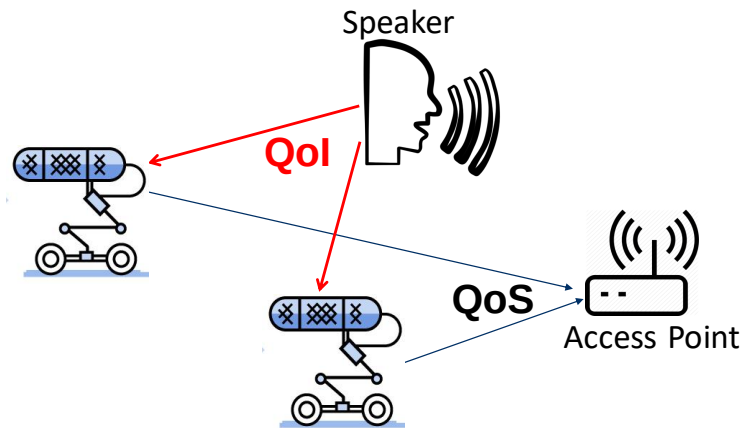


Figure 9.1: Example of the motivating scenario

same approach. This approach must be flexible enough to support such different behaviours; ideally, it should learn how to best behave in these different use cases, even ones that were not foreseen during system design. This flexibility is difficult to achieve with conventional, non-learning-based approaches.

I assume that the vehicles are wirelessly connected to an access point that forwards the collected data for monitoring and/or further processing. Therefore, it is important to maintain good connectivity between the vehicles and the access point to avoid packet losses and unnecessary delays.

Simultaneously, it is important to choose the right locations for best audio quality (e.g., to be close to the active speaker). These two aspects can be in conflict with each other, creating a trade-off between audio and network quality. To express this trade-off, again I use QoI and QoS to evaluate the solution's performance

This trade-off has not yet been fully explored in the literature, especially not for groups of autonomous vehicles and dynamic scenarios (e.g., alternating speakers). A possible approach to explore this trade-off would be to formulate an optimization problem and try to solve it in (near) real time Section 8.1. But this is fraught with many obstacles; to name but a few: the high solution times for conventional optimization problems, the non-obvious way in which QoI/QoS interact towards overall application quality, and the difficulty to adapt to changing application objectives.

Next, the question is, how to build such autonomous vehicle systems. In general, there are two main building blocks: *scene understanding* and *decision making* [LI04]. Understanding the scene is via sensors (such as cameras and microphones) to perceive the environment and localize the vehicles. This is achieved by using various

techniques such as motion detection [Sia+18], semantic segmentation [Sia+17], and mapping [Mil+18]. Here, I assume that these techniques are at hand and are error-free.

Decision making is divided into two blocks [Kir+20]: *planning* and *control*. The *planning* block is necessary for motion and trajectory estimation. There are many possible assumptions regarding trajectory estimation. For instance, trajectories could be static when the route to the destination is predefined and the traffic density is already known [LI04]. Other planning approaches [LaV06] consider dynamics in the environment such as in indoor environments [LK99]. The *control block* decides aspects like speed, steering angle, or braking. Here, *I assume that the speed is constant and I control only the steering angle and braking*.

Similar to the work in [Sah+21], I rely on neural networks to implicitly model the motion patterns of active speakers and act upon. Nevertheless, this is not the only role of neural networks; they also approximate the QoI or QoS values for a given system state. As mentioned in Section 8.1, calculating QoI is based on the data collected from the sensors. One of the reasons why one uses neural networks for approximation instead of directly calculating QoI or QoS is that the algorithms used for calculating QoI are usually time-consuming and normally cannot be expressed explicitly (Section 8.3). Consequently, I use neural networks to implicitly extract features from these data (e.g., location of vehicles and speaker) and express the combination of current data and taken actions via some values [HGS15]. Hence, the neural networks handles the functions of both planning and control blocks.

There exists some work concerned with data acquisition quality (i.e., QoI) and network connectivity (i.e., QoS), when controlling vehicles. The work in [FGD16; JMA15] optimized the movement of one UAV to maximize QoI. On one hand, they assumed that all nodes are fixed except for one node that acts as a relay node for data collection and routing. On the other hand, I assume that the nodes are installed with sensors and move around for data collection, while only one node is fixed as a gateway. This adds an additional challenge with respect to the position of the mobile nodes. Furthermore, I assume that all mobile nodes have a direct connection to the access point, hence, they do not need relay nodes. Meanwhile, considering the challenge of a mobile sink node (with fixed sensors), instead of mobile microphones, is a simplified version of my problem and should be straightforward, since I then control only one node.

Social robotics environments [AL21] are very close to the scenario at hand. These robots are often equipped with one or more sensors and move to improve data acquisition. Nevertheless, such work mainly focuses on data acquisition quality and ignores network aspects. Similarly, planning the trajectory for the vehicles has

been studied in [Cha+19; CSB19], but the focus was only to achieve low latency and low wireless interference.

Combining both QoS and QoI was modeled in [Kat+20], but under the assumption that sensor nodes do not move and only a subset of these nodes is selected for maximizing QoI with a certain QoS level. In summary, previous work aims at either optimizing data acquisition with best effort network performance or maximizing QoI for a given QoS. Unlike previous work, I explore the trade-off between QoI and QoS.

To address these challenges, I develop a dynamic programming-based centralized (Section 9.2) and multi-agent (Section 9.3) solutions. I show empirically how a centralized solution has superior performance compared to standard, ad hoc solutions (Section 9.5.2). Additionally, I develop two variants of *multi-agent* deepRL solutions that are more scalable and practical. I compare these solutions to the solution with fixed microphone positions and when each speaker carries a microphone (Section 9.5.5). In addition to empirical evaluations, I attempt a theoretical analysis to explain how the changes in the scenario (e.g., speed of the speaker) impacts the deepRL training and performance (Section 9.6).

9.1 Problem Formulation

The results in this section are based on the work in [ARK21b], where I discussed with Arunselvan Ramaswamy possible formulations and RL tuning. The problem comprises assumptions about speakers (Section 9.1.1), vehicles carrying microphones along for acoustic data collection[5](Section 9.1.2), and wireless transmission to the access point (Section 9.1.3). Acoustic and wireless properties together describe the utility of a microphone (Section 9.1.4).

9.1.1 Speaker

N speakers are in a room, with at most one speaker active at any time; the *active speaker* trajectory has a stochastic distribution. The speaker's speed is a $U(0, v_{\text{src}})$ random variable, i.e., it is sampled uniformly from the interval $[0, v_{\text{src}}]$ every time a speaker talks. The initial direction is randomly selected and can slightly change while moving. The speaker is reflected when it bounces a wall. Speakers are anomalously treated, meaning that no information about a speaker is retained once

5 Note that vehicle and microphone represent the same thing; I interchangeably use then for logical-meaning sentences. For example, a vehicle moves at slow speed and a microphone records acoustic data)

it stopped speaking; a new talk spurt is treated the same irrespective from which speaker it originates

Without loss of generality, I assume that the talk time of each active speaker is uniformly distributed in $U(\tau_{\min}, \tau_{\max})$; talk times of consecutive talk spurts are stochastically independent. Hence, the average talk time τ is the same for all speakers. Once a speaker finishes talking, there is no acoustic pause, I uniformly and independently choose the next speaker at random.

9.1.2 Microphones and acoustic quality

M moving microphones record audio data from a speaker and transmit them to a fixed access point via some wireless transport (Section 9.1.3). Unlike the speaker's movement, I control the movement of microphones: the control system decides, for each microphone, whether it moves at all and its direction. The Microphones move at a constant speed v_m, or 0. I assume that all microphones are active all the time, so there is no microphone selection here, just to make the problem easier to learn.

At every time-step t, I calculate the *acoustic gain* $g_m(t)$ provided by microphone m from its distance to active speaker at that time:

$$g_m(t) = \begin{cases} -\frac{1}{3}\log\left(\frac{d_m(t)}{d_{\text{th}}}\right), & \text{if } d_m \leq d_{\text{th}} \\ -3\log\left(\frac{d_m(t)}{d_{\text{th}}}\right), & \text{otherwise} \end{cases} \tag{9.1}$$

where d_{th} is a threshold distance at which the relation between d_m and acoustic gain (g_m) changes [SK18]. Again, I highlight that in practice calculating the acoustic gain relies on more complex and time-consuming algorithms. They are not easy to compute in real time [GBK21]; they are more suitable as a ground truth. Instead, I use simpler acoustic features (namely, distance between microphones and the speaker) and let the acoustic gain be part of the utility.

9.1.3 Wireless data transport

Each microphone m sends to the access point audio data of constant size in regular intervals. At time step t, microphone m takes $\delta_m(t)$ time units to do so. This time $\delta_m(t)$ depends on the available data rate, which in turn depends on the wireless signal-to-noise ratio at the access point from microphone m, which in turn depends on distance as well as the number of microphones. I use IEEE 802.11 PHY standard parameters [21] to map SNR to data rate and required time to transmit a packet.

Using the location of the vehicles and the access point, I estimate the attenuation of the wireless signal via the log-normal shadowing model $\mathcal{N}(0, 1.6)$ [Uni21]. Based

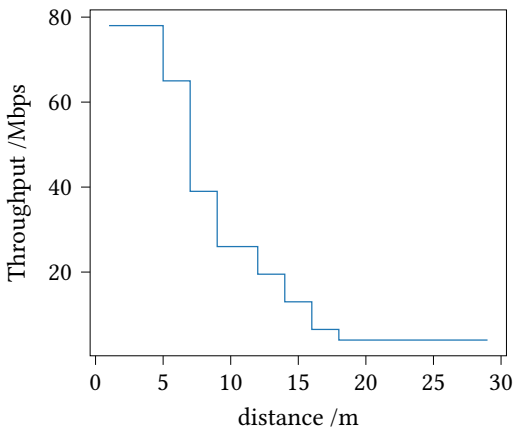


Figure 9.2: IEEE 802.11 data rate vs distance; step function resulting from finite set of 802.11 transmission parameters [21]

on that, I calculate the wireless SNR and thence the transmission time from each vehicle. Figure 9.2 shows the data rate for the step-wise characteristic resulting from IEEE 802.11 finite number of PHY modes.

For simplicity, I ignore propagation delay, medium access delays, as well as retransmission overhead (caused by, e.g., interference from other access points). This is justified as propagation delay inside a cell is marginal in any case, and medium access as well as interference-induced retransmissions can be kept under control by a properly working MAC protocol (I do *not* assume an 802.11 MAC, just 802.11 PHY parameters).

9.1.4 Utility

Finally, I define the *utility* of microphone m as

$$u_m(t) = g_m(t) - w_\delta \delta_m(t). \tag{9.2}$$

To compute utility, I need to normalize δ_m as it is measured in seconds but g_m is unit-less; this is done by factor w_δ, with unit $1/\mathrm{s}$. Moreover, I use w_δ to weight acoustic QoI and network QoS performance against each other. In practice, this weight can be chosen by the application developer. In the following evaluation, I ensure that $0 < g_m(t) \leq 1$ and $0 < w_\delta \delta_m(t) \leq 1$.

I consider two popular models for the utility of a set of microphones. First, the

joint model [BK18] calculates the sum of the microphone utilities

$$r_{\mathrm{J}}(t) = \sum_{m \in M} u_m(t), \tag{9.3}$$

which is useful for stereo streaming applications.

Second, the *contented* model [Xu+18] selects the maximum utility of all microphones

$$r_{\mathrm{C}}(t) = \max_{m \in M} u_m(t) \tag{9.4}$$

which is relevant to noise-cancelling applications.

9.2 Centralized Deep Reinforcement Learning Solution

In this solution, there is a single, central agent running at a central unit (e.g., access point). I assume that this agent knows the locations of the vehicles – installed with microphones – and location of the active speaker (e.g., via one of the localization techniques mentioned earlier in this chapter). Hereafter, I define a continuous observation space so that the state vector $s(t)$, at time slot t, contains the position of each vehicle as well as the position of the active speaker.

At time slot t, action vector $a(t)$ determines the direction of movement of all vehicles. Specifically, each vehicle may move in one of 8 directions: horizontal, vertical and diagonal direction. Additionally, $a(t)$ may specify that a vehicle does not move. Hence, there are 9 possible actions per vehicle, and $a(t)$ is one among 9^M possibilities.

At time slot t, the agent picks an action $a(t)$ and receives feedback for it (performance measure at time slot t), in the from of rewards r calculated using either the joint or contended model from Section 9.1.4, denoted as r_J or r_C, respectively.

Using dynamic programming, the problem of optimally controlling the vehicles reduces to finding a sequence of actions, $\{a(t)\}_{t \geq 0}$, such that the cumulative discounted rewards – for r_{J} or r_{C} – are maximized over time. In other words, maximize:

$$\sum_{t \geq 0}^{\infty} \gamma^t r(t), \tag{9.5}$$

where r could be r_{J} or r_{C}. This boils down to calculating the optimal Q-function Q^*. I train a deepRL (intelligent agent) in order to best approximate Q^*, i.e., to minimize the squared Bellman loss function Equation (2.5). Note that I train the

deepRL in order to learn from past experiences, which is emulated through the use of an experience replay [SB18, Ch. 14].

9.3 Multi-agent DeepRL Solution

The centralized solution presented in Section 9.2 exhibits excellent empirical characteristics, but it is neither practical nor scalable. To overcome this, I develop a multi-agent solution wherein each vehicle is controlled by an autonomous agent (M agents in total). Hence, agent m only determines the direction of movement of a microphone m.

Furthermore, it decides based on *partial knowledge of the environment*. I assume that agent m only knows the positions of vehicle m and the active speaker; it does not know the position of other microphones. Moreover, it is also blind of their existence. The agents are, however, linked through the reward function. All agents obtain the same reward at time t. Using this common reward structure, the empirical results show that the agents learn to cooperate with each other.

To summarize, I define the environment as follows: there are $m = 1, \ldots, M$ agents whose observation $s_m(t)$ represents the position of microphone m and the active speaker position. The action per each agent $a_m(t)$ decides one out of 9 possible actions. Accordingly, I reduce the size of both observation space to be for the location of the agent (per mic) and the speaker (size = 4) and action space (size = 9), compared to the centralized solution (observation size = $2(M + 1)$ and action size = 9^M).

I develop the solutions based on two multi-agent DeepRL paradigms, (a) *shared policy paradigm* [Zha+19], and (b) *separate policy paradigm* [Lan+17]. In the former, I train a single deepRL to take actions in lieu of each of the M intelligent agents. To facilitate effective training, the experiences of all the agents, at every time step, are collected in the experience replay buffer. Training using this buffer amounts to learning from the past experiences of all the agents. This solution is then a scalable version of the centralized solution presented in Section 9.2. Note that this design also facilitates cooperation between agents through the shared experience replay buffer [Nai04; OH19].

When using the separate policy paradigm, I train a separate deepRL for each of the M agents. In particular, each agent maintains a separate experience replay buffer in order to train the associated deepRL. At time t, the actions taken by all of the M agents (guided by different deepRLs) result in the same feedback, i.e., all the agents obtain the same reward at time t. Cooperation between agents is achieved only through the use of the common reward structure.

On the one hand, the centralized solution requires centralized training and centralized execution. On the other hand, both multi-agent solutions have distributed execution. Nevertheless, the shared policy relies on centralized training (i.e., partially distributed implementation), while the separate policy's training is distributed (i.e., fully distributed implementation).

Recall from Section 9.1.4, there are two reward models in order to account for different applications, (a) joint model and (b) contended model. In case of the joint reward model, it can be shown [Gre84] that solving the problem of finding the optimal policy (sequence of actions) for each agent separately amounts to finding the optimal policy for all the M agents combined. In particular, maximizing $\sum_{t\geq 0} \gamma^t u_m(t)$ separately, for each $1 \leq m \leq M$, amounts to maximizing $\sum_{t\geq 0} \gamma^t r_J(t)$ over time (recall that $r_J(t) = \sum_{m=1}^{M} u_m(t)$), which is easy to learn.

But with contended rewards r_C, one cannot make such claims. This is because, unlike the joint model, the utilities are mixed in a non-linear manner (using the max function) in order to obtain the reward at time t. Hence, that non-linearity in the contended model is more interesting to test the performance of a multi-agent solution.

9.4 Practical Baseline Solutions

I adopt two practical implementations to act as baselines to evaluate the deepRL solutions. First, a *carry-on* solution such that all users carry a microphone with themselves (implying $M = N$ and perfect motion control). Second, a *fixed* set-up, where the microphones are preinstalled and cannot be moved. The assumption that only one user is speaking at a time still holds.

Additionally, I compare the deepRL solutions to a greedy *heuristic* that controls the positioning of the microphone nodes. The heuristic checks all possible combinations of moving the vehicles and selects the one that maximizes the *instant* reward at the current state. Note that the run time complexity of the heuristic is not a part of the evaluation. The performance is evaluated just like the RL solution's performance (Section 9.1), so it is either the utilities from all microphones (joint) or the best utility (contended). This algorithm is considered greedy as it ignores the long-term utility maximization in favor of maximizing a per-time-step one.

9.5 Experimental Results

The initial simulation setup consists of two moving vehicles $M = 2$, each installed with a microphone. The centralized RL agent is trained for a speaker that moves in random trajectories (Section 9.1.1) while talking, where the average talking time is uniformly distributed between $\tau_{\min} = 8$ and $\tau_{\max} = 12$ time steps. Then, the next speaker starts speaking from a new random position; a previous speaker might have moved while being silent.

For the upcoming results and unless stated otherwise, I assume that $v_m \geq v_{\text{src}}$ and $w_\delta = 0.1$. In case of the fixed set-up baseline, the microphones are placed on a uniform equidistant grid, distributed evenly inside the room (20×20 m^2). The average speed of the speakers is 1m per time step. I show in [Haia; Haib] some videos to visualize the environment as well as the trained agents in different scenarios.

9.5.1 Convergence of training: Temporal Difference

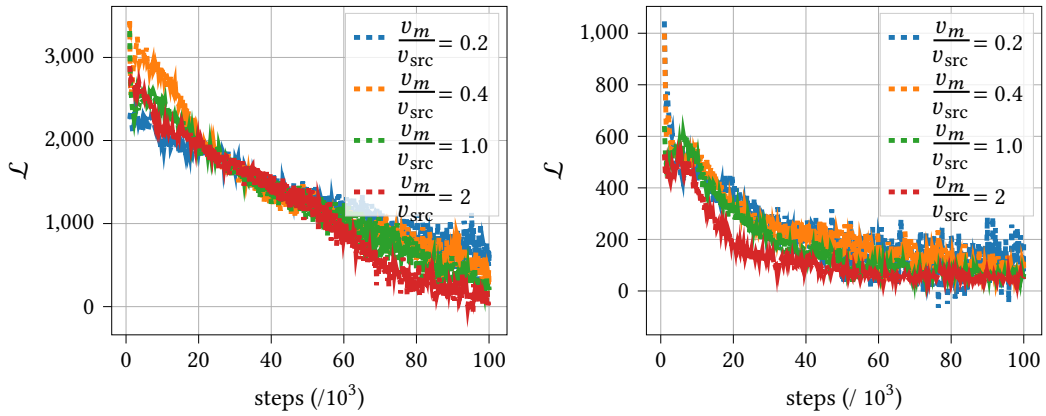
I use the temporal difference loss $\mathcal{L}$ (Equation (2.6)) of a single run (yet tuned over 50 runs) to visualize the convergence of the single agent while learning; the results are in Figure 9.3. Clearly, $\mathcal{L}$ decreases with more training steps for both joint (Figure 9.3 (a)) and contented (Figure 9.3 (b)) models.

Additionally, changing the relative speed of the vehicles $\frac{v_m}{v_{\text{src}}}$ (i.e., the vehicles move longer distances than the speaker per time step) changes the value to which $\mathcal{L}$ converges, for the contended reward model. In general, convergence will also change when varying the hyperparameters.

9.5.2 Performance of centralized RL vs. baselines

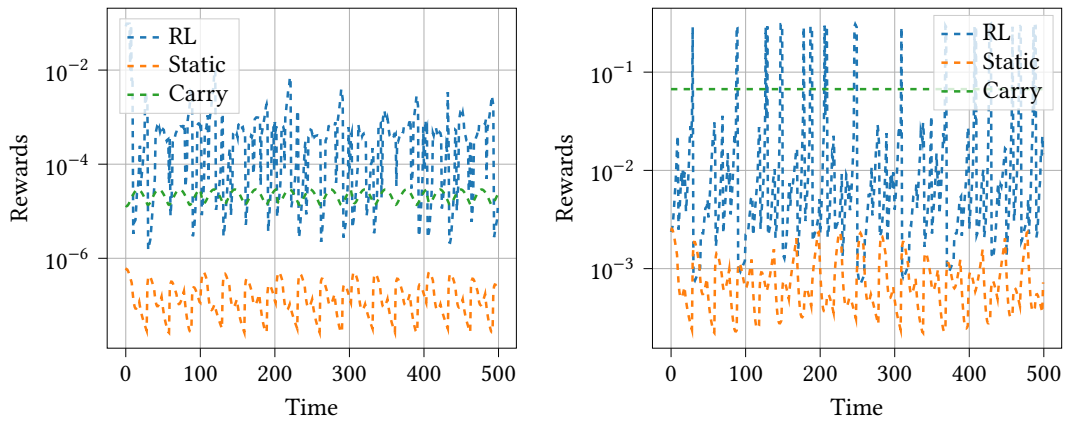
I compare the performance of the trained centralized single RL agent to that of the carry-on and static baseline solutions (Section 9.4). I assume that there are two speakers $N = 2$ and they take turns in speaking; both carry a microphone and move according to their predefined paths inside the room. These paths are selected to represent extreme cases moving near the edges of the room, while maximizing the distance between the speakers. Meanwhile, averaging over multiple paths is shown in Section 9.5.4.

