Michał Barciś

Information Distribution in Multi-Robot Systems

DISSERTATION

submitted in fulfilment of the requirements for the degree of
Doktor der Technischen Wissenschaften

Universität Klagenfurt
Fakultät für Technische Wissenschaften

Supervisors

Univ.-Prof. Dipl.-Ing. Dr. Hermann Hellwagner
Universität Klagenfurt, Austria
Institut für Informationstechnologie

Univ.-Prof. Dipl.-Ing. Dr. Bernhard Rinner
Universität Klagenfurt, Austria
Institut für Vernetzte und Eingebettete Systeme

Expert Reviewers

Univ.-Prof. Dipl.-Ing. Dr. Hermann Hellwagner
Universität Klagenfurt, Austria
Institut für Informationstechnologie

Prof. Enrico Natalizio
Université de Lorraine, France
LORIA

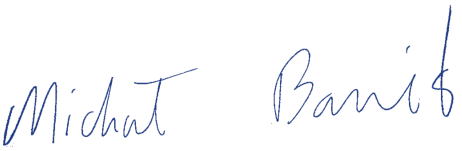
Klagenfurt, 2021

Affidavit

I hereby declare in lieu of an oath that

- the submitted academic paper is entirely my own work and that no auxiliary materials have been used other than those indicated,
- I have fully disclosed all assistance received from third parties during the process of writing the thesis, including any significant advice from supervisors,
- any contents taken from the works of third parties or my own works that have been included either literally or in spirit have been appropriately marked and the respective source of the information has been clearly identified with precise bibliographical references (e.g. in footnotes),
- to date, I have not submitted this paper to an examining authority either in Austria or abroad and that
- when passing on copies of the academic thesis (e.g. in bound, printed or digital form), I will ensure that each copy is fully consistent with the submitted digital version.

I understand that the digital version of the academic thesis submitted will be used for the purpose of conducting a plagiarism assessment. I am aware that a declaration contrary to the facts will have legal consequences.



Michał Barciś, m.p.

Klagenfurt, October 2021

Abstract

This thesis addresses the problem of information distribution in multi-robot systems, which can be intuitively characterized as a decision process answering the questions of what and how much information is worth exchanging in a group of robots. In practice, it is often impossible to share all observations among the collaborating agents. Trying to do so could overload the network and vastly decrease its performance. Hence, robots need to limit the amount of exchanged data by choosing which information is of higher importance. Typically, this is achieved by manually tuning the robotic applications for the specific communication interface. In this work, we present a generic solution that allows the robots to make this decision autonomously.

We start with an evaluation model that is able to quantify the utility of exchanged messages. Then, we introduce an optimization middleware that uses the aforementioned model. It enables the robot to autonomously make a decision if a given message should be sent or not, taking into account constrained communication resources. The middleware is based on Monte-Carlo Tree Search (MCTS). We show two ways to improve the performance of the middleware for the problem of information distribution.

To provide an estimate of how much throughput can be utilized, we propose an adaptive goodput constraint. The constraint applies the state of the art in network congestion control to robotic networks. By utilizing machine learning, the constraint is able to adapt on the fly to changing communication conditions.

The performance and the feasability of the approaches is evaluated using simulation-based study. Finally, we present the practical applicability of the proposed system in two robotic demonstrators. The first one utilizes small mobile robots working in a lab and streaming images to a base station, autonomously prioritizing images with higher utility. The second demonstrator shows a similar behavior incorporated into a mission performed by Unmanned Aerial Vehicles.

Contents

1 *Introduction, motivation, and contributions*

The primary problem addressed in this thesis can be briefly characterized as answering the questions *what* and *how much* information is worth exchanging in a group of robots. Sharing the information obtained by robots with other collaborating peers often allows to increase the performance of a mission at hand. However, transmitting all available data leads to overloading the network. It results in significantly increased latency and message loss, which renders the communication inefficient or often even unusable. Hence, a robot needs to make the decision what information to communicate. This thesis explores options to make such decisions in an autonomous way, taking into account the mission at hand and improving the performance of information distribution.