For the joint model (Figure 9.4 (a)), the RL agent has on average better rewards/performance compared to both the static and carry-on solutions. This is due to additional degrees of freedom for the RL agent that can control the position of



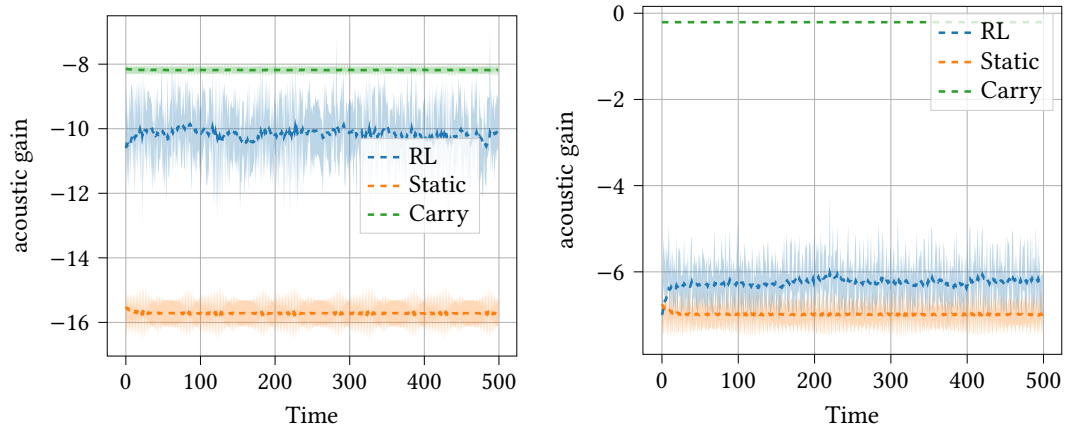
(a) joint model **(b)** contented model

Figure 9.3: Temporal difference error



(a) joint model **(b)** contented model

Figure 9.4: Comparison between centralized single-agent and the baseline models. Paths of the speakers are predefined to be as far as possible from each other, while $\frac{v_m}{v_{\text{src}}} = 0.2$.



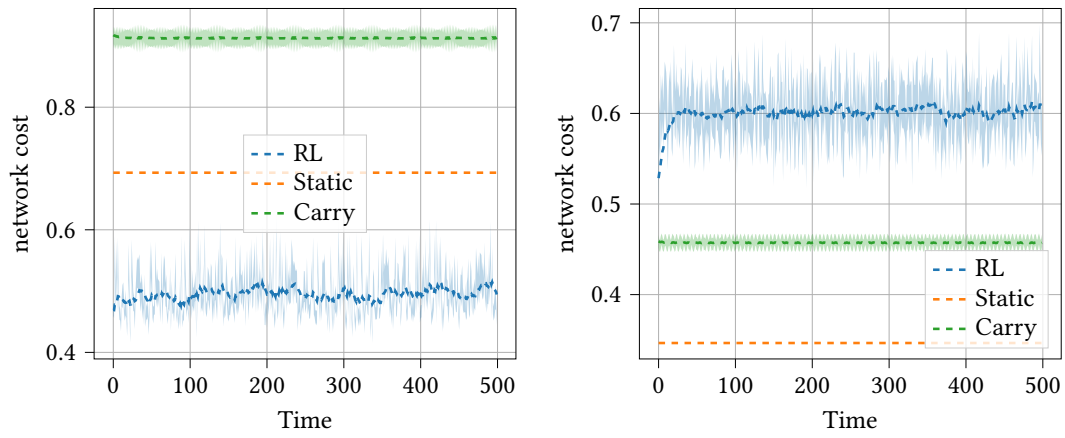
(a) joint model **(b)** contented model

Figure 9.5: Comparison between centralized single-agent and the baseline models with respect to acoustic gain

the microphones to be closer to the speaker, which results in a better performance than having only one microphone close to the speaker as in the carry baseline.

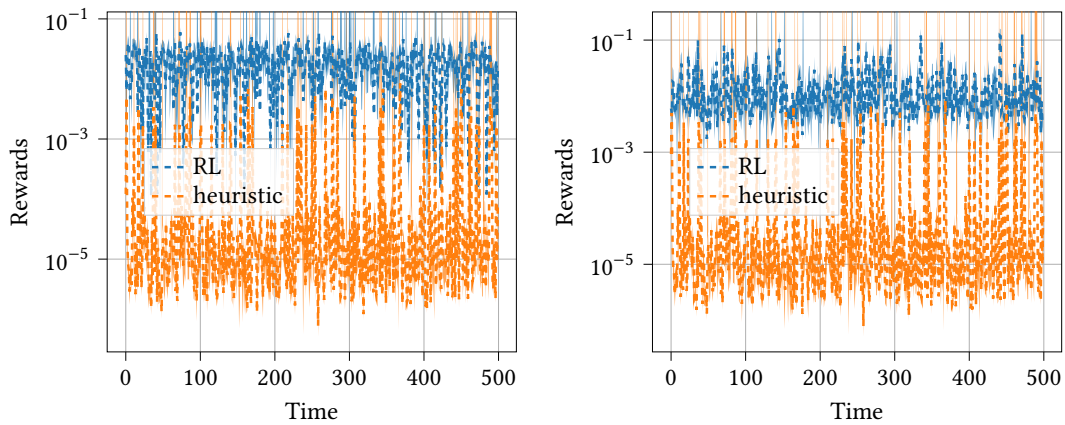
In the case of the contended model (Figure 9.4 (b)), carrying the microphone has the best acoustic performance (Figure 9.5 (b)), but not necessarily the best network one (Figure 9.6 (b)). Because microphones are wirelessly connected to the access point, the network performance degrades if the users carrying the microphones are far away from the access point. Accordingly, it is useful to sacrifice some acoustic performance in return for better network performance. Moreover, the RL agent adapts itself to the movement of the speakers to improve performance (acoustic and network) at some time steps, yet carrying the microphones seems to have on average the best performance here: remember that the acoustic gain is weighted more than the network one 10:1 and the speakers can cross paths close to the access point. It is worth recalling that, unlike the carry-on solution, the RL single-agent can scale for any number of N speakers, while still working with only 2 microphones.

The reason why carrying the microphone can have a high acoustic gain in the joint model (Figure 9.5 (a)) is due to the fact that two speakers may be near each other while carrying the microphone. Yet, still they do not account for how far they are from the access point; decreasing the network performance (Figure 9.6 (a)).



(a) joint model **(b)** contented model

Figure 9.6: Comparison between centralized single-agent and the baseline models with respect to network cost



(a) joint model **(b)** contented model

Figure 9.7: Comparison of single-agent RL with the heuristic model

9.5.3 Performance of centralized RL vs. heuristics

I compare the trained single agent from Section 9.5.2 against the greedy heuristic (Figure 9.7). Here I show the average reward over 50 different path, while the light shades depict the 90% confidence interval.

In the joint reward model (Figure 9.7 (a)), the heuristic solution has a high variance (as well as large confidence interval) since all vehicles follow the speaker. So the acoustic quality drops once the speaker changes and the microphones are far away from the last speaker. Meanwhile, the RL solution has smaller variance because it chooses actions that maximize the average reward and not just instant rewards.

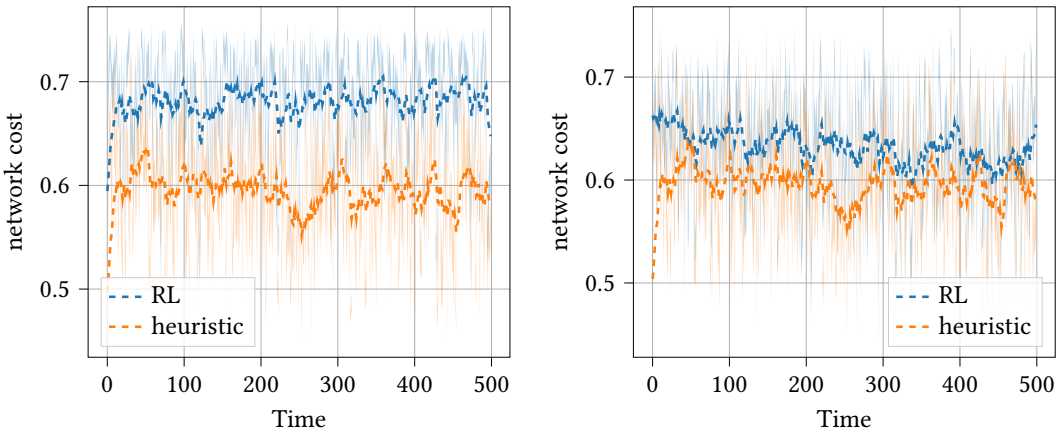
When looking at the contented reward model (Figure 9.7 (b)), the RL agent, again, has on average higher rewards than the heuristic algorithm, because it learns how to move the vehicles. When a new speaker pops up, one of the vehicles will be able to reach the new speaker faster than in the heuristic case.

In both cases of Figure 9.7, the relative speed $\frac{v_m}{v_{\text{src}}} = 0.2$. Although the RL solution has a higher reward than the heuristic one, the latter still has better results than the baselines at high speeds ($v_m \geq 2v_{\text{src}}$). At low speeds ($v_m < v_{\text{src}}$), this highly depends on the reward model. For the joint model, the heuristic still performs better than the baselines, while for the contended model the baselines (more specifically, carrying the microphone) perform better than the heuristic one. These differences are due to the fact that high speeds compensate for the greediness of the heuristic; the vehicles can reach the speaker quickly once it is active.

I show in Figures 9.8 and 9.9 the acoustic and network qualities of the heuristic and RL solutions separately. The RL solution performs better than the heuristic in terms of acoustic gain but the network cost is higher. Note that the acoustic gain has a higher weight than the network cost, si that RL still achieves in total a better performance than the heuristic one.

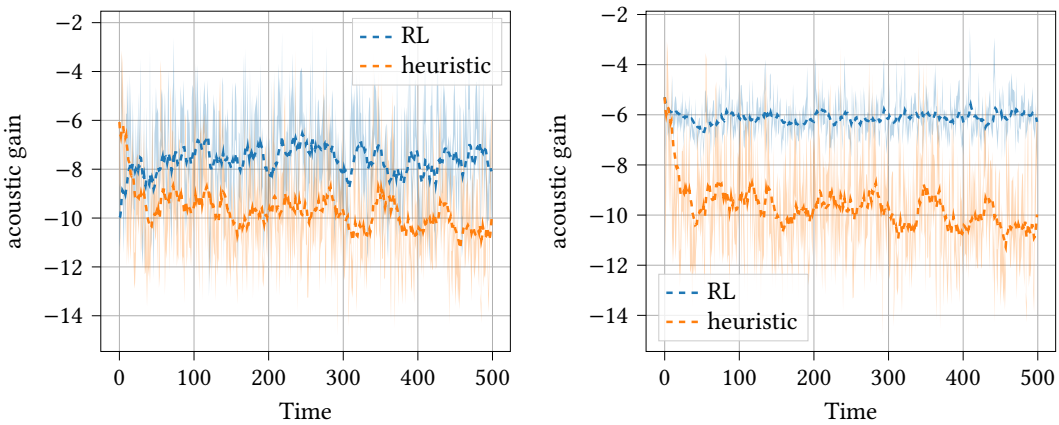
9.5.4 Performance for Varying Vehicle Speeds for Centralized RL

Now, I look into how speed will impact the learning process of the RL agent as well as its impact on performance. I start with the impact on convergence point (Figure 9.10) by varying $v_m \in \{0.2, 0.4, 1, 2\}v_{\text{src}}$. For slow moving vehicles –relative to the speaker– the achieved reward is lower than with vehicles moving at high speed; when $v_m \geq 2v_{\text{src}}$ the RL agent has the highest rewards for both joint (Figure 9.10 (a)) and contended (Figure 9.10 (b)) reward models. Additionally, while



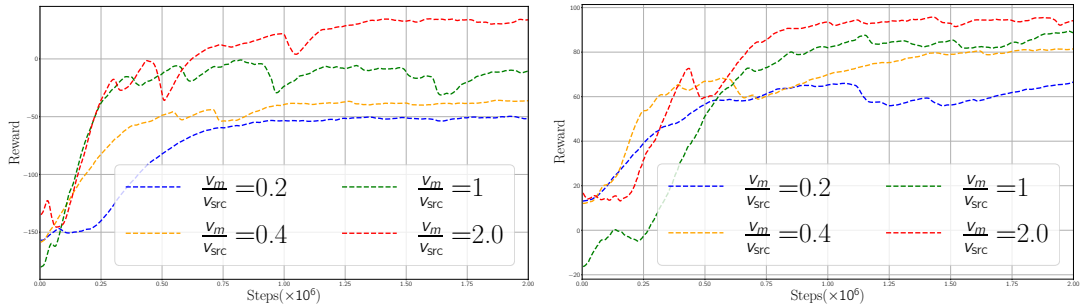
(a) joint model **(b)** contented model

Figure 9.8: Comparison of joint model with respect to network



(a) joint model **(b)** contented model

Figure 9.9: Comparison of joint model with respect to acoustic gain



(a) joint model **(b)** contented model

Figure 9.10: Convergence rate while learning with respect to the speed

training, RL agents with high-speed vehicles reach higher rewards in fewer number of training steps.

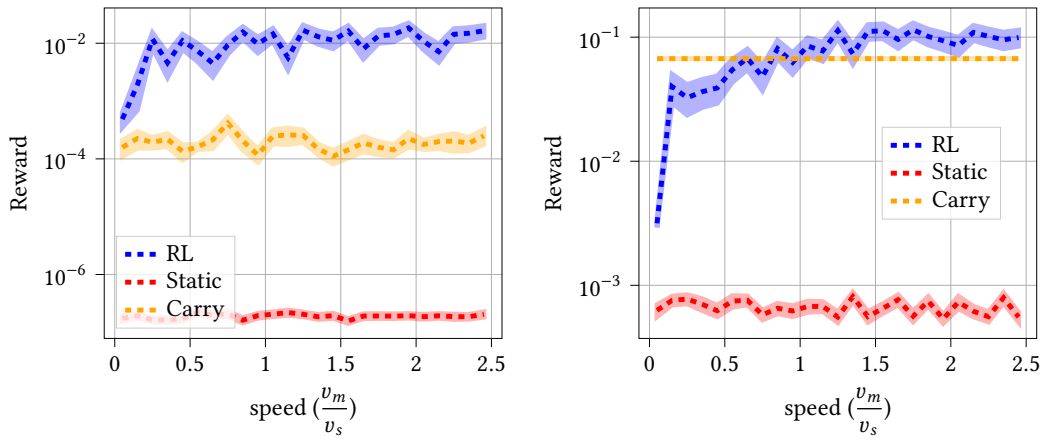
Next, I look into the impact of speed on the reward by repeating the experiment in Section 9.5.2 for different environments with different vehicle speeds. But this time, the speakers move within 50 different regions of interest and I show the results of averaging over 50 runs. Note that now the speakers may cross over between their regions and they are not necessarily far away from each other as in Section 9.5.2. Next, I compare the average reward (as well as the confidence interval) of a trained single-agent RL model to that of carry-on and static baselines (Figure 9.11).

When using the joint model (Figure 9.11 (a)), the RL solution has a better performance than the baselines. Again, this is due to the degree of freedom of moving the vehicles. But this is not enough for the contended model (Figure 9.11 (b)). At low speeds, a vehicle cannot catch up with a moving speaker, thus carrying the microphones is better. Nevertheless, as the speed of the vehicles increases (e.g., $v_m = v_{\text{src}}$) the performance of a single-agent RL becomes better than carrying the microphone.

9.5.5 Changing the Microphone Speeds for Multi-agent RL

Extending the analysis the previous speed analysis, I investigate the impact of changing the vehicle's speed on the multi-agent solution. Recall from Section 9.3 that a multi-agent environment only supports the contended reward model and two different policies: shared and separate.

First, I look at the value to which the mean reward converges for both policies (Figure 9.12). On one hand, the shared policy is sensitive to the microphone's speed; the mean reward increases with the microphone's speed. On the other hand, the



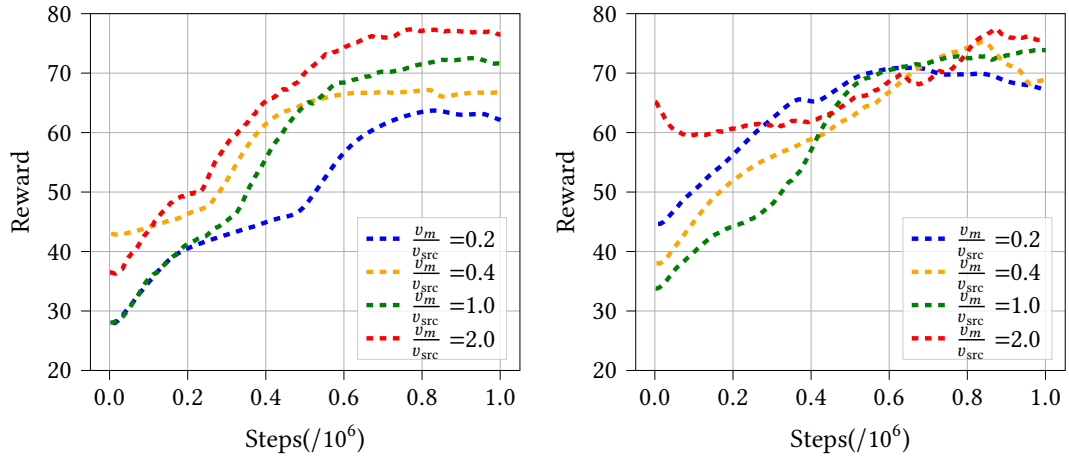
(a) joint model **(b)** contented model

Figure 9.11: Comparing single-agent RL to baselines with respect to the speed. Rewards are evaluated over 50 different paths per each speaker, while the shaded parts show the 90% confidence interval.

separate policy's reward is less sensitive to the microphone's speed, yet, the mean reward and microphone's speed are also directly proportional.

Additionally, at high speeds ($\frac{v_m}{v_{\text{src}}} = 2$), the shared and separate policies have almost the same value of converged mean reward. But at low speeds ($\frac{v_m}{v_{\text{src}}} = 0.2$), the separate policy converges to a higher reward than the shared policy. In conclusion, a task-specific policy (as in separate policy) may be better than having a global policy (as in shared policy) depending on the parameters in the environment.

The reason why a separate policy performs better than a shared one is indeed an inherited property from modular hierarchical learning [SPS99]. The formulation with partial observations and unknown variables (e.g., the position of the other microphones) is a complex environment, which is hard to solve with a single monolithic policy [AKL16]. Hence, hierarchical learning can be used to divide the main objective into sub-goals or sub-policies using fined-grained [Soc+12] or structured supervision [AKL16]. In this formulation, the shared policy is equivalent to a global monolithic policy and the separate policy is equivalent to learning sub-policies. In contrast to previous work, all agents have the same reward functions while they decide their own policies.



(a) Shared policy **(b)** Separate policy

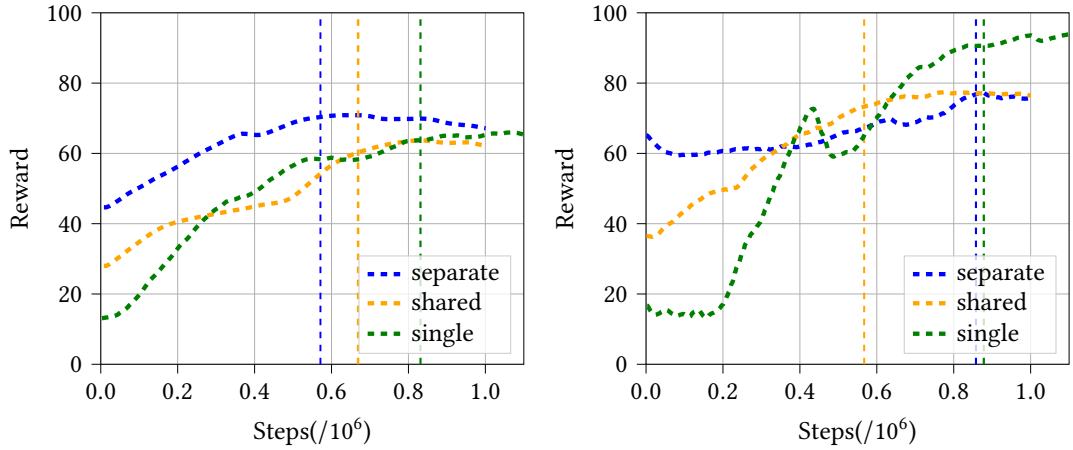
Figure 9.12: Average reward with respect to the speed

9.5.6 Single- vs. Multi-agent RL

Moving forward, I compare the convergence of multi-agent solutions to that of the centralized with contended utility (Figure 9.13). For the sake of a clear visibility of the plots, I show the convergence at low ($\frac{v_m}{v_{\text{src}}} = 0.2$) and high ($\frac{v_m}{v_{\text{src}}} = 2$) speeds. The vertical lines represent the point of convergence for each solution; I define the convergence point as the point after which the reward changes with a variance less than 2.