This chapter provides a non-technical background for the whole thesis. It starts by outlining the context in which this work was conducted (section 1.1) and describes a very initial attempt that was made to understand the involved problems (section 1.2). Then, the characteristics that make the introduced setting challenging and interesting are presented (section 1.3). Finally, it describes the contributions and novelties introduced in this thesis (section 1.4).

This thesis contains parts of already published work [1–4] including contributions from co-authors: Hermann Hellwagner, Agata Barciś, and Nikolaos Tsiogkas.

Contents

1.1 *Karl Popper Kolleg on Networked Autonomous Aerial Vehicles (NAV KPK)*

The Karl Popper Doktorats- und Wissenschaftskolleg (KPK) on Networked Autonomous Aerial Vehicles (NAV) was a research project conducted at the University of Klagenfurt from autumn 2017 to summer 2021. This thesis is one of the outcomes of this project. The main goal of the NAV KPK was to advance collaborative research in the field of NAV at the university. Hence, it was envisioned as four distinct, inter-operating parts developed by people associated with different departments.

In order to propel the research and initiate collaborations a common project goal had been chosen: *autonomous 3D reconstruction* of a real, unknown environment using a group of unmanned aerial vehicles (UAVs). The UAVs use single and stereo camera setups and explore the specified perimeter. This exploration should result in a 3D scene with variable information density. That means, the places that the application deemed more interesting should be reconstructed in greater details. Figure 1.1 presents the visualization of the concepts addressed in the envisioned application. University of Klagenfurt

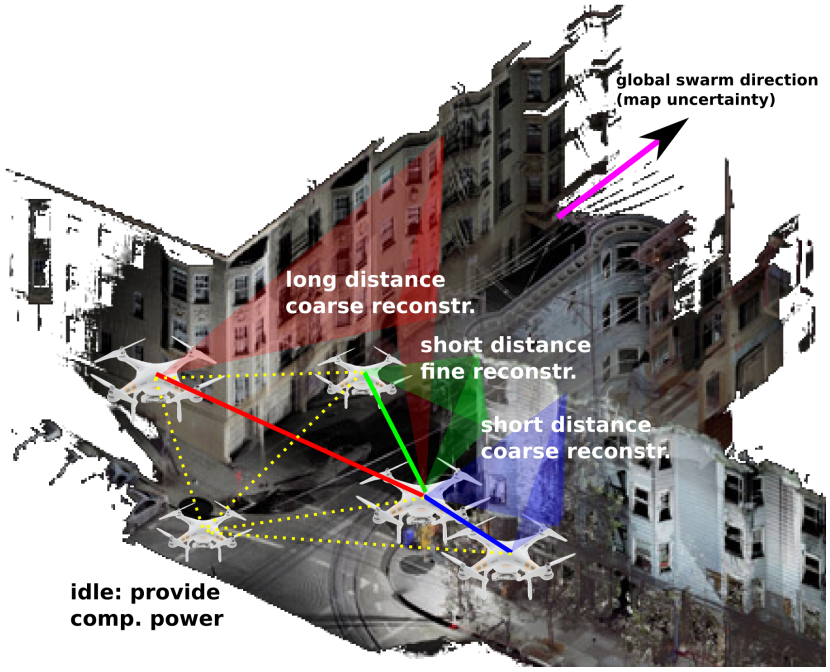


Figure 1.1: Autonomous group of UAVs performing a 3D reconstruction of an unknown environment. Different colors symbolize different levels of details of the reconstruction based on the distance to reconstructed surface. Source: KPK NAV project proposal; modified with permission from a work by Scott Page Design (http://www.scottpagedesign.com/).

had a very strong background needed to initiate this kind of the project. Most importantly, it already had a portfolio of drone-related projects. Among them are projects cDrones[1], SINUS[2], Forest-iMate[3]. All of these efforts are gathered on the *Dronehub K*[4] portal that summarizes all ongoing drone-related work at the university.