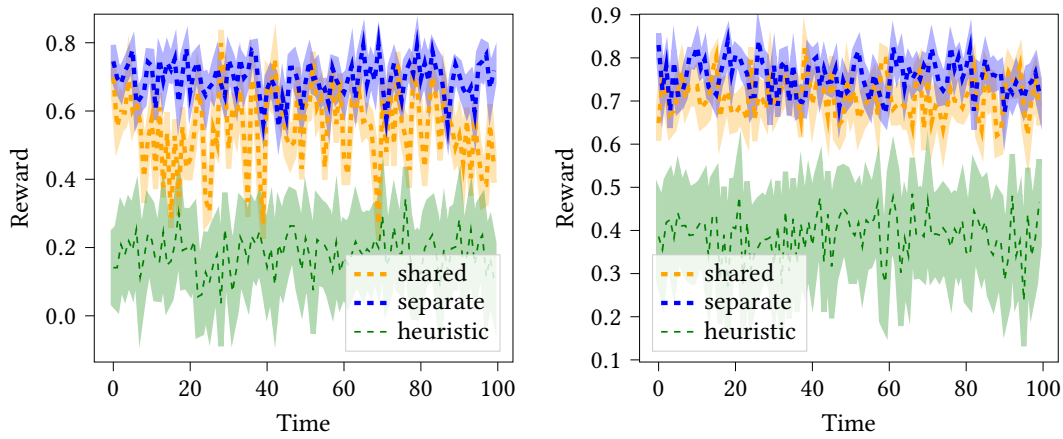
Similar to the results in Section 9.5.5, for low speed, the separate policy has slightly higher rewards than the shared one, while the centralized solution is very similar to the separate one (Figure 9.13 (a)). For high speed (Figure 9.13 (b)), the separate and shared policies have similar reward but the centralized solution performs better (i.e., higher rewards) than either other policy.

Likewise, I compare the converged mean reward of the multi-agent policies to that of the heuristic (Figure 9.14). For both slow and high speeds (Figure 9.14 (a) and Figure 9.14 (b) respectively), the averaged reward for multi-agent's policies is higher than the heuristic's one. Hence, not only the centralized single agent but also the multi-agent RL solutions learn how to move the vehicles better than the heuristic solution.



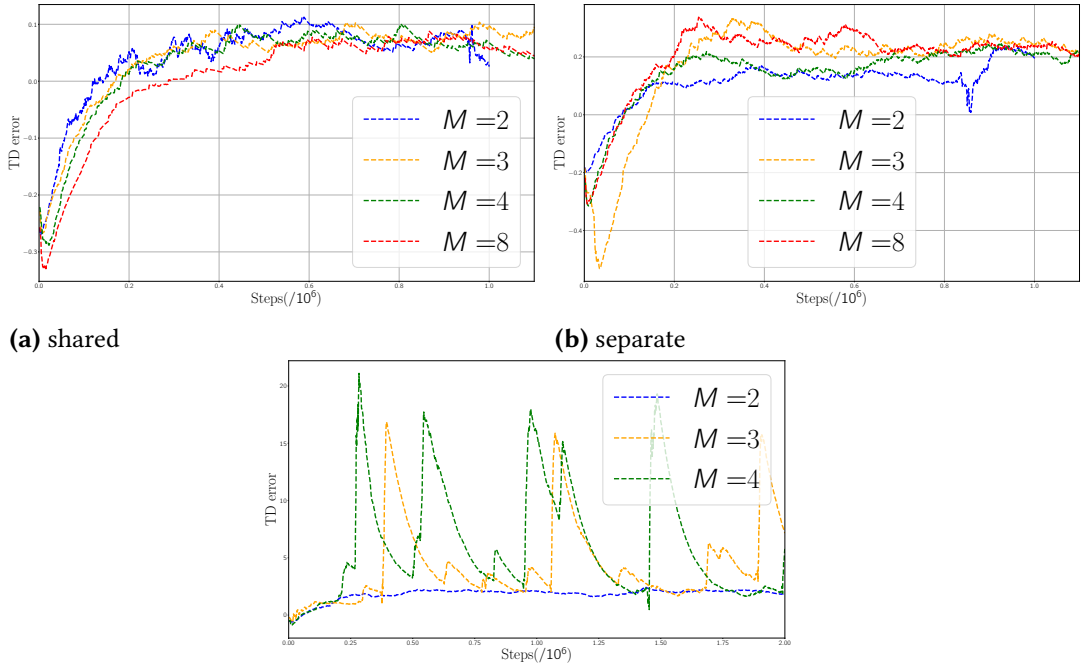
(a) low speed **(b)** high speed

Figure 9.13: Average reward with respect to the policy



(a) slow speed **(b)** high speed

Figure 9.14: Multi-agent vs Heuristic over 50 different runs. The shaded curves depicts 90% confidence interval



(a) shared

(b) separate

(c) centralized

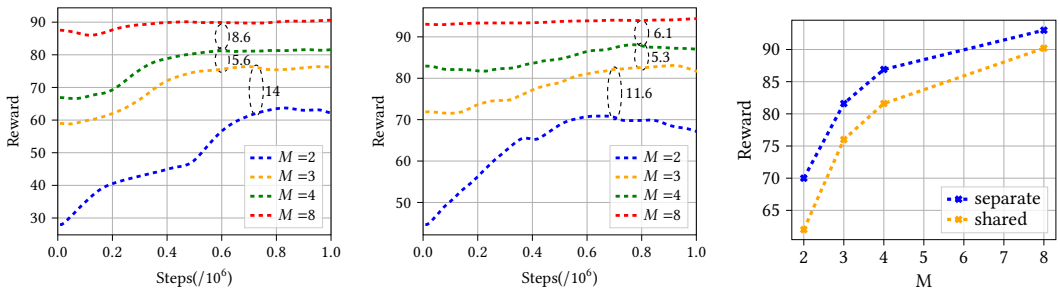
Figure 9.15: Temporal difference error for up-scaling

9.5.7 Up-scaling with multi-agent RL

The centralized single-agent RL solution has indeed better results compared to multiple agents, yet it is not scalable; the more microphones it has, the longer the training will be (recall that action space = 9^M). Meanwhile, a multi-agent approach can better scale up while sacrificing a bit of performance compared to the single-agent RL.

Figure 9.15 shows how the multi-agent solutions scale up with the number of microphones and converge for both policies. Given that an agent does not know about the existence of other cooperating agents, convergence is achieved by averaging the losses over multiple steps per a single state s. In contrast to the multi-agent RL solution, the centralized one shows multiple spikes as the number of microphones increases, which clearly indicates that the agent has not converged. Due to the exponential increase in the action space with respect to the number of microphones, I drop the convergence results for 8 microphones, since it is not feasible on the available training resources: memory (64 GB RAM) and CPU (16 cores Intel Xeon with 2.3 GHz clock frequency).

Next, I look at the gain in reward when increasing the number of microphones.



(a) shared **(b)** separate **(c)** reward

Figure 9.16: Average reward with respect to the number of microphones M

The shared policy (Figure 9.16 (a)) converges to higher rewards as the number of microphones increases. However, the reward gain shows diminishing returns: increasing from 2 to 3 microphones yields a gain of 14, while doubling the number of nodes from 4 to 8 yields only a gain of 8.6.

A similar behaviour is observed for the separate policy (Figure 9.16 (b)); the gain increases with more and more microphones, yet the gain in reward decreases as the number of microphone increases; moving from 2 to 4 microphones gains 11.6 reward, while doubling the number of nodes from 4 to 8 gives only 6.1 gain. The decrease in gain can also be depicted in Figure 9.16 (c).

9.6 Theoretical Discussion

I show how a dynamic environment impacts the behavior of both deepRL and heuristic models, where a generic theory of this impact is discussed in [Ram20]. Here, I show the relation between changing some of the attributes of the environment and the changes in deepRL behavior. For simplicity in notation, I drop the time index from $s(t)$, $a(t)$ and $r(t)$, and merely use s, a and r, respectively. Wherever necessary, I explicitly revert back to the notation involving the time index t.

9.6.1 Impact of Environment Setup on the deepRL Training

As briefly stated earlier (Section 2.3.4), the main objective of deepRL is to find the optimal state-action function $Q^*(s, a)$ (optimal Q-function). To do this, it in turn finds an optimal set of weights of a neural network, θ^*.

Now the question is, can any of the environment attributes (configuration of the environment) impact the convergence of RL? If yes, how? To answer these

questions, I first consider the expected value of squared Bellman loss as a reference Equation (2.5):

$$\mathcal{B}_t^2 = \left(r_t + \gamma \int Q(s_{t+1}, a_{t+1}; \theta_t) p(s_{t+1}|s_t, a_t) - Q(s_t, a_t) \right)^2 \tag{9.6}$$

where the integral is taken with respect to the distribution of the next state $s\prime$. Ideally speaking, if $\nabla_\theta \mathcal{B} = 0$ for every state action pair (s, a), then the model has converged. This seems to depend on learning the kernel transition $p_\theta(s'|s, a)$ such that:

$$\begin{aligned} p_(s'|s, a) &\approx p(s'|s, a; \theta) \\ &= p\left(s'|s, \arg\max_a Q(s, a; \theta) \right) \end{aligned} \tag{9.7}$$

In other words, updating θ will alter the estimate of the transition kernel. However, this is not the only parameter that influences the estimation. Next, I show how the environment variables can influence the kernel transition estimation.

Let us consider the centralized solution. A state s is defined by the position of the speaker x_{src} and the position of the microphones x_M. The speaker position x_{src} does not depend on the microphone positions x_M, but vice versa, they do. Hence, Equation (9.7) becomes:

$$\begin{aligned} p(s'|s, a) &= p(x'_{\text{src}}, x'_M | x_{\text{src}}, x_M, a) \\ &= p(x'_{\text{src}} | x_{\text{src}}) p(x'_M | x_{\text{src}}, x_M, a) \end{aligned} \tag{9.8}$$

I assume that the speakers are inside a square room, with dimension $z \times z$ (Figure 9.17). The initial position of the speaker is assumed to be a random variable that is uniformly distributed across the room. Then, its position changes in accordance to the following transition probability (within an area of $4v_{\text{src}}^2$):

$$p(x'_{\text{src}}|x_{\text{src}}) = \begin{cases} \|x'_{\text{src}}\|_2 \left(\frac{\eta}{4t^2 v_{\text{src}}^2} + \frac{1-\eta}{z^2} \right), & \|x'_{\text{src}} - x_{\text{src}}\|_2 < \sqrt{2} t v_{\text{src}} \\ \frac{\|x'_{\text{src}}\|_2}{z^2}, & \text{otherwise} \end{cases} \tag{9.9}$$

where η is the probability that the same speaker is talking. In simple words, if the distance between the new position of the speaker x'_{src} and the old one x_{src} is less than the distance that a speaker moves within 1 time step, i.e., v_{src}, then two cases are possible. First, the same speaker is talking and has moved with average speed v_{src}

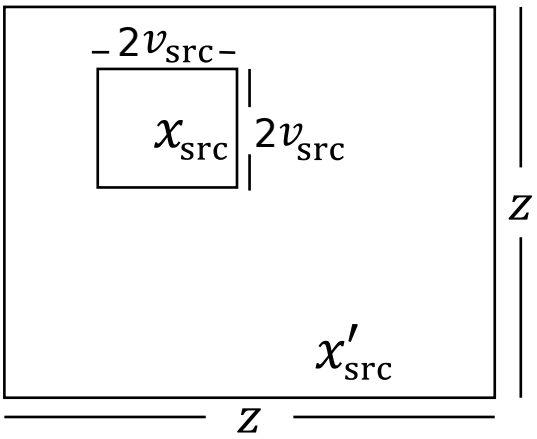


Figure 9.17: Depicting the attributes for the environment. In this example, x_{src} and x'_{src} are the positions of two different speakers, because the distance between them is bigger than v_{src}

in either horizontal, vertical or diagonal directions, i.e., $p(x'_{\text{src}}|\text{same src}) = \frac{\|x'_{\text{src}}\|_2}{4t^2 v_{\text{src}}^2}$. Second, this is the position of a new speaker, which is uniformly distributed inside the room, i.e., $p(x'_{\text{src}}|\text{new src}) = \frac{\|x'_{\text{src}}\|_2}{z^2}$. If there is a counter $\omega(t)$ that resets each time a new speaker starts talking, then η is

$$\eta(t) = p(\tau > \omega(t)) \tag{9.10}$$

Hence, I have shown theoretically how the room dimensions, speaker's speed and talking time (Equation (9.9) and Equation (9.10)) impact the behavior of the trained model; the bigger the room is, the slower the training convergence is and probably the performance of RL will decrease, while the faster the vehicle is, the faster is the convergence and the better the performance is. Consequently, and without loss of generality, the system dynamics characterize the behavior of the training process. A takeaway message is, for short talk duration Equation (9.10) and slow speed Equation (9.9), learning the best policy becomes very hard for the RL agent. As an example, I have shown empirically how the relative speed of the autonomous vehicle, with respect to the speaker's speed, impacts the training phase (Section 9.5.4).

Furthermore, from Equation (2.5), the discount factor γ is another parameter that influences the converged RL policy. Indeed, γ is tightly related to the talking time τ and defines the scope over which the agent maximizes the rewards. To explain

more in detail, I define the discounted reward $G(t)$ as

$$R(t) = \sum_{k=0}^{\infty} \gamma^k r(t+k), \tag{9.11}$$

then relate it to the time horizon τ, so that

$$R(t) = \sum_{k=0}^{\infty} e^{\frac{-k}{\tau}} r(t+k) \tag{9.12}$$

where $\gamma = e^{\frac{-1}{\tau}}$. When $k \geq \tau$, the contributions of future rewards decreases exponentially. By picking $\tau = \infty \implies \gamma = 1$, I model a system with only one speaker speaking all the time. When $\tau \approx 0 \implies \gamma \approx 0$, I model a system where there are multiple speakers, each speaking for a very short duration, so that the objective is to maximize the reward for this short duration. Consequently, the discount factor should be carefully selected with respect to the talking time, e.g., close to $e^{\frac{-1}{\tau}}$; otherwise, the RL solution will converge to a sub-optimal one.

9.6.2 Heuristic Sub-optimality

When considering the joint reward model, the proposed heuristic is indeed equivalent to solving a Markov decision problem associated with the discounted reward problem with a discount factor of $\gamma = 1$. This explains the similarity in the performances of the heuristic and the RL model in Figure 9.7. To elaborate more, let us substitute in Equation (9.11) $\gamma = 0$, then the discounted return $Q^*(s_t, a_t) = r(s_t, a_t)$. Hence, in Q-learning, I end up picking actions that maximize the rewards, i.e., $\underset{a}{\text{argmax}}\, r(s, a)$. This is also, roughly speaking, the strategy of the heuristic baseline, which moves all microphones towards the speaker, regardless how long the speaker has been talking.

Another takeaway to consider if the heauritic will be used, as the values of τ and v_{src} decrease, the speakers' positions will start bouncing inside the room (e.g., seminar rooms where alternating speakers normally do not change their places). This would obviously result in a poor performance using the heuristic solution. Meanwhile, the RL solution is generic and can be retrained and adapted to the changes in the environment.

9.7 Summary

I showed in this chapter how autonomous vehicles improve data acquisition in wireless sensor networks, where the results are supported by simulations and theoretical discussions. I compared the performance of RL-controlled autonomous vehicles to current solutions (called baselines) as well as a heuristic one to show that the proposed solution achieves very good performance and more flexibility.I showed via theoretical analysis how the parameters in a dynamic environment (such as v_{src}, τ and room dimensions impact the RL behavior.

The centralized RL solution is not scalable. Hence, I propose two multi-agent RL solutions with two different policies. One common property for these formulations is that each agent takes an action independent of the actions taken by other agents. This will avoid the need for frequent exchange of information between the agents, resulting in a quick decision making for moving the vehicles. The multi-agent solutions rely on partial information of current states while the action space per agent is the same for any number of nodes. Therefore, their performances are slightly lower than the centralized solution, which is the price they pay to easily upscale for different number of vehicles.

Both multi-agent policies scale with respect to the number of vehicles, but each has its own advantages with respect to the specific application and available resources. On the one hand, the shared policy provides a generic model that can be used by any node in the network. Hence, newly joining nodes can use the same trained model and improve their performance over time. This requires, however, sharing experience from all other nodes, which yields two problems. First, it raises privacy concerns since experiences will be visible to other agents. Second, it will not allow online learning since all experiences need to be shared on one central server, processed and then update the trained model. The fact that there is a round trip between the agent and the central server (sending training data and receiving a new model) will slow down training process due to the communication delay. Note that the communication delay for sending the trained data is different from the transmission time for streaming the data.

On the other hand, the separate policy learns using its own experience with no need of sharing experience with other nodes. Hence, it is easier to use for online learning without privacy concerns. A challenge here is that the learned models could be asymmetric, meaning that each agent learns to be task-specific. The problem then appears when a new agent (i.e., a new vehicle) joins the network, what is the new agent's model? Training the model from scratch is impractical because it may take long time for each agent to find its new task. Accordingly, this

might introduce instabilities in the performance of the multi-agent solution during the training phase, when switching from the old tasks to the new ones.

Part IV

Proof of Concept

The previous parts were solely based on simulations. Hereafter, a test-bed (hardware+software) [Afi+18] is proposed that inherits some of the properties of Part II. I focus here on the software framework and its architecture. The framework allocate jobs to raspberry pi nodes, where the jobs could be python modules or any executables. An additional feature is supporting fail-over mechanism when nodes are no more reachable due to hardware or network problems.

The framework is used then with different acoustic applications such microphone synchronization, scene analysis and privacy applications. The main objective is to show that it is possible to run these applications locally using in-network processing without relying on third parties or a cloud server. Evaluating these applications' algorithms is out of scope of this work. The framework can be used for any other application and not solely acoustic applications.

10 Framework for In-network Processing

The current progress in hardware development of embedded devices offers more performance at reduced costs. Example are neural network-empowering chips, like the Tensor Processing Units (TPUs) from Google or the Bionic SoCs from Apple, which enable the local processing of large neural networks on embedded devices.

Recall from Part II that there are multiple advantages of processing the data on local devices rather than transferring them to a cloud. But since these devices may not be powerful enough to run the whole processing on a single device, the processing is divided into multiple jobs and a device processes one or more jobs. Previous chapters have dealt with algorithmic questions how to decide such divisions. But doing that practically is still a challenge.

In this chapter, I provide the prototype of a framework – called "MARVELO" – that deals with such practical concerns. In particular, it allocates and executes jobs to and on nodes (i.e., the embedded devices). I assume that an application is implemented in a block-oriented fashion (e.g., [Sch+17a]), as is common practice for many online acoustic signal processing algorithms. These blocks, which from now on will be called jobs, represent parts of the application with standard operations, e.g., a Fast Fourier Transform (FFT), or even bigger algorithm parts like a gradient descent filter update. Hence, I look now at placing the jobs across the networking nodes.

The "MARVELO" framework does not split an application into jobs, so that the degree of freedom is defined by the given jobs. Therefore, all jobs are recommended to be kept as small as possible. "MARVELO" combines the concepts of two systems: distributed computing [Ng+20] and routing. When focusing on wireless distributed computing [AM06; Dat+12], such systems consider distributing jobs and servers and collecting their outputs. Hence, there is no interaction between the servers. Recent examples of such systems are DreamLab [Vod] and Folding@home [Gre], where they use the idle processing of embedded devices (e.g., smart devices) to help analyse and process complex data.

The routing concept is adopted from Software Defined Network (SDN); it relies on a controller (a.k.a client), which receives all networking updates (e.g., bandwidth utilization) from the networking nodes, then, it client builds a forwarding table and sends it to the nodes. The nodes use the routing table for data forwarding. SDN has been introduced to WSN in Sensor OpenFlow [LTQ12], Software Defined

Wireless Networks (SDWN) [Cos+12] and TinySDN [OMG14]. However, they were all limited to the routing functionalities and, unlike my framework, they do not support job placement.

More closely related to MARVELO, there have also been some ideas on using job distribution in WSN. PixieOS [Lor+08] collects information about the available resources at each node and allocates jobs accordingly. Apple has also proposed a patent [VL16] where some jobs (e.g., recording) can be activated or deactivated at nodes (mainly smart devices) in an infrared network. Nevertheless, they are concerned only with job distribution of *independent* jobs and do not consider the jobs to be interconnected or cooperating.

SDN-WISE [Gal+15] is an operating system framework that combines the SDN and job distribution concepts. It focuses, however, on networking applications (e.g., firewalls or proxies) and considers routing only for auto-configuration purposes.

10.1 Framework Overview

MARVELO framework combines the concepts of SDN and job distribution, so that it jointly considers job placement and routing. Moreover, it supports various types of applications and is not limited to networking ones. I focus here on WASN applications as a case study.

But first, I describe here the framework architecture and briefly show in Section 10.2 how to use it.

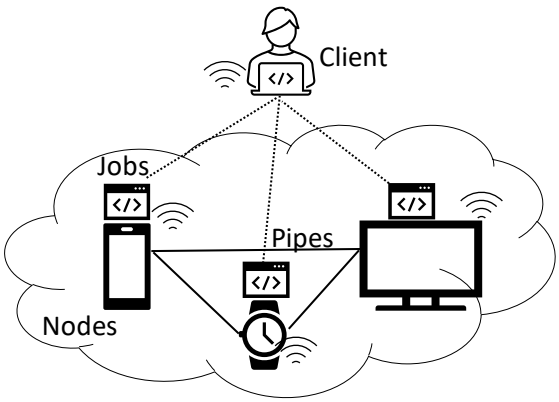


Figure 10.1: Framework Architecture

The framework follows a client-server architecture (Figure 10.1). A client requests the execution of application jobs from one or multiple servers. The *client* role is

hosted either on one of the wireless nodes or other external devices that are still connected to the network (e.g., a gateway or a cloud host). Meanwhile, the *server* role runs on the wireless nodes inside the network. Note that the implementation allows the server to run on nodes outside the network, but this property was not used in my work. The aim is to run an application (from the client) on one or more servers.