As mentioned previously, the KPK project was split into four parts. Each part had a PhD student assigned to it. Each student was supervised by a professor from one of the four different research groups. A photo of the core KPK members together with the Dean is presented in figure 1.2. The parts were as follows.

1. *Autonomous Navigation and Control* — realized by Roland Jung and supervised by prof. Stephan Weiss. The work was focused on collaborative state estimation of UAVs. Roland proposed new

[1] https://campus.aau.at/cris/project/0f4dc44845ae29440145b1a5d23431e8?lang=en
[2] https://itec.aau.at/item/sinus/
[3] https://forest.aau.at/
[4] https://uav.aau.at/

Figure 1.2: Core members of the KPK project together with the Dean holding Crazyflie drones during the opening of the new drone hall at the university, December 2019. Back row, left to right: prof. Christian Bettstetter, prof. Hermann Hellwagner, prof. Gerhard Friedrich (the Dean), prof. Bernhard Rinner, prof. Stephan Weiss. Front row, left to right: Michał Barciś, Agata Barciś, Petra Maždin, Roland Jung.

methods to utilize information about the state of other peers in order to improve the state estimation of each UAV.

2. *Coordination and Mission Planning* — realized by Petra Maždin and supervised by prof. Bernhard Rinner. Petra tackled a problem of multi-coverage, i.e., how to cover multiple points of interest using a set of drones. She addressed it by proposing a novel task assignment method tailored for this specific use-case.

3. *Communications and Networking* — realized by Michał Barciś (author of this thesis) and supervised by prof. Hermann Hellwagner. Initially the work was focused more on communication and setting up the infrastructure for the project, but soon transformed into the ideas described in this thesis.

4. *Network Synchronization* — realized by Agata Barciś and supervised by prof. Christian Bettstetter. Agata investigated ways to utilize mutual coupling between temporal and spatial coordination in a robotic setting.

Even though the parts are not directly related to each other our work greatly benefited from ideas borrowed from other fields. At the end of the project, all four parts were connected in a joint demonstrator showcasing a 3D reconstruction with a group of drones.

1.2 *Information distribution case study*

At the early stage of the KPK project we wanted to better understand the specifics of information distribution in multi-robot systems. In order to do that we have performed a case study. We envisioned what types of information could be exchanged in our final project and what would be their characteristics.

Seven different examples of information types were studied:

synchronization Messages used to synchronize agents. The information could be just a pulse (e.g., for the implementation of the firefly synchronization algorithm [5]) or a full timestamp (e.g., synchronization using *Network Time Protocol* (NTP)).

sync & swarm For the implementation of the sync & swarm model [6] agents need to periodically exchange phases and positions.

mission planning All information needed to plan movements and coordinate robots, e.g., planned trajectories, agents' bids in auction-based protocols, etc.

network diagnostics Diagnostic information for network management, e.g., the amount of transmitted network traffic by each agent.

sensor data (mission-related) Sensor data needed exclusively for the coordination algorithms, e.g., battery level, which could be used to decide that a UAV needs to finish the operation. Other information could also be used for coordination (e.g., position), but we do not consider it separately if it is already needed for a different application with stricter requirements.

sensor data (state estimation[5]) The data could be the same as in the case of sensor data for mission-related applications. However, when this data is used for state estimation, the requirements for frequency and latency are stricter than with mission-related sensor data. Hence, we decided to consider it separately.

generated data All data produced by agents that are not needed to support the operation, but rather are the result of the mission, e.g., terrain maps, pictures, 3D models, etc.

For each information type we have then estimated some characteristics: importance of this information, lifespan, frequency (how often this type of information needs to be exchanged), size, number of recipients and total goodput[6] generated by this information type. We assumed that 15 robots are performing the mission, because that was the maximum number of devices considered to be used in the KPK project. The results of this case study are presented in table 1.1.