10.1.1 Client

The client role is responsible for placing the jobs and sending their dependencies (e.g., files and packages) to the servers. In principle, it has the application graph and an overview of the network status. The decision on which node to run which job can be taken by the client or manually given. Additionally, it collects log files for debugging purposes and controls the life-cycle (e.g., start, stop and restart) of the jobs. Furthermore, it builds the routing table (via BATMAN [Fre]), but the routes can also be manually given on the application level, if desired.

10.1.2 Server

The server role is given to the nodes that run one or more jobs, depending on the computation capacity and the client's decision. A server also keeps the client up-to-date with the status of running jobs (e.g., running or dead) as well as the status of the node itself (e.g., available CPU percentage, memory and disk space). Idle servers (i.e., not running any jobs) keep sending keep-alive messages to inform the client about their availability. This is useful in particular in case of fail-over.

10.1.3 Jobs

A job is a part of the application as defined in Part II. It can be any executable and is not restricted to a specific programming language. The job profile, such as CPU and memory usage, should be given to the client. Otherwise, default values are used for deciding job placement. Alternatively, a job can be given a default server to run on.

10.1.4 Pipes

For two jobs to exchange the output between each other, they need some form of interprocess communication. MARVELO uses Linux named pipes to do so.

Compared to other inter-processing communication alternatives [Din20], named pipes are faster and allow data exchange between different devices.

Here, pipes are data communication channels between jobs, where is the communication is assumed to be simple enough to use raw binary bytes. Additionally, "MARVELO" supports custom Pipes, so that not only can the data types be bytes, but data can be objects such as JavaScript Object Notation (JSON) and Pickle. Note that Pickles are only supported for Python-based jobs.

10.2 Implementation

The MARVELO framework is implemented with "dispy", a framework for executing multiple jobs in parallel across one or many machines. The main objective of *dispy* is to execute a job with different datasets independently *with no communication among jobs*. In MARVELO, we additional need communication between jobs and have to add it. Furthermore, extending dispy further, the framework mixes both parallel and sequential computations, i.e., allowing pipeline computations, and not only parallel ones. In the following, I highlight the most commonly used components and their features [Afi].

10.2.1 Attributes and Features

The implementation is based on three main classes: Job, Node and Pipe. A Job has three main attributes: GROUPS, DEFAULT_NODE and MAX_QUEUE. GROUPS, which is also used by Node, is a string (or a list of strings) that acts as a tag to group jobs with common requirements. For example, jobs that needs microphones can have a tag GROUPS = "microphone". Similarly, nodes that are equipped with a microphone, have GROUPS = "microphone". A job can have multiple tags and hence belong to multiple groups, e.g., GROUPS = ["microphone", "camera"]. By default, all jobs and nodes have GROUPS = "ALL", i.e., any Job can run on any Node.

Note that a job normally runs on a server node, but if there is a need to explicitly run a job on a client, there is a LocalJob class, which inherits all properties of Job, except that it can only run on the client. Such jobs may be useful for analysing data from other jobs or sending feedbacks.

DEFAULT_NODE is a string attribute that defines the IP address or the hostname of the node (i.e., server) that should be running the job. If this server is not reachable or does not have enough resources, the client will look for other nodes within the job's GROUPS.

MAX_QUEUE is an integer that keeps the most recent MAX_QUEUE input data (depending on the input data size) and delete old ones if the queue is full. It is optional and does not have to be used.

There are other optional Job attributes such as DEPENDENCIES and HEAD. The former is a string (or a list of strings) to point to the path (or paths) of files and packages that a job needs to run. For example, it could be an audio file that a job will load and process, or a neural network model that a job will load to infer some data. HEAD is a Boolean attribute; if it is True, it adds 8 bits (acting as flags) to the data to describe its status. It contains the following flags in this order: [Corrupted, Reset, Finished, Free, Free, Free, Free, Free]

- Corrupted marks a Dummy packet or a packet that resulted from a corrupted packet. A job can use such information to handle the packet in a special way or just drop it.
- Reset marks the sequence number is restarting at 0. It only can be set on a source node. It is helpful when a new source of data (e.g., a microphone) is selected; the sequence number is reset.
- Finished marks the packet to be the last one being processed, then the job will be shut down.
- Free flags can be set individually, e.g. for a job, if it need some extra flags.

Pipe has an important attribute, BLOCK_COUNT. It is an integer that defines the number of outputs that need to be buffered from the sending job before forwarding them to the receiving job. The default value is BLOCK_COUNT = 1. For example, if a job's output is 8 bytes and BLOCK_COUNT = 2, then 16 bytes will be buffered before sending the data. Note that in case of custom pipes (e.g., JSON and Pickle), 2 objects will be buffered before sending the data.

10.2.2 Job Placement and Redistribution

There are two options for job placement. If the job profiles (e.g., CPU and memory usage) as well as the data rate requirement between the jobs are given, methods discussed in Part II are used for generating a configuration file with job placement (i.e., GROUPS and DEFAULT_NODE). Otherwise, and this is the default option, default resource requirements are used (hard-coded in the framework) and the jobs are placed accordingly.

In case of node failures, jobs running on the failed nodes will fail-over to other available nodes *within the same* GROUPS. Selecting these nodes depends on available resources and the wireless link status between the possible new nodes and the old one running the other jobs. In case of wireless link failures (e.g., obstacles between nodes), the BATMAN routing protocols is used to find alternative routes [Fre]. Note that initial placement, when all network information is given, does not use BATMAN for routing decisions; these decisions are made by MARVELO, i.e., the optimization problems in Chapter 3.

10.2.3 Job Synchronization

For in-network processing, we need to share the pipe's state among all jobs, so that they can react accordingly. For example, if a packet is lost, not only does the next job, but all other next jobs need to know this information. This can be checked via a sequence number. Nevertheless, if the source job (e.g., microphone) has been restarted (or newly selected), the sequence number will be reset, so other jobs need to know this information as well.

In this context, a synchronization wrapper is built around each job on a node. This wrapper consists of two parts, *SQN-Check* dealing with the input data of a job and *Head-Check* dealing with the output data of the job (Figure 10.2). This wrapper can be deactivated (default).

The SQN-Checker checks the sequence numbers (SQN) and headers of the job's incoming data. In the case of a missing Block (i.e., **SQN is higher than expected**), a Dummy Block is created, either by creating a packet filled with zeros or reusing the data from the last packet. Selecting between both options is defined by setting the attribute mode. The actual packet is buffered until the SQN matches again, i.e., until the receiving and sending jobs have the same SQN. Here I chose to generate dummy packets instead of resending the packets, since the latter imposes longer delays (via acknowledgements and timeouts), which are not desirable for real-time applications.

If the **SQN is smaller than expected** and the reset flag is not set in HEAD, the packet is dropped. If the reset = True, the SQN will reset.

Head-Check has one main role, increment the SQN of outgoing packets. An additional role, which is exclusive to source jobs (i.e., the first jobs in the application graph), is to set the reset flag to True. This is useful in case of failover of source jobs. Then, the SQN needs to be reset since the source job will start from 1 while all other jobs already have high SQN. Otherwise, all packets from the source job will be dropped until SQN of the source job matches that of the others.

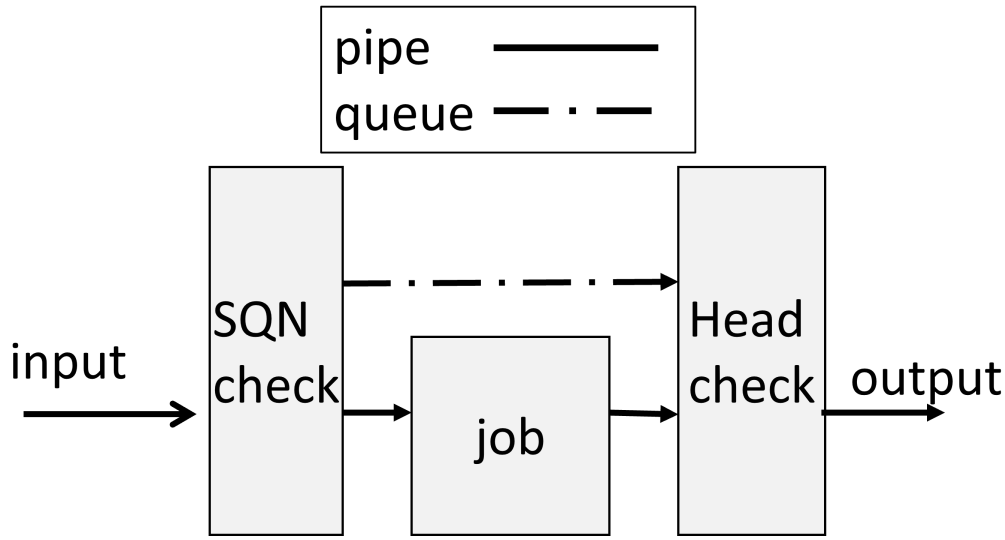


Figure 10.2: Visualization of the synchronization wrapper.

10.3 Show Cases

The framework has been seen in action in a show-and-tell special session at the Information Technology Society Conference for Speech Communication in Oldenburg, Germany (2018) and at the International Workshop on Acoustic Signal Enhancement (IWAENC 2022) conference in Bamberg, Germany. Here, I describe briefly the applications as well as the features that were used from the framework.

10.3.1 Job Distribution

The following applications have been used as a proof of concept of the framework functionality and usability. The main focus here is the capability of using in-network processing, while my main role in all of the following was setting up the network and assisting my colleagues with troubleshooting.

Scene Classification

The details behind the scene classification implementation [JA22] have been discussed in [Ebb+18]. There are mainly two objectives of this application. First, sound recognition (e.g., speech or music) for the input audio signal. Second, it provides starting and ending times for each recognized sound. Moreover, it supports poly-

phonic sound, e.g., it detects multiple events (e.g., clapping and whistling) at the same time.

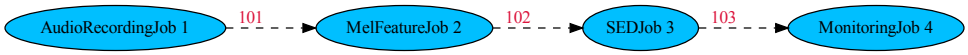


Figure 10.3: Sound event detection graph [Ebb+18] generated by the framework

In Figure 10.3, I show the application graph developed by Janek Ebbers [JA22], yet visualized by the my framework. The application consists of 4 jobs: *AudioRecordingJob* is used for streaming audio data from the microphones. *MelFeatureJob* extracts audio features and forward them to *SEDJob* for sound event detection. The output is then visualized via the *MonitoringJob*.

Privacy Preserving Audio Classification

The audio features used for scene classification may also carry a significant amount of speaker-dependent data, which compromises the speaker's privacy and makes the system vulnerable to speaker recognition attacks. Hence, the work in [Ale22; NM21] introduces a variational information feature extraction scheme that allows sound classification while minimizing the feature representation's level of information that is needed for speaker recognition.

Figure 10.4 shows the privacy application graph developed by Alexander Nelus. *MicGenerator* is a job that can either collect audio data from the microphone or forwarded previously recorded audio data. The *FeautureExtractionJob* is responsible for the variational feature extraction and forward the output to the scene classification job (*TrustJob*) and the speaker recognition attacker (*ThreatJob*).

Coherence Drift Based Sampling Rate Synchronization

When collection audio data from multiple microphones, there is the challenge of having different sampling clocks between the microphones. This yields many problem such as the illusion of have a moving audio source. To overcome this problem, this demo uses an algorithm [GSH21] as well as a hardware [Afi+18] that were developed by my colleagues, which can control the sampling frequency of the microphones.

The application graph of the algorithm is shown in Figure 10.5. *AudioSourceJob* is running on two nodes: one with a normal microphone and the other can control the

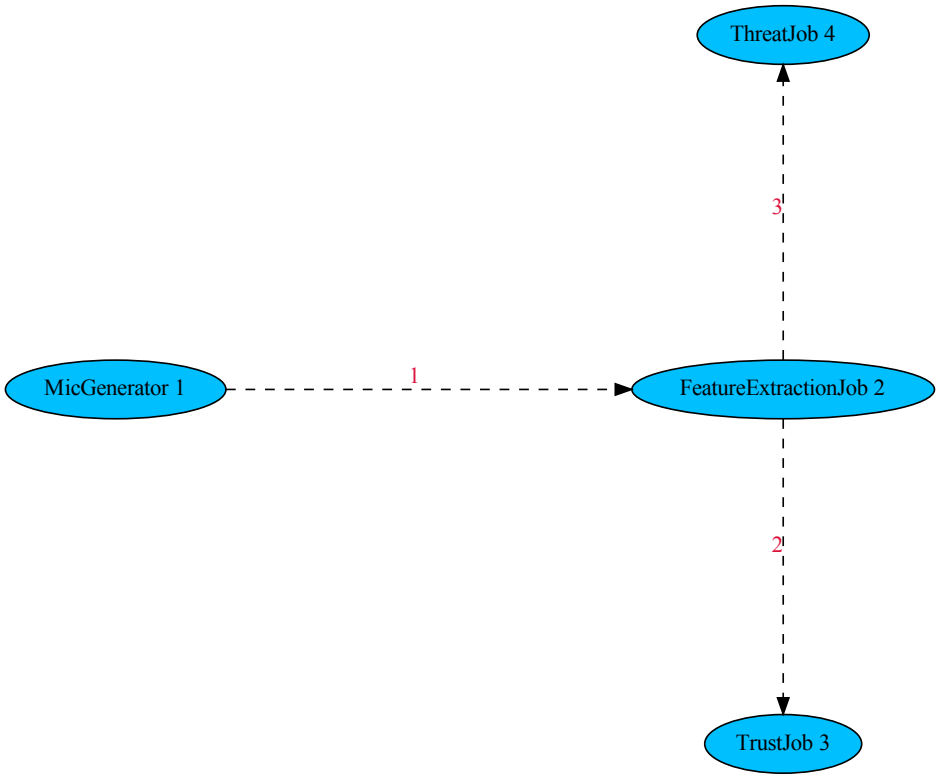


Figure 10.4: Privacy preserving graph [NM21] generated by the framework

sampling rate via *HWFreqChangeJob*. The audio data is processed by *CoarseSyncJob* and its output is forwarded to *SROJob* and *DelayJob* to estimate the SRO and delay between audio data, respectively. *MonitoringJob* is used to visualize the SRO and delay in real time.

[GSH21]

10.3.2 Failover in Wireless Distributed Computing

Failover in wIreleSS dIstributed cOmputiNg (Fission), which means division or splitting into two or more parts, is an ad-on property to the framework that allows failover of a job from one node to another Section 10.2.2. This is useful in particular

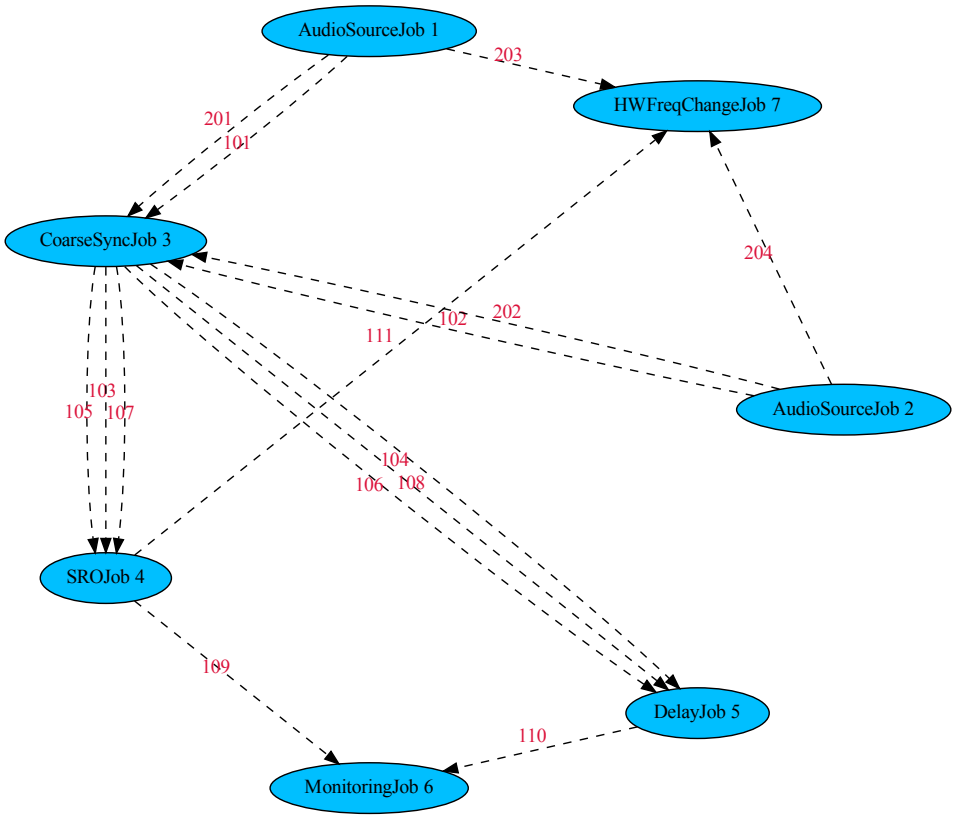


Figure 10.5: SRO estimation graph [GSH21] generated by the framework

when a node leaves the network (e.g., a node failure or a smart phone leaves the room).

Fission has been successfully shown in a live demo with Section 10.3.1 and Section 10.3.1 demos, where a node fails (i.e., restarting a node running one or more jobs) and other node(s) takes/take over. Demo [TA22] presents a unique fail-over feature; when a microphone is dead, another microphone is selected and the application is still able to synchronize the sampling frequency with the new microphone. As a take away message, Fission is a generic framework that can work with different applications having different properties (e.g., interacting with hardware). My main contribution here was developing the framework but not the acoustic applications themselves.

11 Case Studies

Use cases were used as examples of application with different functions that can use my framework smoothly. Here, I use case studies to select a couple of applications to study delay of in-network processing. Although I claim that executing jobs distributed on multiple nodes can be faster than just forwarding the raw data and processing it on a single node, these jobs need to be carefully distributed without over-utilizing any node or the wireless network. Otherwise, the (processing and/or communication) delays could be higher or the throughput might be inadequate than what would be the case by just forwarding the data and processing it remotely [Afi+18]. Here, I use the framework to evaluate the end-to-end delay of acoustic applications in different distribution scenarios.

One of the major contributors of the total delay is the wireless network topology. The most common topology nowadays is the star topology; nodes are connected to an Access Point (AP) and the communication between any two nodes goes via the AP. In contrast to the star topology, a mesh topology allows direct communication between the nodes without relying on an AP.

In any of these topologies, there are multiple sources of delay, especially when it comes to in-network processing. From the literature, key aspects are the channel access delay [AYY02; RLH15], the processing delay [Afi+18; Gon20] and the transmission delay [WSM08].

Without loss of generality, I explain in Figure 11.1 the sources of delay and how they relate to end-to-end delay in in-network processing. I assume that there are 3 wireless nodes: Node X, Node Y and a Gateway, where Node X and Node Y are running Job A and Job B, respectively. Furthermore, I assume that the output from Job A needs to be processed by Job B, whose output is forwarded to the gateway. The processing delay of a job (black region) is the time needed by a node to finish processing this job. It depends on the job's complexity and the processing capabilities per node. After processing the first packet at Node X, the output is forwarded to Node Y.