[5] State estimation is a problem of determining the current state of a robot given the measurements from its sensors. Some of these measurements might be communicated from other robots (e.g., relative distance between two robots) and this exchanged data is the one we are characterizing here.

[6] Goodput is the amount of data exchanged on the application layer, not taking into account any protocol overheads. Throughout the thesis we use goodput instead of throughput to emphasize the application-layer focus of this work.

Table 1.1: Approximate goodput required by information types foreseen to be used in the KPK project.

field	information type	importance	lifespan	frequency [Hz]	size [kB]	recipients	recipients #	goodput [kbps]
synchronization	pulse	high	very short	higher than 1 Hz	0.01	all nearby	5	3.2
	timestamps	high	short	0.1 Hz	0.01	all nearby	5	0.32
sync & swarm	phase & position	medium/high	short	100 — 1 Hz	0.01	all nearby	5	160
mission planning	path plan	high	long	rare (0.01 Hz)	10	group	7	448
	mission commands	critical	medium	very rare (0.001 Hz)	0.1	broadcast	15	0.096
	decision making	high	medium	rare (0.01 Hz)	0.1	group	7	0.448
	points of interests	low	long	very rare (0.001 Hz)	0.01	broadcast	15	0.0096
sensor data (mission-related)	battery status	low	medium	each minute (0.017 Hz)	0.01	group/broadcast	7	0.00952
network diagnostics	signal strength	medium	short	1 — 0.1 Hz	0.01	all nearby	7	2.4
	througput, delay	medium	very short	1	0.01	all nearby	7	4.48
sensor data (state estimation)	IMU	very low	very short	100	0.01	group/partner	2	16
	rotor speeds	very low	very short	100	0.01	group/partner	2	128
	estimated position	medium	very short	10	0.01	group/partner	7	44.8
	estimated velocity	medium	very short	10	0.01	group/partner	7	44.8
	collaborative state estimation vector	low	very short	10	0.1	group/partner	2	128
generated data	positions of keyframes	low	infinite	5	0.01	depends on mission	15	48
	SLAM keyframes	low	infinite	5	100	depends on mission	3	96 000
	pictures/patches	low	infinite	5	1 000	depends on mission	2	640 000
	map updates	medium	infinite	rare	1000	depends on mission	5	32 000

This analysis led us to the formulation of the following three observations, which we identified to be the most important aspects to focus on:

1. Information types that are important and valid for a limited amount of time usually require relatively small amount of goodput. On the other hand, less important messages with long lifespan require many bytes of data to be exchanged.

2. If broadcast messages were to be sent separately to all recipients (i.e., they would use the amount of traffic proportional to the number of recipients), significant amount of additional traffic would be generated. Hence, it is desirable to pursue approaches allowing for broadcast messages and avoid generating amounts of traffic proportional to the number of recipients. In particular, whenever possible the broadcast messages should not be acknowledged.

3. In the case studied by us, information types generated frequently are usually less important than information types generated rarely. Therefore, usually it is not a problem if a high-frequency message is lost, but special care should be taken not to drop rare messages.

These observations were later used by us extensively while designing our experiments and robotic testbeds.

1.3 Motivation

Inspired by the KPK project theme and taking into account the mission characteristics obtained during the case study, we decided to realize the research in the following setting. We address the communication requirements of a fleet of small UAVs realizing a 3D reconstruction of an unknown environment without a centralized coordination. UAVs should explore the given area, locate objects of interest, capture photos of these objects, and send them to the base station. Then the base station processes the captured data to obtain 3D models based on these photos.