Transmission delay between nodes X and Y is the time needed to transmit the packet from Job A (gray region) to Job B. There are two main factors that affect the transmission delay: wireless channel condition [Che+09] (influencing data rate and the required number of retransmissions for a packet) and routing (influencing interference from neighboring transmissions and not the multi-hop itself). The authors

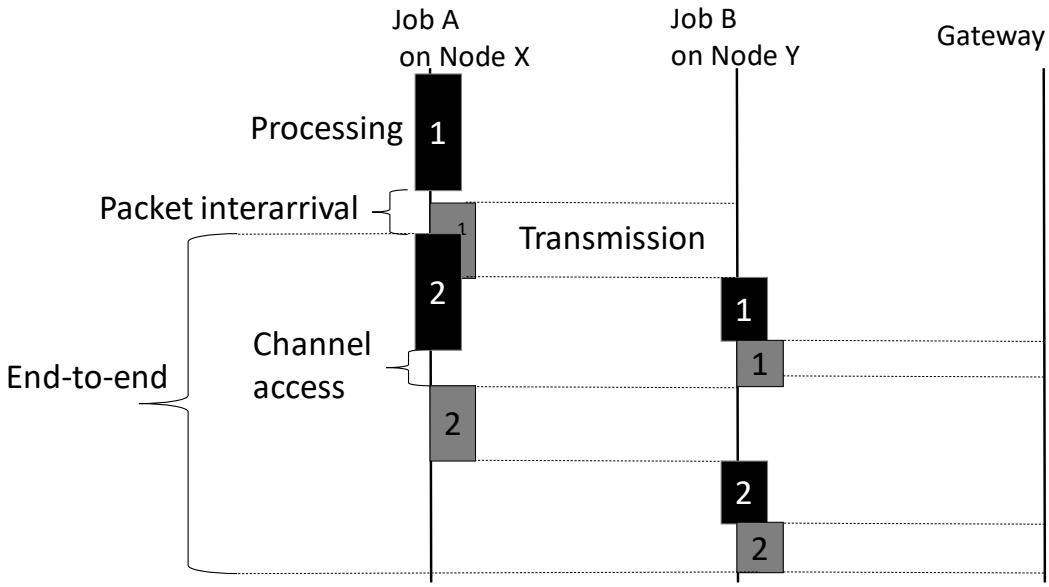
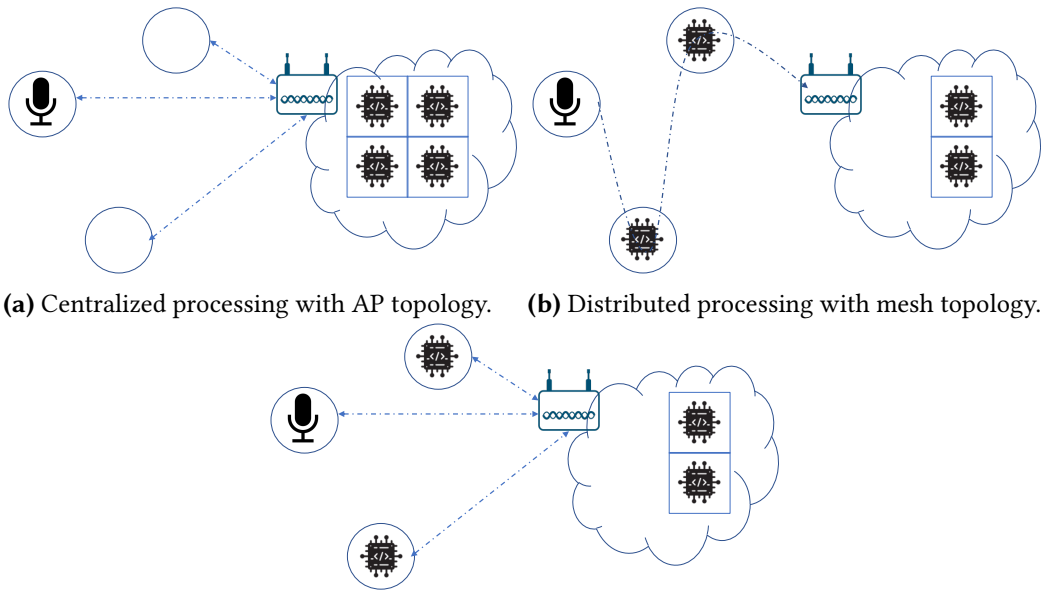


Figure 11.1: A snapshot of delays in wireless in-network processing.

in [WSM08] considered a single-hop network for data transmission, assuming that all nodes are in mutual communication range. Meanwhile, the work on multi-hop delay estimation has been theoretically and analytically studied in [CS15; Kan+20], but there are no experimental studies, especially that encompass transmission and processing delays for acoustic applications in the context of in-network processing.

After processing the second packet at Node X, the node cannot directly send to Node Y, since the channel is not yet free. This is an example of channel access delay, where a node has to wait because the wireless channel is busy (Node Y utilizes the channel to send to the Gateway) and the receiver cannot be sending and receiving at the same time (duplex constraints). Another type of delay is the packet inter-arrival delay, where the first Job needs to wait for receiving the next Packet for processing. In acoustic applications, this can be controlled using the microphone's sampling rate, which control the time needed creating samples that are then encapsulated in a packet. Last but not least, end-to-end delay is the aggregation of all previously mentioned delays, i.e., the delay starting from processing a packet at the first job till receiving its processed output from the last job at the gateway.

Here, I strive to close the gap between theory and practice: I empirically evaluate the end-to-end delay of different acoustic applications using the framework and compare it to centralized processing. Consequently, there are in total three different scenarios (Figure 11.2). First, centralized processing, where the raw data



(a) Centralized processing with AP topology. **(b)** Distributed processing with mesh topology.

(c) Distributed processing with AP topology.

Figure 11.2: Examples of application processing

is forwarded to a centralized server for processing. Second, a mesh-distributed scenario, where jobs are distributed among nodes for processing and the nodes can directly communicate with each other. Finally, an AP-distributed setup where jobs are again distributed among nodes but they communicate with each other via an AP (Figure 11.2).

11.1 System Model

I describe here examples for typical acoustic applications suitable for in-network processing deployment. I choose two exemplary acoustic applications for experimentation, where each application has a different application graph with different computation requirements: Cocktail Party for speaker separation (Section 11.1.1) and Double-cross-Correlation Processor (DXCP) for microphone synchronization (Section 11.1.2).

11.1.1 Cocktail Party Application

In a room with multiple speakers $s \in S$ who may talk at the same time, there are several microphones $p \in P$ collecting the combined audio signals from these

speakers, known as X. The aim is to separate these mixed signals into the original source signals, represented by $s_p \in S, \quad \forall p \in P$. In other words, we need to reconstruct the individual source signals from the observed mixed signals. To achieve this, the process of separating the signals employs the ICA [Car89] method. Here we have three assumptions: 1) the mixed signals are a combination of the original signals, 2) the original signals are independent, and 3) the original signals are not Gaussian.

The process involves estimating the $|P| \times |P|$ matrix $\boldsymbol{A}$, a mixing matrix that represents the acoustic properties of the environment, and the source signals S from the mixed signals X, where $X = \boldsymbol{A}S$. A set of four jobs (*Cov1, Whitening, norm* and *Cov2* as shown in Figure 11.3 (a)) are carried out to achieve this, starting with defining job *Cov1* as $\text{cov}(X) = xx^T = U\sigma^2U^T$. This can be rewritten as $xx^T = EDE^T$ where E is the eigenvector and D is a diagonal matrix of the eigenvalues. Then, job *whitening* whitens the data so that $X_w = (D^{\frac{-1}{2}}E^T)X$

This result is then fed into the *norm* job, where vector normalization is applied, resulting in X_n.

Next in job *cov2*, we calculate the $\text{cov}(X_nX_w)$ and calculate its eigenvector B. Finally, the source signals are estimated as $S' = BX_w$. These four jobs will be executed in a wireless distributed computing environment.

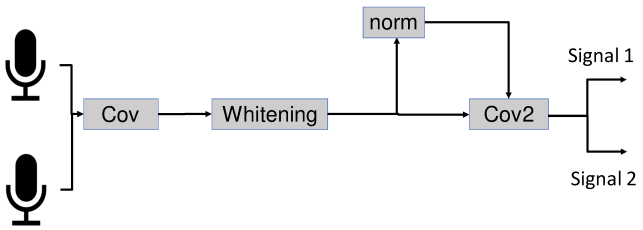
11.1.2 Synchronization

In WASN, the microphones have their own sampling clock, however, when the clocks are not synchronize, it will degrade the performance of acoustic applications. I consider DXCP[CTE19] for estimating the sampling rate offset between different microphones, which can then be used to adjust the sampling clock [Afi+18] or to compensate for their synchronization error in further acoustic processing. DXCP applies only under the assumption that sampling rate offset is time-invariant.

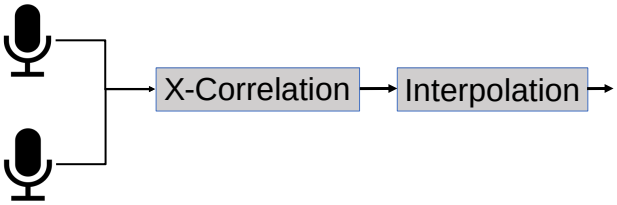
The DXCP application can be divided into two main jobs (Figure 11.3 (b)): cross-correlation and parabolic interpolation. The former is used to estimate the accumulated time delay between two (time-framed) signals. The output is then forwarded to the latter that uses a second-order polynomial interpolation to find the maximum lag and estimate the sampling rate offset.

11.2 Experiment Setup

The wireless network consists of 4 Raspberry Pi nodes (each supported with 4 CPU's) and one access point, whose MAC follows IEEE 802.11 MAC standard. I run

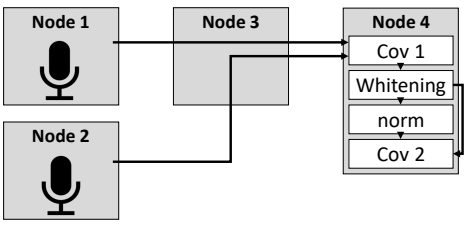


(a) ICA processing jobs.

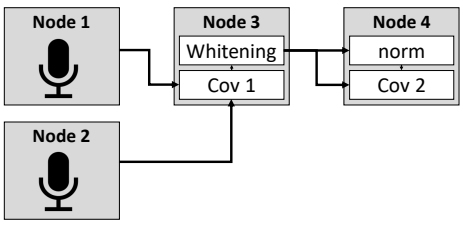


(b) DXCP processing jobs.

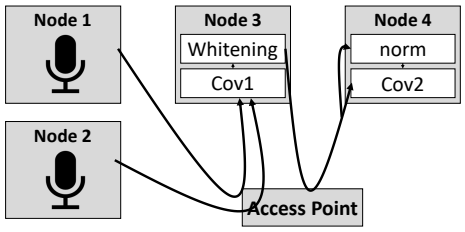
Figure 11.3: Examples of acoustic applications.



(a) Centralized.

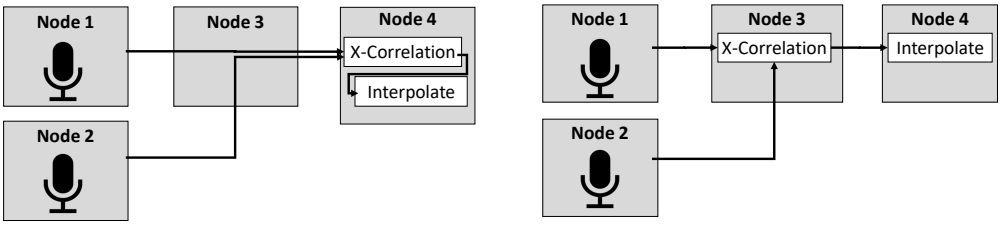


(b) Mesh-distributed.



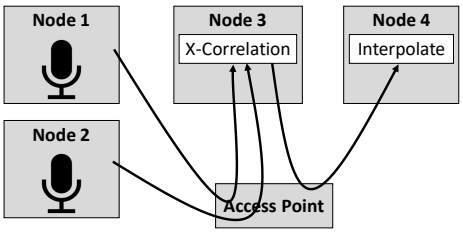
(c) AP-distributed.

Figure 11.4: Cocktail party distribution.



(a) Centralized.

(b) Mesh-distributed.



(c) AP-distributed.

Figure 11.5: Synchronization distribution.

6 experiments: two applications (cocktail-party and synchronization applications) running in three different topologies:

- **centralized**: all jobs run at a central server while the network operates in an ad hoc mode. I define a two-hop route where data need to be forwarded by an intermediate node before reaching the central server (Figure 11.5 (a) and Figure 11.4 (a)).
- **mesh-distributed**: jobs are distributed among nodes while the network operates in an adhoc mode. In this case, the intermediate node can process the data instead of just forwarding it (Figure 11.5 (b) and Figure 11.4 (b)).
- **AP-distributed**: jobs are distributed the same way as in *mesh-distributed* scenario , but the nodes communicate via AP (Figure 11.5 (c) and Figure 11.4 (c)).

I run the centralized scenario in an ad hoc mode because it allows to set a static multi-hop routing over the nodes (e.g., Node 3 here) and not via the access point. The reason of using multi-hop routing is to emulate the behavior when a node cannot communicate directly with the access point. Therefore, Node 3 is only forwarding the data from the microphones (Node 1 and Node 2) to Node 4 (acting

as a central server). Although the nodes operate in the same collision domain, the two-hop route is used to highlight the drawbacks of the centralized topology compared to mesh-distributed.

The reason for selecting this distribution among nodes is to have fair distribution of the computation power (i.e., not over-utilizing the nodes) and minimize the transmission delay. Each experiment runs using a recorded audio of 170 seconds, then data are collected for further analysis. The reason of using a recorded audio is to have sanity checks; I make sure that the algorithmic output from each scenario is the same per application. I do not change the position of the nodes between scenarios to exclude the dependency on wireless channel conditions.

11.3 Results

I measure the network delay – defined as the aggregation of both transmission and channel access delays (Figure 11.1) – between two jobs running on different nodes for each experiment. In other words, network delay for a job T is the time from starting to send a packet from job T's predecessor (or the microphone, if T has no predecessor) to the time until that data has been completely received by T. Additionally, I measure the end-to-end delay for each experiment and observe the relation between the transmission delay and end-to-end delay.

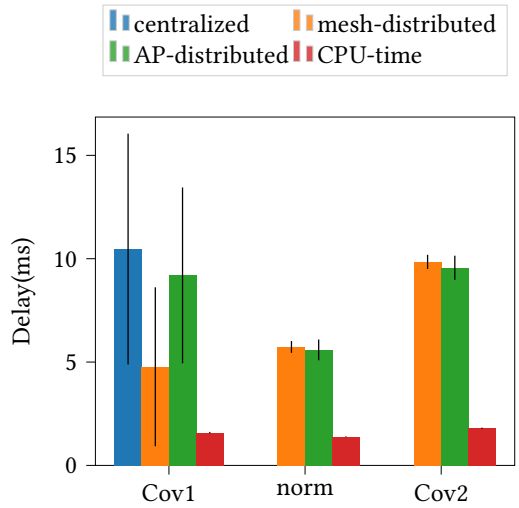
11.3.1 Wireless Network Delay

I show in Figure 11.6 two types of delay: network delay (as in *centralized*, *mesh-distributed* and *AP-distributed*) and processing delay (represented by *CPU-time*).

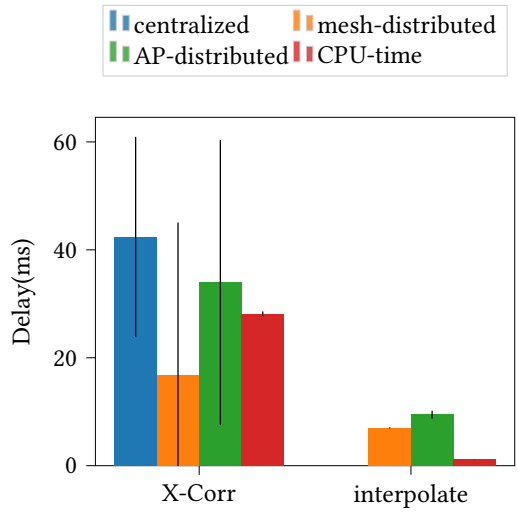
For example, I show the network delay in the cocktail party application for *Cov1* job as the average delay for receiving the data from the microphones (here the recorded data) to *Cov1*, while the network delay for task *norm* is the average delay for receiving the data from *Whitening*. In case of centralized processing, I look only at the network delay of the first job, since the rest of the jobs are running on the same node, i.e., they have no network delay (yet I will talk later about the processing delay).

In the cocktail party application (Figure 11.7 (a)), I observe that the network delay for *Cov1* job with centralized processing is the highest compared to other delays. This is expected since centralized processing requires forwarding the raw data from the microphone over two hops. Although the data also go through two hops in the AP-distributed scenario (since they need to go through the access point anyway), the network delay is slightly lower than the centralized processing, which is due to

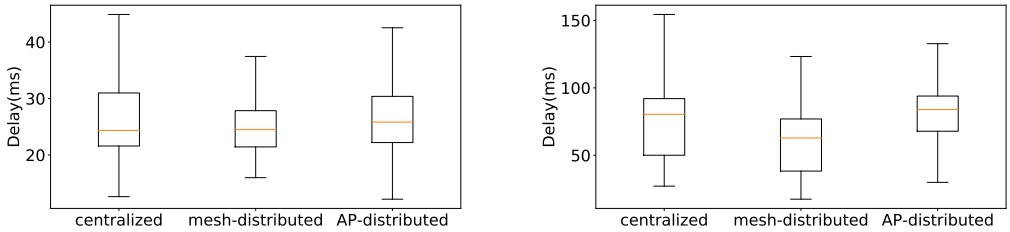
Figure 11.6: Network delay for different Setups.



(a) ICA network delay.



(b) DXCP network delay.



(a) Cocktail party.

(b) Synchronization.

Figure 11.8: Box plots for end-to-end delay.

the higher interference in mesh networks — interference management is distributed over multiple nodes and not handled by one node as in AP. This also explains why the *norm* job has similar network delay for mesh-distributed and AP-distributed scenarios (same applies to *Cov2*). I highlight that the network delay for *Cov2* is not only due to transmission channel access delay from *Whitening* job, but also the processing delay of *norm* job. This is seen in Figures 11.4 (b) and 11.4 (c), where *Cov2* has to wait for both the network delay when *Whitening* finishes processing till Node 4 received the packet and the processing delay from *norm* job running on the same node.

I observe a similar behaviour in the synchronization application (Figure 11.7 (b)); the mesh-distributed topology has the lowest network delay when forwarding recorded data to the first job, *X-correlation*) while having similar network delay to AP-distributed for the *interpolate* job. In contrast to the cocktail party application, I observe that the network delay for mesh-distributed topology is lower than the processing delay for the *X-correlation* task. This may result in unwanted delays from queuing (from previous jobs or high inter-arrival packet rate) or even packet losses, but this discussion is left for future work.

11.3.2 End-to-End Delay

The sum of the delay per each scenario in Figure 11.6 shows that distributed processing scenarios have more network delay than centralized scenarios. Meanwhile, we do not sum up all network delays to calculate the end-to-end delay, thanks to the pipeline execution; when a packet is being transmitted another packet/s is/are being processed.

I show in Figure 11.8 the end-to-end delay for both the cocktail party (Figure 11.8 (a)) and synchronization (Figure 11.8 (b)) applications. I observe that the median end-to-end delay of the cocktail party application is almost similar for all

scenarios. This is probably due to the fact that the *Cov1* job processes *N* chunks of audio signals to send a single output chunk (i.e., BLOCK_COUNT = N). Thus, the waiting time of a job can be a bottleneck for the end-to-end delay (N:1 processing). Nevertheless, the third quartile of the mesh-distributed scenario still has the lowest delay.

In case of the synchronization application, the relation between the output and the input data is 1:1. Hence, both the median and upper bound of the mesh-distributed scenario are the lowest for all scenarios.

11.4 Conclusion

I showed that wireless in-network processing with mesh topology can achieve lower network delay than the centralized and AP topologies. Meanwhile, the lowest network delay setup does not necessarily have the lowest end-to-end delay, since that also depends on the job processing delay. Yet, it is at least as low as other setups. Additionally, mesh topologies need to be carefully designed since interference avoidance is harder than the that of AP, which will increase the network delay.

12 Conclusions & Outlook

In this chapter, I conclude the thesis by summarizing the key research outcomes in relation to the research aims and questions, as well as the value and contribution herein.

One of the aims in this thesis was to highlight the missing properties in regard to managing distributed wireless network processing and whether their impact is substantial. I considered these properties from three different aspects: wireless, application-specific and application-generic.

The wireless aspects appear when nearby nodes share the bandwidth and operate in the same collision domain, making them subject to collisions. Moreover, being wireless adds other features such as power control and multi-casting. In this context, I formulated optimization problems for distributed wireless network processing while managing collisions. Compared to the related work, my solution improves network reliability by decreasing the symbol error rate. I investigated other features for distributed wireless processing, such as power control and multi-casting, showing slight improvements in throughput but not as notable as collision management.

The application-specific aspect comes from the acoustic application. Here, I did exploit the synchronization feature of WASN to synchronize wireless transmissions; it showed higher throughput compared to state of art, CSMA/CA.

Mobility is the application-generic aspect, which also fits well in wireless networks. Hence, I put it to use in autonomous vehicles, carrying microphones, for streaming audio data. I compared such a scenario with stationary microphones and carrying the microphones. Not only did it provide scalability, but it also showed good (and in some cases better) quality compared to the other scenarios.

I evaluated these aspects, while considering the trade-off between the quality of the collected information and the network performance, expressed as QoI and QoS. To find the relation between both metrics, I studied the impact of selecting a subset of microphones, which did not only show an improvement in the network performance, but also in the acoustic one. Similarly, when distributing the acoustic processing, some processing jobs can be dropped to improve, again, both acoustic and network qualities. Although, it may be thought that both QoI and QoS are inversely proportional, I showed that in some scenarios they may be actually directly proportional.

To put ideas into practice, I developed a test-bed for managing wireless in-network processing, while showing use cases for WASN. The framework adopts some of the developed resource allocation approaches and was used with a couple of case studies to show empirically that in-network processing is effective in terms of latency.

I provided different problem formulations for network management, with varying number of variables and constraints. Choosing the most suitable formulation depends on the used solver; some solvers can find a solution faster with a few number of variables or a few number of constraints. But in general, getting optimal solutions for complex problems is time-consuming, hence, I proposed a heuristic solution and derived its upper bound theoretically and showed empirically that it achieves good results compared to the optimal solutions.

Furthermore, two meta-heuristic solutions were proposed. First, a GA that improves its solution over time and second, an RL-based solution that finds a feasible solution as fast as possible and then terminates. Typical use cases for these approaches would be finding a solution within a due date using GA. If the due date is extended, GA may be able to find a better solution. The RL solution can be used in case of failover; given an invalid solution, RL finds a new solution with minimum number of migrations (i.e., replacement).

I used the outcome of the study on QoI/QoS to design an RL solution for controlling the movements of the autonomous vehicles. Comparing its performance to a heuristic solution, the former showed a better quality. Moreover, I developed a multi-agent solution that can learn faster and be more scalable, in case of controlling many vehicles. Additionally, I showed theoretically and empirically how environment properties impact the learning process.

Meanwhile, when there are multiple applications competing for the network resources, it is important to decide which application to admit. Hence, I developed an RL solution to control the admission based on revenue, priority or both. The proposed solution then showed better results compared to a first-come-first-serve protocol.