Such a mission creates unique communication challenges. It introduces multiple message types, for instance: sensor data (e.g., position, attitude, battery level, etc.), synchronization, mission commands, gathered images, and 3D models. Each of these message types has different characteristics, e.g., varying sending frequency, priority, size, and payload. These characteristics are often heavily dependent on the mission progress and status. For example, exchanging map fragments can be very useful at the beginning of the mission, when all agents explore the environment and contribute new information; conversely, at the end of the mission exchanging

map fragments may be of little use, since all agents already know the map and are focused on other mission objectives. Alternatively, two distant UAVs are not interested in the precise position of each other, but for UAVs operating in proximity this information might be crucial.

The communication conditions that such a UAV fleet operates in are dynamic: they depend on the proximity of the UAVs, their attitudes, obstacles between them, radio wave propagation, etc. Thus, it is almost impossible to plan the communication a priori and a communication system that adapts to the observed situation is beneficial.

1.4 Contributions

The setting presented in the previous section is relatively complex and influenced by many variables. In order to avoid the situation in which a solution would work just in one particular setting, we conceptualized a generic abstraction layer in our communication model and named it *information distribution layer*. The application logic interacts directly with this layer whenever it wants to communicate with other robots. Then, the information distribution process utilizes present communication interfaces in order to exchange the information. This concept is visualized in figure 1.3. The goal of the research described in this thesis was to design and implement the information distribution layer. While pursuing it the following main contributions were achieved.

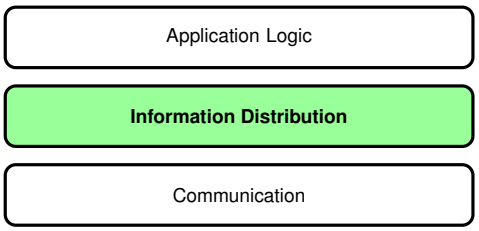


Figure 1.3: Visualization of the proposed architecture with the information distribution middle layer.

Evaluation model

The presented research could be informally summarized as finding an *efficient* way for robots to communicate. However, the term *efficient* is very dependent on the application and could have various meanings in different settings. This issue was addressed by defining a utility-based evaluation model for the problem of information distribution. The main goal of this model is to quantify how well the process of information distribution was realized during a given mission.

The results were initially presented during the Mission-Oriented Wireless Sensor, UAV and Robot Networking (MiSARN) INFOCOM Workshop [1]. After a positive feedback we decided to extend this work to a journal article published in *Sensors* [2]. The model deemed beneficial also for the problem of distributed task assignment [7], outside of the initially envisioned context . This contribution is described in detail in chapter 4.

Optimization middleware

The main contribution of this research is a method to efficiently coordinate communication in a multi-robot system. Our approach incorporates different information types in one framework and is efficient enough to be used on robots with limited computational power. Furthermore, it was realized as the aforementioned information distribution layer (figure 1.3), which allows to transparently incorporate it into an existing solution.

This contribution resulted in a journal publication in Frontiers in Robotics and AI [3]. In this thesis it is described in chapter 5.

Adaptation method

The optimization middleware incorporated various constraints: for instance, it is able to maintain application goodput below a predefined level. While using these constraints, we have realized that determining the value of the constraint is sometimes hard and a solution that autonomously detects this value would be beneficial. This resulted in the formulation of an adaptive goodput constraint based on perceived network latency. A similar solution could also be applied to other systems, so we decided to publish it as a separate article and present at the Wireless Days 2021 conference [4]. It is described in chapter 6 of this thesis.

Robotic experimental setup

The optimization middleware and the adaptive constraint were both extensively tested in simulation and on mobile robots. This resulted in creation of a testing environment that could be utilized to conduct similar experiments. The demonstrator based on this experimental setup is described in chapter 7.

Furthermore, during the work on the contributions we utilized multiple hardware platforms. They are described in section 3.1. All software developed while working on this thesis is open source and published under a permissive license on GitHub[7]. Parts were even used outside of the scope of this thesis: for instance, the Python module facilitating the use of NS-3 was used in the work on task allocation by Maždin et al. [7], and the Ansible scripts used to manage our robots were used throughout our department.