My contribution shows how to do in-network processing in wireless networks from theory to practice, yet there is still room for improvement and further investigations. For example, I studied job distribution while considering multiple wireless features simultaneously, such as power control and multi-cast transmission. But other features (e.g., mobility) were investigated separately. The reason I did that was to simplify the problem without having high dimension of variables. Accordingly, my work here is a first step that gives some insight into the gain of exploiting all wireless features simultaneously. How much gain we can achieve is yet to be studied.

One of the advantages behind disturbing the jobs is energy-efficiency, yet I have not study that in my thesis. I claim that processing the data before sending it, may decrease the amount of the data need to be sent. Another claimed advantage is the cost; exploiting the idle resources of smart devices lying around will decrease the required resources to be rent. However, the energy cost of operating these devices compared to renting the resources is yet to be investigated.

On one hand, I assumed that a subset of microphones is given when distributing the acoustic processing inside the network. On the other hand, I look separately at selecting the best subset of microphones to serve both QoI and QoS. Combining both approaches is an interesting, yet more complex problem that I have not looked into. Similarly, selecting a subset of microphones to move in autonomous vehicles may have simplified the problem of vehicle control (but not the subset selection), so that instead of controlling all microphones, a subset of microphones would be selected that best serve QoI and QoS. In this case, hierarchical learning would have been a good candidate [SPS99].

As seen in this thesis, calculating QoI is a complex process even just for acoustic processing. For other multimedia applications, QoI may follow different, even more complex calculations. An alternative approach, to hand craft the reward based on QoI, is curiosity learning [Pat+17]; it is agnostic to the external reward, which is QoI here. This is a natural follow up to my work since I use RL to solve many problems in this thesis.

Last but not least, distributed processing in WASN is immature and needs to be more motivated within the signal processing community. Here, I jointly worked with my colleagues from four different fields of signal processing (privacy, synchronization, feature extraction and scene analysis) to develop the required features of the framework. Nevertheless, additional features may be required to further simplify the deployment or the debugging of acoustic or any other application. Hence, the framework can be further developed while spreading awareness of in-networking processing.

Appendices

A CSMA/CA: Channel vs. Node Throughput

Proof. When ignoring the propagation and inter-frame spacing delays in CSMA/CA, the sending time τ_s and the collision time τ_c are the same. In other words, the time needed to detect that there is collision is equal to the time needed to transmit a packet; the acknowledgment time is neglected. Following from equation (6.2)

$$S = \frac{p_{\text{tr}} p_s L}{p_{\text{tr}}\mu + \sigma}. \tag{A.1}$$

$$= \frac{p_s L}{\mu + E[B]/N}, \tag{A.2}$$

where the average time of the channel being idle with N nodes not transmitting is due to the average back-off window given than the node wants to transmit $\sigma = p_{\text{tr}} E[B]/N$.

Now, let us consider the throughput x per node [[LK13] – eq. 4] instead of that per channel.

$$x = \frac{L}{\mu}\frac{\theta}{1 + N\theta} p_s \tag{A.3}$$

$$= \frac{L}{N\mu + E[B]} p_s, \tag{A.4}$$

where $\theta = \frac{\mu}{E[B]}$ is the ratio between the Markov states of a node is sending and the channel being idle. Alternatively, the node throughput is given by

$$x = S/N \tag{A.5}$$

$$= \frac{p_s L/N}{\mu + NE[B]/p_{\text{tr}}} \tag{A.6}$$

$$= \frac{L/N}{E[B] + N\mu} p_s \tag{A.7}$$

$$= (\text{A.2}) \tag{A.8}$$

■

Bibliography

[02] **IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)**. *IEEE Std 802.15.1-2002* (2002), 1–473. DOI: 10.1109/IEEESTD.2002.93621 (see page 13).

[19] *IEEE 802.11ac-2013*. https://standards.ieee.org/standard/802_11ac-2013.html. [Online; accessed 2019]. 2019 (see page 120).

[20] **IEEE Standard for Low-Rate Wireless Networks**. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (2020), 1–800. DOI: 10.1109/IEEESTD.2020.9144691 (see page 13).

[21] **IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications**. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), 1–4379. DOI: 10.1109/IEEESTD.2021.9363693 (see pages 13, 15, 102, 125, 126, 128, 134, 143, 153, 154).

[AAK18] Haitham Afifi, Sébastien Auroux, and Holger Karl. **MARVELO: Wireless virtual network embedding for overlay graphs with loops**. In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. 2018, 1–6. DOI: 10.1109/WCNC.2018.8377194 (see pages 8, 25, 30).

[Abd+16a] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati. **Efficient Virtual Network Embedding With Backtrack Avoidance for Dynamic Wireless Networks**. *IEEE Transactions on Wireless Communications* 15:4 (Apr. 2016), 2669–2683. ISSN: 1536-1276. DOI: 10.1109/TWC.2015.2507134 (see page 29).

[Abd+16b] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati. **Efficient Virtual Network Embedding With Backtrack Avoidance for Dynamic Wireless Networks**. *IEEE Transactions on Wireless Communications* 15:4 (Apr. 2016), 2669–2683. ISSN: 1536-1276 (see page 61).

[ABZ20] Rosa Ma Alsina-Pagès, Patrizia Bellucci, and Giovanni Zambon. **Smart Wireless Acoustic Sensor Network Design for Noise Monitoring in Smart Cities**. *Sensors* 20:17 (2020). ISSN: 1424-8220. DOI: 10.3390/s20174765. URL: https://www.mdpi.com/1424-8220/20/17/4765 (see page 12).

[Ach+17] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. **Constrained Policy Optimization**. In: ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, 22–31 (see page 20).

[Afi] Haitham Afifi. *MARVELO Documentation*. https://marvelo.readthedocs.io/en/latest/. Accessed: 2022-12-10 (see pages 9, 182).

[Afi+18] Haitam Afifi, Joerg Schmalenstroeer, Joerg Ullmann, Reinhold Haeb-Umbach, and Holger Karl. **MARVELO - A Framework for Signal Processing in Wireless Acoustic Sensor Networks**. In: *Speech Communication; 13th ITG-Symposium*. 2018, 1–5 (see pages 177, 186, 189, 192).

[Afi+21] Haitham Afifi, Michael Guenther, Andreas Brendel, Holger Karl, and Walter Kellermann. **Reinforcement Learning-based Microphone Selection in Wireless Acoustic Sensor Networks Considering Network and Acoustic Utilities**. In: *Speech Communication; 14th ITG Conference*. 2021, 1–5 (see pages 9, 115, 124, 139).

[Afi+22] Haitham Afifi, Holger Karl, Tobias Gburrek, and Joerg Schmalenstroeer. **Data-driven Time Synchronization in Wireless Multimedia Networks**. In: *IWCMC 2022 Multimedia Symposium (IWCMC 2022 Multimedia)*. Dubrovnik, Croatia, May 2022 (see pages 8, 26).

[AHK19] Haitham Afifi, Konrad Horbach, and Holger Karl. **A Genetic Algorithm Framework for Solving Wireless Virtual Network Embedding**. In: *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2019, 1–6. DOI: 10.1109/WiMOB.2019.8923271 (see pages 8, 25, 66).

[AK19a] Haitham Afifi and Holger Karl. **An Approximate Power Control Algorithm for a Multi-Cast Wireless Virtual Network Embedding**. In: *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*. 2019, 95–102. DOI: 10.23919/WMNC.2019.8881324 (see pages 8, 25, 50).

[AK19b] Haitham Afifi and Holger Karl. **Power allocation with a wireless multicast aware routing for virtual network embedding**. In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2019, 1–4 (see pages 8, 25, 43).

[AK20] Haitham Afifi and Holger Karl. **Reinforcement Learning for Virtual Network Embedding in Wireless Sensor Networks**. In: *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2020, 123–128. DOI: 10.1109/WiMob50308.2020.9253442 (see pages 25, 74).

[Akb+19] Ayhan Akbas, Huseyin Ugur Yildiz, Ahmet Murat Ozbayoglu, and Bulent Tavli. **Neural network based instant parameter prediction for wireless sensor network optimization models**. *Wireless Networks* 25:6 (Aug. 2019), 3405–3418. ISSN: 1572-8196. DOI: 10.1007/s11276-018-1808-y (see page 131).

[AKL16] Jacob Andreas, Dan Klein, and Sergey Levine. **Modular Multitask Reinforcement Learning with Policy Sketches**. *CoRR* abs/1611.01796 (2016). arXiv: 1611.01796. URL: http://arxiv.org/abs/1611.01796 (see page 165).

[AL21] Neziha Akalin and Amy Loutfi. *Reinforcement Learning Approaches in Social Robotics*. 2021. arXiv: 2009.09689 (see pages 149, 151).

[Ale22] Haitham Afifi Alexander Nelus. *Privacy-perversing Adversial Feature Extraction in Speaker Classification Tasks (Demo at WASPAA 2021)*. 2022. URL: https://ruhr-uni-bochum.sciebo.de/s/flwOSPlsp6fYAKi (see pages 9, 186).

[AM06] Sanjay P. Ahuja and Jack R. Myers. **A Survey on Wireless Grid Computing**. *The Journal of Supercomputing* 37:1 (July 2006), 3–21. ISSN: 1573-0484. DOI: 10.1007/s11227-006-3845-z. URL: https://doi.org/10.1007/s11227-006-3845-z (see page 179).

[ARK21a] Haitham Afifi, Arunselvan Ramaswamy, and Holger Karl. **A Reinforcement Learning QoI/QoS-Aware Approach in Acoustic Sensor Networks**. In: *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. 2021, 1–6. DOI: 10.1109/CCNC49032.2021.9369626 (see pages 115, 124, 131).

[ARK21b] Haitham Afifi, Arunselvan Ramaswamy, and Holger Karl. **Reinforcement Learning for Autonomous Vehicle Movements in Wireless Sensor Networks**. In: *ICC 2021 - IEEE International Conference on Communications*. 2021, 1–6. DOI: 10.1109/ICC42927.2021.9500318 (see pages 9, 115, 152).

[ARK22] Haitham Afifi, Arunselvan Ramaswamy, and Holger Karl. **Reinforcement Learning for Autonomous Vehicle Movements in Wireless Multimedia Applications**. In: *Pervasive and Mobile Computing*. submitted. 2022 (see page 9).

[ASK21] Haitham Afifi, Fabian Sauer, and Holger Karl. **Reinforcement Learning for Admission Control in Wireless Virtual Network Embedding**. *CoRR* abs/2110.01262 (2021). arXiv: 2110.01262. URL: https://arxiv.org/abs/2110.01262 (see pages 8, 25, 86).

[ASV15] Ram Bhushan Agnihotri, Ajay Vikram Singh, and Shekhar Verma. **Challenges in wireless sensor networks with different performance metrics in routing protocols**. In: *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*. 2015, 1–5. DOI: 10.1109/ICRITO.2015.7359295 (see page 123).

[AYY02] Khaled Arisha, Moustafa Youssef, and Mohamed Younis, 21–40. In: *System-Level Power Optimization for Wireless Multimedia Communication: Power Aware Computing*. Ed. by Ramesh Karri and David Goodman. Boston, MA: Springer US, 2002. ISBN: 978-0-306-47720-1. DOI: 10.1007/0-306-47720-3_2 (see page 189).

[Bac10] Francis R. Bach. **Convex Analysis and Optimization with Submodular Functions: a Tutorial**. *CoRR* abs/1010.4207 (2010). arXiv: 1010.4207. URL: http://arxiv.org/abs/1010.4207 (see page 124).

[BAD14] J. van de Belt, H. Ahmadi, and L. Doyle. **A Dynamic Embedding Algorithm for Wireless Network Virtualization** (June 2014) (see page 29).

[BAD17] J. van de Belt, H. Ahmadi, and L. E. Doyle. **Defining and Surveying Wireless Link Virtualization and Wireless Network Virtualization**. *IEEE Communications Surveys Tutorials* 19:3 (2017), 1603–1627. DOI: 10.1109/COMST.2017.2704899 (see page 29).

[Bai95] Leemon Baird. "Residual Algorithms: Reinforcement Learning with Function Approximation." In: *Machine Learning Proceedings 1995*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann, 1995, 30–37. ISBN: 978-1-55860-377-6. DOI: https://doi.org/10.1016/B978-1-55860-377-6.50013-X. URL: https://www.sciencedirect.com/science/article/pii/B978155860377650013X (see page 18).

[BAK04a] H. Buchner, R. Aichner, and W. Kellermann. **TRINICON: A versatile framework for multichannel blind signal processing**. In: *IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*. Montreal, Canada, May 2004 (see page 120).

[BAK04b] H. Buchner, R. Aichner, and W. Kellermann. **TRINICON: a versatile framework for multichannel blind signal processing**. In: *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*. Vol. 3. 2004, iii–889. DOI: 10.1109/ICASSP.2004.1326688 (see pages 12, 101).

[Ban+09] Yonghwan Bang, Jongpil Han, Kyusang Lee, Jongwon Yoon, Jinoo Joung, Sungbo Yang, and June-Koo Kevin Rhee. **Wireless network synchronization for multichannel multimedia services**. In: *Proc. 11th Intl. Conf. Advanced Communication Technology*. Vol. 02. 2009, 1073–1077 (see page 99).

[Ban+11] Nikhil Bansal, Kang-Won Lee, Viswanath Nagarajan, and Murtaza Zafer. **Minimum Congestion Mapping in a Cloud**. In: *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC '11. San Jose, California, USA: ACM, 2011, 267–276. ISBN: 978-1-4503-0719-2 (see page 53).

[BBK18] M. Bachmann, A. Brendel, and W. Kellermann. **Resource Allocation for Distributed Blind Source Separation**. In: *Speech Commun.; 13th ITG-Symp.* Oldenburg, Germany, Oct. 2018 (see page 117).

[Ber99] D.P. Bertsekas. **Nonlinear Programming**. Athena Scientific, 1999 (see page 19).

[BFO96] G. Bianchi, L. Fratta, and M. Oliveri. **Performance evaluation and enhancement of the CSMA/CA MAC protocol for 802.11 wireless LANs**. In: *Proceedings of PIMRC '96 - 7th International Symposium on Personal, Indoor, and Mobile Communications*. Vol. 2. 1996, 392–396 vol.2. DOI: 10.1109/PIMRC.1996.567423 (see pages 102, 104).

[Bia00] G. Bianchi. **Performance analysis of the IEEE 802.11 distributed coordination function**. *IEEE Journal on Selected Areas in Communications* 18:3 (2000), 535–547. DOI: 10.1109/49.840210 (see pages 102–104, 107).

[BK18] A. Brendel and W. Kellermann. **Distance Estimation of Acoustic Sources Using the Coherent-to-Diffuse Power Ratio Based on Distributed Training**. In: *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*. 2018, 1–5 (see pages 149, 155).

[Ble+16] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer. **Boost online virtual network embedding: Using neural networks for admission control**. In: *2016 12th International Conference on Network and Service Management (CNSM)*. 2016, 10–18. DOI: 10.1109/CNSM.2016.7818395 (see page 86).

[Ble+18] A. Blenk, P. Kalmbach, J. Zerwas, M. Jarschel, S. Schmid, and W. Kellerer. **NeuroViNE: A Neural Preprocessor for Your Virtual Network Embedding Algorithm**. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, 405–413. DOI: 10.1109/INFOCOM.2018.8486263 (see page 86).

[BM94] Justin Boyan and Andrew Moore. **Generalization in Reinforcement Learning: Safely Approximating the Value Function**. In: *Advances in Neural Information Processing Systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1994. URL: https://proceedings.neurips.cc/paper/1994/file/ef50c335cca9f340bde656363ebd02fd-Paper.pdf (see page 18).

[BMH19] Liu Boyang, Wu Muqing, and Zou Haosen. **Virtual Network Embedding Based on Hybrid Adaptive Genetic Algorithm**. In: *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. 2019, 1197–1202. DOI: 10.1109/ICCC47050.2019.9064173 (see page 65).

[BT17] Martin Bunder and Joseph Tonien. **Closed form expressions for two harmonic continued fractions**. *The Mathematical Gazette* 101:552 (2017), 439–448. DOI: 10.1017/mag.2017.125 (see page 25).

[BTS98] Timothy X. Brown, Hui Tong, and Satinder Singh. **Optimizing Admission Control While Ensuring Quality of Service in Multimedia Networks via Reinforcement Learning**. In: *Proceedings of the 11th International Conference on Neural Information Processing Systems*. NIPS'98. Denver, CO: MIT Press, 1998, 982–988 (see page 86).

[CA22] Aleksej Chinaev and Haitham Afifi. *Acquisition of Asynchronous Data and Parameter Estimation based on Double-Cross- Correlation Processor with Phase Transform (Demo at WASPAA 2021)*. 2022. URL: https://sigport.org/documents/acquisition-asynchronous-data-and-parameter-estimation-based-double-cross-correlation (see page 9).

[Car89] J. -. Cardoso. **Source separation using higher order moments**. In: *International Conference on Acoustics, Speech, and Signal Processing*, 1989, 2109–2112 vol.4. DOI: 10.1109/ICASSP.1989.266878 (see page 192).

[CEP07] Todor Cooklev, John C. Eidson, and Afshaneh Pakdaman. **An Implementation of IEEE 1588 Over IEEE 802.11b for Synchronization of Wireless Local Area Network Nodes**. *IEEE Transactions on Instrumentation and Measurement* 56:5 (2007), 1632–1639. DOI: 10.1109/TIM.2007.903640 (see page 100).

[CG17a] Dani Cherkassky and Sharon Gannot. **Blind Synchronization in Wireless Acoustic Sensor Networks**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25:3 (2017), 651–661. DOI: 10.1109/TASLP.2017.2655259 (see page 101).

[CG17b] Dani Cherkassky and Sharon Gannot. **Blind Synchronization in Wireless Acoustic Sensor Networks**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25:3 (2017), 651–661. DOI: 10.1109/TASLP.2017.2655259 (see page 124).

[Cha+19] U. Challita, A. Ferdowsi, M. Chen, and W. Saad. **Machine Learning for Wireless Connectivity and Security of Cellular-Connected UAVs**. *IEEE Wireless Communications* 26:1 (2019), 28–35. DOI: 10.1109/MWC.2018.1800155 (see page 152).

[Che+09] X. Chen, T. R. Newman, D. Datla, T. Bose, and J. H. Reed. **The Impact of Channel Variations on Wireless Distributed Computing Networks**. In: *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*. Nov. 2009, 1–6. DOI: 10.1109/GLOCOM.2009.5425441 (see page 189).

[CKS21] Jonah Casebeer, Jamshed Kaikaus, and Paris Smaragdis. **Communication-Cost Aware Microphone Selection for Neural Speech Enhancement with Ad-Hoc Microphone Arrays**. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, 8438–8442. DOI: 10.1109/ICASSP39728.2021.9414775 (see page 124).

[CL13] Sundeep Prabhakar Chepuri and Geert Leus. **Sparsity-Promoting Sensor Selection for Non-linear Measurement Models**. *CoRR* abs/1310.5251 (2013). arXiv: 1310.5251. URL: http://arxiv.org/abs/1310.5251 (see page 124).

[CLZ16] H. Chen, X. Li, and F. Zhao. **A Reinforcement Learning-Based Sleep Scheduling Algorithm for Desired Area Coverage in Solar-Powered Wireless Sensor Networks**. *IEEE Sensors Journal* 16:8 (2016), 2763–2774. DOI: 10.1109/JSEN.2016.2517084 (see page 85).

[Cob+17] Maximo Cobos, Fabio Antonacci, Anastasios Alexandridis, Athanasios Mouchtaris, and Bowon Lee. **A Survey of Sound Source Localization Methods in Wireless Acoustic Sensor Networks**. *Wireless Communications and Mobile Computing* 2017 (Aug. 2017), 3956282. ISSN: 1530-8669. DOI: 10.1155/2017/3956282. URL: https://doi.org/10.1155/2017/3956282 (see page 117).

[Cos+12] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo. **Software Defined Wireless Networks: Unbridling SDNs**. In: *2012 European Workshop on Software Defined Networking*. Oct. 2012, 1–6. DOI: 10.1109/EW{SDN}.2012.12 (see page 180).

[CS15] Shanti Chilukuri and Anirudha Sahoo. **Delay-Aware TDMA Scheduling for Multi-Hop Wireless Networks**. In: *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. ICDCN '15. Goa, India: Association for Computing Machinery, 2015. ISBN: 9781450329286. DOI: 10.1145/2684464.2684493. URL: https://doi.org/10.1145/2684464.2684493 (see page 190).

[CSB19] Ursula Challita, Walid Saad, and Christian Bettstetter. **Interference Management for Cellular-Connected UAVs: A Deep Reinforcement Learning Approach**. *IEEE Transactions on Wireless Communications* 18:4 (Apr. 2019), 2125–2140. ISSN: 1558-2248. DOI: 10.1109/twc.2019.2900035. URL: http://dx.doi.org/10.1109/TWC.2019.2900035 (see page 152).

[CTE19] A. Chinaev, P. Thüne, and G. Enzner. **A Double-Cross-Correlation Processor for Blind Sampling Rate Offset Estimation in Acoustic Sensor Networks**. In: *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* May 2019, 641–645. DOI: 10.1109/ICASSP.2019.8683605 (see page 192).

[Dat+12] Dinesh Datla, Xuetao Chen, Thomas Tsou, Sahana Raghunandan, S.M. Shajedul Hasan, Jeffrey H. Reed, Carl B. Dietrich, Tamal Bose, Bruce Fette, and Jeong-Ho Kim. **Wireless distributed computing: a survey of research challenges**. *IEEE Communications Magazine* 50:1 (2012), 144–152. DOI: 10.1109/MCOM.2012.6122545 (see page 179).

[Dek+22] Gert Dekkers, Fernando Rosas, Toon van Waterschoot, Bart Vanrumste, and Peter Karsmakers. **Dynamic sensor activation and decision-level fusion in wireless acoustic sensor networks for classification of domestic activities**. *Information Fusion* 77 (2022), 196–210. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.07.022 (see page 118).

[DG15] Yuval Dorfan and Sharon Gannot. **Tree-Based Recursive Expectation-Maximization Algorithm for Localization of Acoustic Sources**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23:10 (2015), 1692–1703. DOI: 10.1109/TASLP.2015.2444654 (see page 101).

[DH04] Hui Dai and Richard Han. **TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks**. *SIGMOBILE Mob. Comput. Commun. Rev.* 8:1 (Jan. 2004), 125–139. ISSN: 1559-1662. DOI: 10.1145/980159.980173 (see page 100).

[Din18] Yuanming Shi; Zhi Ding. **Low-Rank Optimization for Data Shuffling in Wireless Distributed Computing** (2018) (see page 74).

[Din20] Hamed Dinari. **Inter-Process Communication (IPC) in Distributed Environments: An Investigation and Performance Analysis of Some Middleware Technologies**. *International Journal of Modern Education and Computer Science* 12 (Apr. 2020), 36–52. DOI: 10.5815/ijmecs.2020.02.05 (see page 182).

[Drä+18] Sevil Dräxler, Manuel Peuster, Marvin Illian, and Holger Karl. **Generating Resource and Performance Models for Service Function Chains: The Video Streaming Case**. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, 318–322. DOI: 10.1109/NETSOFT.2018.8460029 (see page 85).

[DSB01] Joseph H. DiBiase, Harvey F. Silverman, and Michael S. Brandstein, 157–180. In: *Microphone Arrays: Signal Processing Techniques and Applications*. Ed. by Michael Brandstein and Darren Ward. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. ISBN: 978-3-662-04619-7. DOI: 10.1007/978-3-662-04619-7_8 (see page 99).

[DSK18] Sevil Dräxler, Stefan Schneider, and Holger Karl. **Scaling and Placing Bidirectional Services with Stateful Virtual and Physical Network Functions**. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, 123–131. DOI: 10.1109/NETSOFT.2018.8459915 (see page 117).

[Ebb+18] Janek Ebbers, Jens Heitkaemper, Joerg Schmalenstroeer, and Reinhold Haeb-Umbach. **Benchmarking Neural Network Architectures for Acoustic Sensor Networks**. In: *Speech Communication; 13th ITG-Symposium*. 2018, 1–5 (see pages 185, 186).

[EGE03] Jeremy Elson, Lewis Girod, and Deborah Estrin. **Fine-Grained Network Time Synchronization Using Reference Broadcasts**. *SIGOPS Oper. Syst. Rev.* 36:SI (Dec. 2003), 147–163. ISSN: 0163-5980. DOI: 10.1145/844128.844143 (see page 100).

[ERS16] Guy Even, Matthias Rost, and Stefan Schmid. **An Approximation Algorithm for Path Computation and Function Placement in s**. In: *Structural Information and Communication Complexity*. Ed. by Jukka Suomela. Cham: Springer International Publishing, 2016, 374–390. ISBN: 978-3-319-48314-6 (see page 53).

[Faj+11] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. **VNE-AC: Virtual Network Embedding Algorithm Based on Ant Colony Metaheuristic**. In: *2011 IEEE International Conference on Communications (ICC)*. 2011, 1–6. DOI: 10.1109/icc.2011.5963442 (see page 65).

[FGD16] E. F. Flushing, L. M. Gambardella, and G. A. Di Caro. **On Using Mobile Robotic Relays for Adaptive Communication in Search and Rescue Missions**. In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2016, 370–377 (see page 151).

[FGV05] Cédric Févotte, Rémi Gribonval, and Emmanuel Vincent. **BSS_EVAL Toolbox User Guide – Revision 2.0**. report. 2005. URL: https://hal.inria.fr/inria-00564760 (visited on 09/24/2019) (see pages 127, 142).

[Fis+13] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. **Virtual Network Embedding: A Survey**. *IEEE Communications Surveys Tutorials* 15:4 (2013), 1888–1906 (see page 27).

[FK09] András Frank and Tamás Király, 87–126. In: *Research Trends in Combinatorial Optimization: Bonn 2008*. Ed. by William Cook, László Lovász, and Jens Vygen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-76796-1. DOI: 10.1007/978-3-540-76796-1_6 (see page 124).

[Fra+20] Lidia Pocero Fraile, Stelios Tsampas, Georgios Mylonas, and Dimitrios Amaxilatis. **A Comparative Study of LoRa and IEEE 802.15.4-Based IoT Deployments Inside School Buildings**. *IEEE Access* 8 (2020), 160957–160981. DOI: 10.1109/ACCESS.2020.3020685 (see page 13).

[Fre] Freifunk. *Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.)* https://github.com/open-mesh-mirror/batman-adv. Accessed: 2022-11-24 (see pages 181, 184).

[FS94] Michael P. Fitz and James P. Seymour. **On the bit error probability of QAM modulation**. *International Journal of Wireless Information Networks* 1:2 (Apr. 1994), 131–139. ISSN: 1572-8129 (see page 61).

[Fu+20] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao. **Dynamic Service Function Chain Embedding for NFV-Enabled IoT: A Deep Reinforcement Learning Approach**. *IEEE Transactions on Wireless Communications* 19:1 (Jan. 2020), 507–519. ISSN: 1558-2248. DOI: 10.1109/TWC.2019.2946797 (see page 65).

[FYD09] Osama Farrag, Mohamed Younis, and William D'Amico. **MAC Support for Wireless Multimedia Sensor Networks**. In: *Proc. IEEE Global Telecommunications Conference (GlobeCom)*. 2009, 1–6. DOI: 10.1109/GLOCOM.2009.5425242 (see page 100).

[Gal+15] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo. **SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks**. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015, 513–521. DOI: 10.1109/INFOCOM.2015.7218418 (see page 180).

[Gao+15] X. Gao, W. Zhong, Z. Ye, Y. Zhao, J. Fan, X. Cao, H. Yu, and C. Qiao. **Virtual Network Mapping for Reliable Multicast Services with Max-Min Fairness**. In: *2015 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2015, 1–6. DOI: 10.1109/GLOCOM.2015.7417549 (see page 29).

[GB16] Juliver Gil Herrera and Juan Felipe Botero. **Resource Allocation in NFV: A Comprehensive Survey**. *IEEE Transactions on Network and Service Management* 13:3 (2016), 518–532. DOI: 10.1109/TNSM.2016.2598420 (see page 7).

[GBK21] Michael Günther, Andreas Brendel, and Walter Kellermann. **Online estimation of time-variant microphone utility in wireless acoustic sensor networks using single-channel signal features**. In: *29th European Signal Processing Conference (EUSIPCO 2021)*. Dublin, Ireland, 2021 (see page 153).

[GBW01] S. Gannot, D. Burshtein, and E. Weinstein. **Signal enhancement using beamforming and nonstationarity with applications to speech**. *IEEE Transactions on Signal Processing* 49:8 (2001), 1614–1626. DOI: 10.1109/78.934132 (see page 101).

[Gei06] Peter Geibel. **Reinforcement Learning for MDPs with Constraints**. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, 646–653. ISBN: 978-3-540-46056-5 (see page 20).

[GFF15a] Javier García, Fern, and o Fernández. **A Comprehensive Survey on Safe Reinforcement Learning**. *Journal of Machine Learning Research* 16:42 (2015), 1437–1480. URL: http://jmlr.org/papers/v16/garcia15a.html (see page 20).

[GFF15b] Javier García, Fern, and o Fernández. **A Comprehensive Survey on Safe Reinforcement Learning**. *Journal of Machine Learning Research* 16:42 (2015), 1437–1480 (see page 20).

[GKS03] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. **Timing-Sync Protocol for Sensor Networks**. In: *Proc. 1st Intl. Conf. Embedded Networked Sensor Systems (SenSys)*. SenSys. Los Angeles, California, USA: Association for Computing Machinery, 2003, 138–149. ISBN: 1581137079. DOI: 10.1145/958491.958508 (see page 100).

[Gon20] Xiaowen Gong. *Delay-Optimal Distributed Edge Computing in Wireless Edge Networks*. 2020. arXiv: 2002.02596 [cs.NI] (see page 189).

[GR03] Jana van Greunen and Jan Rabaey. **Lightweight Time Synchronization for Sensor Networks**. In: *Proc. 2nd ACM Intl. Conf. Wireless Sensor Networks and Applications*. WSNA. San Diego, CA, USA: Association for Computing Machinery, 2003, 11–19. ISBN: 1581137648. DOI: 10.1145/941350.941353 (see page 100).

[Gre] Greg Bowman. *Folding at home*. https://foldingathome.org/. Accessed: 2022-12-10 (see page 179).

[Gre84] Rick Greer. **Trees and Hills: Methodology for Maximizing Functions of Systems of Linear Relations**. NLD: North-Holland Publishing Co., 1984. ISBN: 0444875786 (see page 157).

[GSH21] Tobias Gburrek, Joerg Schmalenstroeer, and Reinhold Haeb-Umbach. **On Synchronization of Wireless Acoustic Sensor Networks in the Presence of Time-varying Sampling Rate Offsets and Speaker Changes**. *arXiv preprint arXiv:2110.12820* (2021) (see pages 12, 99, 101, 102, 105–107, 186–188).

[Gün+19] Michael Günther, Haitham Afifi, Andreas Brendel, Holger Karl, and Walter Kellermann. **Sparse Adaptation of Distributed Blind Source Separation in Acoustic Sensor Networks**. In: *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2019, 190–194. DOI: 10.1109/WASPAA.2019.8937194 (see pages 115, 117, 121).

[Gun+21] Michael Gunther, Haitham Afifi, Andreas Brendel, Holger Karl, and Walter Kellermann. **Network-Aware Optimal Microphone Channel Selection in Wireless Acoustic Sensor Networks**. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, 820–824. DOI: 10.1109/ICASSP39728.2021.9414528 (see pages 9, 101, 115, 119, 122, 124, 127–129, 131, 141).

[Guo+19] X. Guo, H. Lin, Z. Li, and M. Peng. **Deep Reinforcement Learning based QoS-aware Secure Routing for SDN-IoT**. *IEEE Internet of Things Journal* (2019), 1–1. ISSN: 2372-2541. DOI: 10.1109/JIOT.2019.2960033 (see page 65).

[Gup16] Naresh Kumar Gupta. 2016 (see page 13).

[Gur18] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2018. URL: http://www.gurobi.com (see pages 57, 135).

[Haia] Haitham Afifi. *Trained multi agents*. https://git.cs.uni-paderborn.de/hafifi/particle_mic. Accessed: 2022-12-10 (see page 158).

[Haib] Haitham Afifi. *Trained RL agents*. https://git.cs.uni-paderborn.de/hafifi/rl_uav. Accessed: 2022-12-10 (see page 158).

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. **Deep Reinforcement Learning with Double Q-learning**. *CoRR* abs/1509.06461 (2015). eprint: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (see page 151).

[HHA19] Hado van Hasselt, Matteo Hessel, and John Aslanides. *When to use parametric models in reinforcement learning?* 2019. arXiv: 1906.05243 [cs.LG] (see page 131).

[Hil+18] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018 (see page 88).

[HS19] J. Fernando Hernandez-Garcia and Richard S. Sutton. **Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target**. *CoRR* abs/1901.07510 (2019). arXiv: 1901.07510. URL: http://arxiv.org/abs/1901.07510 (see page 141).

[Hua+22] Sandy Huang, Abbas Abdolmaleki, Giulia Vezzani, Philemon Brakel, Daniel J. Mankowitz, Michael Neunert, Steven Bohez, Yuval Tassa, Nicolas Heess, Martin Riedmiller, and Raia Hadsell. **A Constrained Multi-Objective Reinforcement Learning Framework**. In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, Aug. 2022, 883–893. URL: https://proceedings.mlr.press/v164/huang22a.html (see page 20).

[HWW11] William E Hart, Jean-Paul Watson, and David L Woodruff. **Pyomo: modeling and solving mathematical programs in Python**. *Mathematical Programming Computation* 3:3 (2011), 219–260 (see pages 48, 57, 135).

[JA22] Joerg Schmalenstoeer Janek Ebbers Tobias Gburrek and Haitham Afifi. *Acoustic scene classification on a Raspberry-Pi network (Demo at WASPAA 2021)*. 2022 (see pages 9, 185, 186).

[JB09] Siddharth Joshi and Stephen Boyd. **Sensor Selection via Convex Optimization**. *IEEE Transactions on Signal Processing* 57:2 (2009), 451–462. DOI: 10.1109/TSP.2008.2007095 (see page 124).

[JG85] David S Johnson and Michael R Garey. **A 7160 theorem for bin packing**. *Journal of Complexity* 1:1 (1985), 65–106. ISSN: 0885-064X (see pages 56, 60).

[JMA15] I. Jawhar, N. Mohamed, and J. Al-Jaroodi. **UAV-based data communication in wireless sensor networks: Models and strategies**. In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2015, 687–694. DOI: 10.1109/ICUAS.2015.7152351 (see pages 149, 151).

[Jou+17] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn,

Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. **In-Datacenter Performance Analysis of a Tensor Processing Unit**. *CoRR* abs/1704.04760 (2017). arXiv: 1704.04760. URL: http://arxiv.org/abs/1704.04760 (see page 5).

[JSH12] Florian Jacob, Joerg Schmalenstroeer, and Reinhold Haeb-Umbach. **Microphone Array Position Self-Calibration from Reverberant Speech Input**. In: *IWAENC 2012; International Workshop on Acoustic Signal Enhancement*. 2012, 1–4 (see pages 12, 117).

[Kad16] Yasin Murat Kadioglu, 117–125. In: *Computer and Information Sciences: 31st International Symposium, ISCIS 2016, Kraków, Poland, October 27–28, 2016, Proceedings*. Ed. by Tadeusz Czachórski, Erol Gelenbe, Krzysztof Grochla, and Ricardo Lent. Springer International Publishing, 2016. ISBN: 978-3-319-47217-1 (see page 6).

[Kan+20] Takeshi Kanematsu, Kosuke Sanada, Zhetao Li, Tingrui Pei, Young-June Choi, Kien Nguyen, and Hiroo Sekiya. **Throughput and delay analysis for IEEE 802.11 multi-hop networks considering data rate**. *International Journal of Distributed Sensor Networks* 16:9 (2020), 1550147720959262. DOI: 10.1177/1550147720959262. eprint: https://doi.org/10.1177/1550147720959262. URL: https://doi.org/10.1177/1550147720959262 (see page 190).

[Kar20] Thommen George Karimpanal. **Neuro-evolutionary Frameworks for Generalized Learning Agents**. *CoRR* abs/2002.01088 (2020). arXiv: 2002.01088. URL: https://arxiv.org/abs/2002.01088 (see page 65).

[Kat+20] R. Katona, V. Cionca, D. O'Shea, and D. Pesch. **Virtual Network Embedding for Wireless Sensor Networks Time Efficient QoS/QoI Aware Approach**. *IEEE Internet of Things Journal* (2020), 1–1 (see page 152).

[Kim+07] T. Kim, H. T. Attias, S.-Y. Lee, and T.-W. Lee. **Blind Source Separation Exploiting Higher-Order Frequency Dependencies**. *IEEE Trans. Audio, Speech, Language Process.* 15:1 (Jan. 2007), 70–79 (see page 117).

[Kir+20] Bangalore Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick Pérez. **Deep Reinforcement Learning for Autonomous Driving: A Survey**. *CoRR* abs/2002.00444 (2020). arXiv: 2002.00444. URL: https://arxiv.org/abs/2002.00444 (see page 151).

[KSM03] Byung-Jae Kwak, Nah-Oak Song, and L.E. Miller. **Analysis of the stability and performance of exponential backoff**. In: *Proc. IEEE Wireless Communications and Networking*, vol. 3. 2003, 1754–1759 vol.3. DOI: 10.1109/WCNC.2003.1200652 (see pages 102, 104).

[KW05] Holger Karl and Andreas Willig, 59–81. In: *Protocols and Architectures for Wireless Sensor Networks*. 2005. DOI: 10.1002/0470095121.ch3 (see pages 7, 11, 13).

[KW06] I. Y. Kim and O. L. de Weck. **Adaptive weighted sum method for multiobjective optimization: a new method for Pareto front generation**. *Structural and Multidisciplinary Optimization* 31:2 (Feb. 2006), 105–116. ISSN: 1615-1488. DOI: 10.1007/s00158-005-0557-6 (see page 21).

[Lan+17] Marc Lanctot, Vinícius Flores Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. **A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning**. *CoRR* abs/1711.00832 (2017). arXiv: 1711.00832. URL: http://arxiv.org/abs/1711.00832 (see page 156).

[LaV06] S. M. LaValle. **Planning Algorithms**. Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006 (see page 151).

[LFZ13] Olaf Landsiedel, Federico Ferrari, and Marco Zimmerling. **Chaos: Versatile and Efficient All-to-All Data Sharing and in-Network Processing at Scale**. In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys '13. Roma, Italy: Association for Computing Machinery, 2013. ISBN: 9781450320276. DOI: 10.1145/2517351.2517358 (see pages 7, 27).

[Li+16] M. Li, X. Wang, S. Chen, M. Song, and Y. Ma, 238–249. In: *Human Centered Computing: Second International Conference, HCC 2016, Colombo, Sri Lanka, January 7-9, 2016, Revised Selected Papers*. Springer International Publishing, 2016. ISBN: 978-3-319-31854-7. DOI: 10.1007/978-3-319-31854-7_22 (see page 29).

[Li+17] M. Li, C. Hua, C. Chen, and X. Guan. **Application-driven virtual network embedding for industrial wireless sensor networks**. In: *IEEE Intl. Conf. on Communications (ICC)*. May 2017, 1–6. DOI: 10.1109/ICC.2017.7996431 (see page 29).

[LI04] Kun Li and P. Ioannou. **Modeling of traffic flow of automated vehicles**. *IEEE Transactions on Intelligent Transportation Systems* 5:2 (2004), 99–113. DOI: 10.1109/TITS.2004.828170 (see pages 150, 151).

[Liu+10] C. H. Liu, C. Bisdikian, J. W. Branch, and K. K. Leung. **QoI-Aware Wireless Sensor Network Management for Dynamic Multi-Task Operations**. In: *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. 2010, 1–9 (see page 131).

[LK13] Rafael Laufer and Leonard Kleinrock. **On the capacity of wireless CSMA/CA multihop networks**. In: *Proc. IEEE INFOCOM*. 2013, 1312–1320. DOI: 10.1109/INFCOM.2013.6566924 (see pages 104, 107, 112, 205).

[LK99] S.M. LaValle and J.J. Kuffner. **Randomized kinodynamic planning**. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 1. 1999, 473–479 vol.1. DOI: 10.1109/ROBOT.1999.770022 (see page 151).

[Lor+08] Konrad Lorincz, Bor-rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. **Resource Aware Programming in the Pixie OS**. In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. Raleigh, NC, USA: ACM, 2008, 211–224. ISBN: 978-1-59593-990-6. DOI: 10.1145/1460412.1460434. URL: http://doi.acm.org/10.1145/1460412.1460434 (see page 180).

[LTQ12] T. Luo, H. P. Tan, and T. Q. S. Quek. **Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks**. *IEEE Communications Letters* 16:11 (Nov. 2012), 1896–1899. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2012.092812.121712 (see page 179).

[Lv+12] P. Lv, Z. Cai, J. Xu, and M. Xu. **Multicast Service-Oriented Virtual Network Embedding in Wireless Mesh Networks**. *IEEE Communications Letters* 16:3 (Mar. 2012), 375–377. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2012.012412.112364 (see page 29).

[Mah+17] Aneeq Mahmood, Reinhard Exel, Henning Trsek, and Thilo Sauter. **Clock Synchronization Over IEEE 802.11—A Survey of Methodologies and Protocols**. *IEEE Transactions on Industrial Informatics* 13:2 (2017), 907–922. DOI: 10.1109/TII.2016.2629669 (see page 100).

[Mar18] Andreas Brendel Markus Bachman Haitham Afifi. *Acoustic signal extraction and enhancement over acoustic sensor networks (Demo at ITG conference on Speech Communication)*. 2018 (see page 9).

[McC76] Garth P. McCormick. **Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems**. *Mathematical Programming* 10:1 (Dec. 1976), 147–175. ISSN: 1436-4646. DOI: 10.1007/BF01580665 (see pages 46, 51).

[MCN17] Rajarshi Middya, Nabajit Chakravarty, and Mrinal Kanti Naskar. **Compressive Sensing in Wireless Sensor Networks – a Survey**. *IETE Technical Review* 34:6 (2017), 642–654. DOI: 10.1080/02564602.2016.1233835. eprint: https://doi.org/10.1080/02564602.2016.1233835. URL: https://doi.org/10.1080/02564602.2016.1233835 (see page 7).

[MGS09] Daniel Mueller-Gritschneder, Helmut Graeb, and Ulf Schlichtmann. **A Successive Approach to Compute the Bounded Pareto Front of Practical Multiobjective Optimization Problems**. *SIAM Journal on Optimization* 20:2 (2009), 915–934. DOI: 10.1137/080729013 (see page 127).

[MI98] Noboru Murata and Shiro Ikeda. **An on-line algorithm for blind source separation on speech signals**. In: *Proc. NOLTA98*. Vol. 3. Crans-Montana, Switzerland, Sept. 1998, 923–926 (see page 117).