[7] https://github.com/zeroos/infdist/

2 State of the art

This chapter presents the state of the art of information distribution in multi-robot systems. It is divided into three sections corresponding to three main contributions, and chapters of this thesis. In section 2.1, we describe the works that directly address the same problem as this thesis does and we emphasize the differences and novelties of our approach. Then, in sections 2.2 and 2.3 we describe, respectively, the state of the art approaches to evaluation of information distribution and adaptation to varying communication conditions. As the introduction to this chapter we provide a brief overview of different approaches to communication of UAVs.

Exploring the challenges related to communication among UAVs is recently a popular topic. Surveys addressing it tackle different aspects of the problem. Multiple publications [8–10] study different communication requirements related to various applications of UAVs, including practical experiments with UAVs [11]. They characterize typical scenarios for UAV missions and specify what communication solution could be used in each of them. Interestingly, these publications suggest not only that the communication requirements for different missions might differ a lot, but even that sometimes during one mission these requirements might be changing [10]. Nevertheless, Hayat et al. [9] suggest that IEEE 802.11 might be a reasonable choice for UAV communication and that is also the technology we decided to utilize in the work described in this thesis.

Extensive experimental studies were conducted to confirm in practice the applicability of IEEE 802.11 [12, 13] and cellular [14, 15] networks. They show that each of these technologies might be a good choice, depending on the requirements. However, if cellular connections are to be used on a larger scale, a collaboration with network providers might be required, because nowadays these networks are optimized for ground users [15].

Recently, Shi et al. [16] published an in-depth analysis of communication protocols and various software solutions that facilitate the information exchange. The fact that there is a lot of active research in this field shows that the topic is of interest and there is a need for

Contents

improvements. In this thesis we approach this need from the application perspective and propose a way to improve software independent of the chosen hardware solution.

2.1 *Optimization of information distribution*

Advances in the field of *multi-robot systems* (MRS) have allowed sophisticated real-world applications in science and industry. Some examples include underwater archaeology [17, 18], search and rescue [19], and manufacturing [20].

The information distribution problem refers to the decision process that, in a broad sense, answers the questions *what* information to exchange, *when*, and *how much* to achieve a good performance of the whole system while obeying the limited resource utilization, such as available throughput or energy usage. The work by Zhivkov et al.[21] can serve as a good example of why this problem is important. They introduce a communication testbed and use it to show how different communication strategies can tremendously change the outcome of the robotic mission.

Despite the need to exchange a multitude of information in MRS, often the literature neglects the role of communication in such systems. Usually, it is either taken for granted or the main focus is on maintaining the connectivity [22]. Only recently, the problem of information distribution in MRS began to receive attention [17, 23–27].

In the problem of information distribution, the decision process aims to optimize the communication in MRS with specified constraints. To perform such an optimization, the importance of each communication event has to be assessed. Usually, the importance of these events depends on the mission currently executed by the MRS. Hence, domain knowledge about the mission has to be incorporated into the solution. This can be achieved in a direct or an indirect way.

A *direct* approach [23–25, 28, 29] simulates the course of the mission to estimate the impact of each communication event on the mission objective. Although such approaches guarantee the best performance for a specific application, they assume the ability to forward-simulate the mission, which might require considerable effort or be infeasible.

On the contrary, *indirect* approaches assign a utility to the data being processed during the mission. They communicate the most valuable (i.e., providing the highest utility) pieces of information, assuming this would positively impact the mission performance. A common way to define such utilities is given by information-theoretic measures such as KL-divergence [26, 30]. However, such approaches imply that the application logic takes into account information uncer-

tainty. In many practical applications this is not the case, for example, often outdated information is just used as if it were current. Therefore, it might be challenging to integrate these solutions into existing systems. Another indirect approach is to design utility functions tailored to a specific application [7, 31, 32]. Often, such approaches treat information in a myopic manner, i.e., taking into account only the immediate value of communication without evaluating its long-term impact. More sophisticated models are rarely considered [31–33]. In this thesis, we present a solution based on an indirect approach that takes into account the whole mission progress (i.e., is not myopic).

Another aspect to consider when speaking about the information distribution problem is how to define what information can be exchanged. It is commonly assumed that the whole state of an agent (e.g., a robot) can be shared during a single communication event [25, 28, 30]. This approach allows researchers to avoid the question of *what* to communicate and focus only on the question of *when* to communicate. Although in many cases it is fair to assume that all information could be transmitted in a single message, it is not applicable in a generic setting [26]. As an example, suppose the state of an agent consists of its position, battery level, and camera images. In that case, it might be enough to regularly send only the first two components and reduce the rate of a resource-hungry transmission of images.

Such a simplification results in a significant reduction of the decision space, which enables efficient computations [25]. This thesis presents an approach that avoids this assumption, while still benefiting from a similar reduction of the decision space in specific situations (see section 5.3 for details). To achieve this, we categorize all information exchanged among agents into *information types*. It is an abstract way to group together information items that possess similar properties (e.g., only the most recent information of a given type is important for a robot). For instance, an information type could be the position of a robot, the camera image or the map generated by all robots collaboratively.

In this thesis, we exploit an indirect approach as a way to decouple the information distribution optimization process from the mission at hand. This decoupling is possible thanks to the evaluation model (described in chapter 4) that provides a unified abstraction of a mission at hand to the optimization method (chapter 5). This allows us to propose a generic information distribution optimization middleware that is easily integrable into any robotic system communicating by exchanging messages, without any modifications to the application logic.

2.2 *Evaluation models*

In this section, we examine multiple fields and describe their approach to modeling the value of information distribution. We start with the works already described in the previous section, because they share the most similarities with our approach. However, we also describe other approaches that inspired our evaluation model: Quality of Service (QoS), utility-based approaches and information theory. We observe that most of the related work can be characterized either as widely applicable and generic (e.g., in the field of Quality of Service) or focused on a specific application (e.g., in the field of information distribution). Our model shares some properties from both of these worlds. It is generic enough to be used in a uniform way with a variety of information types. This property allowed us to define an optimization method that supports all messages exchanged during the mission. At the same time, it is possible to specialize the model's behavior for any information types taking into account their most important characteristics for a mission at hand. Because of these unique properties of the introduced model, we did not perform any direct quantitative comparisons with other approaches. Instead, in this section we argue how our approach differs from the state of the art and in section 4.4 we present a theoretical analysis which shows how generic the model is and what can be expressed by it.

Information distribution

Let us examine a bit closer some approaches to information distribution in robotic systems to see how they evaluate which information is more useful than another one. Gadd and Newman [34] present a pull-based, centralized solution for map distribution in a fleet of mobile vehicles. They compare multiple methods of choosing the data samples to download based on time and location. Cieslewski et al. [35] address the problem of decentralized data distribution for the simultaneous localization and mapping problem and tackle it by reducing the amount of exchanged data. Mahdoui et al. [36] present a decentralized solution to an exploration problem and choose which information to share by operating on meta information instead of the primary data. Such solutions are very specific and focus on one particular type of information for a particular mission; it is not clear how to generalize such an approach to other information types or how it would perform in other applications.

More generic approaches are also considered. A work by Marcotte et al. [24] presents an algorithm for optimizing communication under

bandwidth constraints. The approach assigns a value to each message by simulating the decisions of the agent given that a message was received and without this message. They present an efficient algorithm to compute these values, but they introduce multiple assumptions about the structure of the mission at hand. Best et al. [37] focus on optimizing the data distribution in a multi-robot mission and define the utility of a message as its impact on a mission objective. This impact is obtainable in some specific cases, but might be hard to estimate in general.