[Mil+18] Stefan Milz, Georg Arbeiter, Christian Witt, Bassam Abdallah, and Senthil Yogamani. **Visual SLAM for Automated Driving: Exploring the Applications of Deep Learning**. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018, 360–36010. DOI: 10.1109/CVPRW.2018.00062 (see page 151).

[Mil91] D.L. Mills. **Internet time synchronization: the network time protocol**. *IEEE Transactions on Communications* 39:10 (1991), 1482–1493. DOI: 10.1109/26.103043 (see page 100).

[Mit96] Melanie Mitchell. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: MIT Press, 1996. ISBN: 0-262-13316-4 (see page 66).

[MKA21] Pedro Martinez-Julia, Ved P. Kafle, and Hitoshi Asaeda. **A Genetic Approach to Continuous Optimization of Virtual Network Embedding**. In: *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2021, 70–74. DOI: 10.1109/ICIN51074.2021.9385542 (see page 65).

[Mor20] M. Morales. **Grokking Deep Reinforcement Learning**. Manning, 2020. ISBN: 9781638356660 (see page 18).

[MR14] Bruno F. Marques and Manuel P. Ricardo. **Improving the energy efficiency of WSN by using application-layer topologies to constrain RPL-defined routing trees**. In: *Proc. 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. 2014, 126–133. DOI: 10.1109/MedHocNet.2014.6849114 (see page 100).

[Mus+17] Ibrahim Mustapha, Borhanuddin M. Ali, A. Sali, M.F.A. Rasid, and H. Mohamad. **An energy efficient Reinforcement Learning based Cooperative Channel Sensing for Cognitive Radio Sensor Networks**. *Pervasive and Mobile Computing* 35 (2017), 165–184. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2016.07.007. URL: https://www.sciencedirect.com/science/article/pii/S1574119216301079 (see page 85).

[Nai04] Ranjit Nair. **Coordinating multiagent teams in uncertain domains using distributed POMDPs** . PhD thesis. 2004 (see page 156).

[Ng+20] Jer Shyuan Ng, Wei Yang Bryan Lim, Nguyen Cong Luong, Zehui Xiong, Alia Asheralieva, Dusit Niyato, Cyril Leung, and Chunyan Miao. **A Survey of Coded Distributed Computing**. *CoRR* abs/2008.09048 (2020). arXiv: 2008.09048. URL: https://arxiv.org/abs/2008.09048 (see page 179).

[NM18] Alexandru Nelus and Rainer Martin. **Gender Discrimination Versus Speaker Identification Through Privacy-Aware Adversarial Feature Extraction**. In: *Speech Communication; 13th ITG-Symposium*. 2018, 1–5 (see page 124).

[NM21] Alexandru Nelus and Rainer Martin. **Privacy-Preserving Audio Classification Using Variational Information Feature Extraction**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 2864–2877. DOI: 10.1109/TASLP.2021.3108063 (see pages 186, 187).

[OH19] Afshin Oroojlooyjadid and Davood Hajinezhad. **A Review of Cooperative Multi-Agent Deep Reinforcement Learning**. *CoRR* abs/1908.03963 (2019). arXiv: 1908.03963. URL: http://arxiv.org/abs/1908.03963 (see page 156).

[OMG14] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel. **TinySDN: Enabling multiple controllers for software-defined wireless sensor networks**. In: *2014 IEEE Latin-America Conference on Communications (LATINCOM)*. Nov. 2014, 1–6. DOI: 10.1109/LATINCOM.2014.7041885 (see page 180).

[PAA19] D. Praveen Kumar, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. **Machine learning algorithms for wireless sensor networks: A survey**. *Information Fusion* 49 (2019), 1–25. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2018.09.013. URL: https://www.sciencedirect.com/science/article/pii/S156625351830277X (see pages 27, 86).

[Pat+17] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. **Curiosity-Driven Exploration by Self-Supervised Prediction**. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, 2778–2787 (see page 201).

[Pro01] J.G. Proakis. **Digital Communications**. McGraw-Hill series in electrical and computer engineering : communications and signal processing. McGraw-Hill, 2001. ISBN: 9780071181839 (see page 53).

[PS08] Jan Peters and Stefan Schaal. **Reinforcement learning of motor skills with policy gradients**. *Neural Networks* 21:4 (2008). Robotics and Neuroscience, 682–697. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2008.02.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608008000701 (see page 17).

[Qua19] Qualcom. *The wireless edge transformation has begun*. https://www.qualcomm.com/media/documents/files/the-wireless-edge-transformation.pdf. April,2019 (see page 27).

[Rad+19] Roxana Radulescu, Patrick Mannion, Diederik M. Roijers, and Ann Nowé. **Multi-Objective Multi-Agent Decision Making: A Utility-based Analysis and Survey**. *CoRR* abs/1909.02964 (2019). arXiv: 1909.02964. URL: http://arxiv.org/abs/1909.02964 (see page 19).

[Ram20] Arunselvan Ramaswamy. *Theory of Deep Q-Learning: A Dynamical Systems Perspective*. 2020. arXiv: 2008.10870 [cs.LG]. URL: arxiv.org/abs/2008.10870 (see page 169).

[Rie05] Martin Riedmiller. **Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method**. In: *Machine Learning: ECML 2005*. Ed. by João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, 317–328. ISBN: 978-3-540-31692-3 (see page 18).

[Rig+16] R. Riggio, A. Bradai, D. Harutyunyan, T. Rasheed, and T. Ahmed. **Scheduling Wireless Virtual Networks Functions**. *IEEE Transactions on Network and Service Management* 13:2 (June 2016), 240–252. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2549563 (see page 29).

[RLH15] Hyun-Gyu Ryu, Sang-Keum Lee, and Dongsoo Har. **Data Transmission with Reduced Delay for Distributed Acoustic Sensors**. *International Journal of Distributed Sensor Networks* 11:11 (2015), 247612. DOI: 10.1155/2015/247612. eprint: https://doi.org/10.1155/2015/247612. URL: https://doi.org/10.1155/2015/247612 (see page 189).

[Roi+14] Diederik Marijn Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. **A Survey of Multi-Objective Sequential Decision-Making**. *CoRR* abs/1402.0590 (2014). arXiv: 1402.0590. URL: http://arxiv.org/abs/1402.0590 (see page 19).

[RSN20] Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. **Multi-objective reinforcement learning for the expected utility of the return**. English. In: 2018 Adaptive Learning Agents, ALA 2018 - Co-located Workshop at the Federated AI Meeting, FAIM 2018 ; Conference date: 14-07-2018 Through 15-07-2018. July 2020 (see page 19).

[RTL20] Majid Raeis, Ali Tizghadam, and Alberto Leon-Garcia. **Reinforcement Learning-based Admission Control in Delay-sensitive Service Systems**. *CoRR* abs/2008.09590 (2020). arXiv: 2008.09590. URL: https://arxiv.org/abs/2008.09590 (see page 86).

[RWO15] Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. **Computing Convex Coverage Sets for Faster Multi-Objective Coordination**. *J. Artif. Int. Res.* 52:1 (Jan. 2015), 399–443. ISSN: 1076-9757 (see page 19).

[SAC13] G. Di Stasi, S. Avallone, and R. Canonico. **Virtual network embedding in wireless mesh networks through reconfiguration of channels**. In: *IEEE 9th Intl. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Oct. 2013, 537–544. DOI: 10.1109/WiMOB.2013.6673410 (see page 29).

[Sah+21] Olimpiya Saha, Guohua Ren, Javad Heydari, Viswanath Ganapathy, and Mohak Shah. **Deep Reinforcement Learning Based Online Area Covering Autonomous Robot**. In: *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. 2021, 21–25. DOI: 10.1109/ICARA51699.2021.9376477 (see page 151).

[SB18] Richard S. Sutton and Andrew G. Barto. **Reinforcement Learning: An Introduction**. Second. The MIT Press, 2018 (see pages 15, 16, 18, 156).

[SBB13] Sukhwinder Sharma, Rakesh Kumar Bansal, and Savina Bansal. **Issues and Challenges in Wireless Sensor Networks**. In: *2013 International Conference on Machine Intelligence and Research Advancement*. 2013, 58–62. DOI: 10.1109/ICMIRA.2013.18 (see page 123).

[SBV10] Manohar Shamaiah, Siddhartha Banerjee, and Haris Vikalo. **Greedy sensor selection: Leveraging submodularity**. In: *49th IEEE Conference on Decision and Control (CDC)*. 2010, 2572–2577. DOI: 10.1109/CDC.2010.5717225 (see page 124).

[SC18] Christopher Stanton and Jeff Clune. **Deep Curiosity Search: Intra-Life Exploration Improves Performance on Challenging Deep Reinforcement Learning Problems**. *CoRR* abs/1806.00553 (2018). arXiv: 1806.00553. URL: http://arxiv.org/abs/1806.00553 (see page 65).

[Sch+17a] J. Schmalenstroeer, J. Heymann, L. Drude, C. Boeddecker, and R. Haeb-Umbach. **Multi-Stage Coherence Drift Based Sampling Rate Synchronization for Acoustic Beamforming**. *IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)* (2017) (see pages 31, 41, 71, 149, 179).

[Sch+17b] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. **Proximal Policy Optimization Algorithms**. *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (see page 88).

[Sel+17] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga. **Practical Service Placement Approach for Microservices Architecture**. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. May 2017, 401–410. DOI: 10.1109/CCGRID.2017.28 (see page 29).

[Sha48] C. E. Shannon. **A mathematical theory of communication**. *The Bell System Technical Journal* 27:3 (1948), 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x (see page 103).

[Sia+17] Mennatullah Siam, Sara Elkerdawy, Martin Jagersand, and Senthil Yogamani. **Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges**. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, 1–8. DOI: 10.1109/ITSC.2017.8317714 (see page 151).

[Sia+18] Mennatullah Siam, Heba Mahgoub, Mohamed Zahran, Senthil Yogamani, Martin Jagersand, and Ahmad El-Sallab. **MODNet: Motion and Appearance based Moving Object Detection Network for Autonomous Driving**. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, 2859–2864. DOI: 10.1109/ITSC.2018.8569744 (see page 151).

[SK18] Mikhail B. Salin and Dmitrii A. Kosteev. *Examples of usage of nearfield acoustic holography methods for far field estimations: Part 1. CW signals*. 2018. arXiv: 1812.03826 [eess.AS] (see page 153).

[Sko+18] Lea Skorin-Kapov, Martín Varela, Tobias Hoßfeld, and Kuan-Ta Chen. **A Survey of Emerging Concepts and Challenges for QoE Management of Multimedia Services**. *ACM Trans. Multimedia Comput. Commun. Appl.* 14:2s (May 2018). ISSN: 1551-6857. DOI: 10.1145/3176648 (see page 123).

[Soc+12] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. **Semantic Compositionality Through Recursive Matrix-Vector Spaces**. In: *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2012 (see page 165).

[SPS99] Richard S. Sutton, Doina Precup, and Satinder Singh. **Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning**. *Artificial Intelligence* 112:1 (1999), 181–211. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(99)00052-1 (see pages 165, 201).

[Sun+22] Jie Sun, Yi Zhang, Feng Liu, Huandong Wang, Xiaojian Xu, and Yong Li. **A survey on the placement of virtual network functions**. *Journal of Network and Computer Applications* 202 (2022), 103361. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103361. URL: https://www.sciencedirect.com/science/article/pii/S1084804522000285 (see page 7).

[TA22] Joerg Schmalenstoeer Tobias Gburrek and Haitham Afifi. *Signal synchronization using online weighted average coherence drift (Demo at IWAENC 2022)*. 2022 (see pages 9, 188).

[Tes94] Gerald Tesauro. **TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play**. *Neural Computation* 6:2 (1994), 215–219. DOI: 10.1162/neco.1994.6.2.215 (see page 18).

[Uni21] International Telecommunication Union. 2021. URL: https://www.itu.int/rec/R-REC-P.1238-10-201908-I/en (see page 153).

[VL16] K. W. Jonhson V. M. Tiscareno and C. H. Lawrence. "Systems and methods for receiving infrared data with a camera designed to detect images based on visible light." June 2016 (see page 180).

[Vod] Vodafone Research Lab. *DreamLab by Vodafone*. https://www.vodafone.com/vodafone-foundation/focus-areas/dreamlab-app. Accessed: 2022-12-10 (see page 179).

[W B12] B. Gillett W. Briggs L. Cochran. **Calculus for Scientists and Engineers**. Pearson, 2012 (see pages 54, 55).

[Wan+19] 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* (2019), 1–1. ISSN: 2376-4562. DOI: 10.1109/TETC.2019.2902661 (see page 66).

[WCY20] Shuyi Wang, Haotong Cao, and Longxiang Yang. **A Survey of Service Function Chains Orchestration in Data Center Networks**. In: *2020 IEEE Globecom Workshops (GC Wkshps*. 2020, 1–6. DOI: 10.1109/GCWkshps50303.2020.9367463 (see page 7).

[WK05] Xin Wang and K. Kar. **Throughput modelling and fairness issues in CSMA/CA based ad-hoc networks**. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 1. 2005, 23–34 vol. 1. DOI: 10.1109/INFCOM.2005.1497875 (see page 102).

[WSM08] C. Wang, Y. Sun, and H. Ma. **Analysis of Data Delivery Delay in Acoustic Sensor Networks**. In: *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Vol. 1. 2008, 283–287. DOI: 10.1109/EUC.2008.21 (see pages 189, 190).

[WW19] Zhong-Qiu Wang and DeLiang Wang. **Combining Spectral and Spatial Features for Deep Learning Based Blind Speaker Separation**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27:2 (2019), 457–468. DOI: 10.1109/TASLP.2018.2881912 (see page 124).

[WZ15] Kyle Hollins Wray and Shlomo Zilberstein. **Multi-Objective POMDPs with Lexicographic Reward Preferences**. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, 1719–1725. ISBN: 9781577357384 (see page 19).

[Xu+18] Xingwang Xu, Zisheng Cao, Hualiang Qiu, Mingyu Wang, and Xiaozheng Tang. *Unmanned aerial vehicle (UAV) for collecting audio data*. US Patent 9,889,931. Feb. 2018 (see pages 149, 155).

[Yao+20] Haipeng Yao, Sihan Ma, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Song Guo. **A Continuous-Decision Virtual Network Embedding Scheme Relying on Reinforcement Learning**. *IEEE Transactions on Network and Service Management* 17:2 (2020), 864–875. DOI: 10.1109/TNSM.2020.2971543 (see page 86).

[ZB13] Yiran Zhao and Lori Breslow. **Literature review on hybrid/blended learning**. *Unpublished manuscript, HarvardX, Harvard University, Cambridge, MA* (2013) (see page 123).

[ZH14] Yuan Zeng and Richard C. Hendriks. **Distributed Delay and Sum Beamformer for Speech Enhancement via Randomized Gossip**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22:1 (2014), 260–273. DOI: 10.1109/TASLP.2013.2290861 (see page 124).

[Zha+13] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. **A unified enhanced particle swarm optimization-based virtual network embedding algorithm**. *International Journal of Communication Systems* 26:8 (2013), 1054–1073. DOI: https://doi.org/10.1002/dac.1399. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.1399. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.1399 (see page 65).

[Zha+18] Jie Zhang, Sundeep Prabhakar Chepuri, Richard Christian Hendriks, and Richard Heusdens. **Microphone Subset Selection for MVDR Beamformer Based Noise Reduction**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26:3 (2018), 550–563. DOI: 10.1109/TASLP.2017.2786544 (see page 124).

[Zha+19] Jian Zhang, Yaozong Pan, Haitao Yang, and Yuqiang Fang. **Scalable Deep Multi-Agent Reinforcement Learning via Observation Embedding and Parameter Noise**. *IEEE Access* 7 (2019), 54615–54622. DOI: 10.1109/ACCESS.2019.2913235 (see page 156).

[ZHH18] Jie Zhang, Richard Heusdens, and Richard Christian Hendriks. **Rate-Distributed Spatial Filtering Based Noise Reduction in Wireless Acoustic Sensor Networks**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26:11 (2018), 2015–2026. DOI: 10.1109/TASLP.2018.2851157 (see page 124).

[ZMS20] Marco Zimmerling, Luca Mottola, and Silvia Santini. **Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services**. *ACM Comput. Surv.* 53:6 (Dec. 2020). ISSN: 0360-0300. DOI: 10.1145/3410159 (see page 100).

List of Publications

Articles in Refereed Conference Proceedings

[1] **MARVELO: Wireless virtual network embedding for overlay graphs with loops**. In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. 2018, 1–6. DOI: 10.1109/WCNC.2018.8377194. Joint work with Sébastien Auroux and Holger Karl.

[2] **Reinforcement Learning for Autonomous Vehicle Movements in Wireless Sensor Networks**. In: *ICC 2021 - IEEE International Conference on Communications*. 2021, 1–6. DOI: 10.1109/ICC42927.2021.9500318. Joint work with Haitham Afifi, Arunselvan Ramaswamy, and Holger Karl.

[3] **Reinforcement Learning for Autonomous Vehicle Movements in Wireless Multimedia Applications**. In: *Pervasive and Mobile Computing*. submitted. 2022. Joint work with Haitham Afifi, Arunselvan Ramaswamy, and Holger Karl.

[4] **Reinforcement Learning-based Microphone Selection in Wireless Acoustic Sensor Networks Considering Network and Acoustic Utilities**. In: *Speech Communication; 14th ITG Conference*. 2021, 1–5. Joint work with Michael Guenther, Andreas Brendel, Holger Karl, and Walter Kellermann.

[5] **Sparse Adaptation of Distributed Blind Source Separation in Acoustic Sensor Networks**. In: *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2019, 190–194. DOI: 10.1109/WASPAA.2019.8937194. Joint work with Michael Günther, Andreas Brendel, Holger Karl, and Walter Kellermann.

[6] **Network-Aware Optimal Microphone Channel Selection in Wireless Acoustic Sensor Networks**. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, 820–824. DOI: 10.1109/ICASSP39728.2021.9414528. Joint work with Michael Gunther, Andreas Brendel, Holger Karl, and Walter Kellermann.

[7] **A Genetic Algorithm Framework for Solving Wireless Virtual Network Embedding**. In: *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2019, 1–6. DOI: 10.1109/WiMOB.2019.8923271. Joint work with Konrad Horbach and Holger Karl.

[8] **An Approximate Power Control Algorithm for a Multi-Cast Wireless Virtual Network Embedding**. In: *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*. 2019, 95–102. DOI: 10.23919/WMNC.2019.8881324. Joint work with Holger Karl.

[9] **Power allocation with a wireless multi-cast aware routing for virtual network embedding**. In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2019, 1–4. Joint work with Holger Karl.

[10] **Reinforcement Learning for Virtual Network Embedding in Wireless Sensor Networks**. In: *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2020, 123–128. DOI: 10.1109/WiMob50308.2020.9253442. Joint work with Holger Karl.

[11] **Data-driven Time Synchronization in Wireless Multimedia Networks**. In: *IWCMC 2022 Multimedia Symposium (IWCMC 2022 Multimedia)*. Dubrovnik, Croatia, May 2022. Joint work with Holger Karl, Tobias Gburrek, and Joerg Schmalenstroeer.

[12] **A Reinforcement Learning QoI/QoS-Aware Approach in Acoustic Sensor Networks**. In: *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. 2021, 1–6. DOI: 10.1109/CCNC49032.2021.9369626. Joint work with Arunselvan Ramaswamy and Holger Karl.

[13] **MARVELO - A Framework for Signal Processing in Wireless Acoustic Sensor Networks**. In: *Speech Communication; 13th ITG-Symposium*. 2018, 1–5. Joint work with Joerg Schmalenstroeer, Joerg Ullmann, Reinhold Haeb-Umbach, and Holger Karl.

[14] **Reinforcement Learning for Admission Control in Wireless Virtual Network Embedding**. In: vol. abs/2110.01262. 2021. arXiv: 2110.01262. URL: https://arxiv.org/abs/2110.01262. Joint work with Fabian Sauer and Holger Karl.

Poster

[15] **Acoustic signal extraction and enhancement over acoustic sensor networks (Demo at ITG conference on Speech Communication)**. In: *ITG conference on Speech Communication*. 2018. Joint work with Markus Bachman and Andreas Brendel.

[16] **Acquisition of Asynchronous Data and Parameter Estimation based on Double-Cross- Correlation Processor with Phase Transform (Demo at WASPAA 2021)**. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2021. Joint work with Aleksej Chinaev.

[17] **A Rapid Prototyping for Wireless Virtual Network Embedding using MARVELO**. In: *2019 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2019) (Demo)*. 2019. Joint work with Holger Karl, Sebastian Eikenberg, Arnold Mueller, Lars Gansel, Alexander Makejkin, Kai Hannemann, and Rafael Schellenberg.

[18] **Privacy-perversing Adversial Feature Extraction in Speaker Classification Tasks (Demo at WASPAA 2021)**. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2022. URL: https://ruhr-uni-bochum.sciebo.de/s/flwOSPlsp6fYAKi. Joint work with Alexander Nelus.

[19] **Signal synchronization using online weighted average coherence drift (Demo at IWAENC 2022)**. In: *International Workshop on Acoustic Signal Enhancement (IWAENC 2022)*. 2022. Joint work with Joerg Schmalenstoeer Tobias Gburrek.

[20] **Distributed Processing Plattform for Wireless acoustic sensor networks (Demo at IWAENC 2022)**. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2022. Joint work with Tobias Gburrek Teamproject 2019 and Joerg Schmalenstoeer.