The main difference between the model proposed in this thesis and the literature mentioned above is that the proposed model focuses only on evaluation of the information distribution and is separate from the optimization method. The most straightforward application of the proposed model in this field is to use it in order to compare different solutions and find one that performs the best in the scenario at hand. But the model also addresses both of the problems mentioned earlier in this section: it is generic enough to be used comprehensively with all information types that might be required by an application and it does not need to explicitly know the influence of each message on the mission objective. Hence, in chapter 5 we present a method that makes use of the model in order to optimize communication.

Quality of Service (QoS)

Typically, evaluation methods are mostly focused on the QoS indicators, such as throughput/goodput, jitter, latency, etc. For instance, Krinkin et al. [38] compare latency and throughput achieved by multiple *Data Distribution Service* (DDS) implementations by sending 100 messages in a row and measuring the reception time of the first and the last message. Similar criteria together with jitter are used by Tardioli et al. [39] in order to compare multiple data distribution algorithms in Robot Operating System (ROS). Martin et al. [40] focus on the influence of security features on latency. Maruyama et al. [41] take into account not only latency, but also examine achieved throughput and introduce multiple capabilities that a communication framework might have (e.g., the ability to send messages of 2 MB in size without data loss using their experimental setup). In a study focused on real-time applications, Oh et al. [42] use *deadline meet ratio* as one of the criteria. Tornell et al. [43] perform analytical evaluation of their algorithm and validate the results using experiments. In addition to the delay criterion, they also examine the number of exchanged messages.

In contrast to our method, all of the approaches mentioned so far

in this section focus on the technical aspects and try to objectively measure parameters of the network. However, exchanging lots of data with minimum latency does not automatically imply that these exchanges are useful. Our approach addresses this problem and aims at capturing the actual value that communication provides for the application at hand.

Many techniques are being developed for measuring Quality of Experience (QoE) (e.g., PEVQ [44] for video streaming). Such techniques are robust and widely used for specific use cases, but it is not possible to generalize them for broader use. They could be incorporated into the proposed model, but on their own they are not able to evaluate the whole information distribution. We have shown how a QoE-based metric can be incorporated into the evaluation model introduced in this thesis for evaluating the performance of video streaming in section 4.3.

By analogy to QoE (and its basis on QoS), we could say that our proposed model aims at quantifying the "quality of information" for robots; i.e., the model, while building upon QoS, on a higher level captures the usefulness of information by considering the requirements of robotic interactions and missions.

Utility-based approaches

Utility-based approaches have proved to be beneficial in many related fields, including communication. D'Aronco and Frossard [45] present an interesting way to manage network congestion in order to maximize the utility gained by network nodes, with unspecified utility functions. Kimura et al. [46] use utilities in order to solve a message routing problem. They calculate utilities for each pair of nodes and base it on the encounter frequency. Then, they use the computed value in order to make a decision whether to send a message or not. The field of real-time computing also uses utility-based definitions extensively. Raut et al. [47] consider a scheduling problem in which the job's value deteriorates over time. Li et al. [48] present a heuristic to solve a similar problem, but using arbitrary utility functions. Gan et al. [49] apply an exponentially decaying utility function in order to address the emergency vehicles scheduling problem.

There are multiple publications on utility-based evaluation of communication, but they focus mainly on multimedia transmissions over the Internet [50, 51] and are not directly applicable to robots.

The widespread use of utility functions motivated us to utilize it in our definition of the model. The main difference between our approach and the presented ones is that we calculate the utility continuously at each point in time and integrate it in order to get a total

value of utility. This approach is more intuitive to work with and allows us to easily express information value achieved by combining multiple messages.

Information theory

The field of information theory [52] also considers very similar problems to the ones addressed by our evaluation model, but focuses on the amount of information provided by exchanged data and not on the usefulness of this information for the mission at hand. Nevertheless, it was successfully employed in order to estimate the utility of communication in a work presented by Williamson et al. [30, 53]. The authors also presented the application of their approach using multiple benchmark problems.

In order to use such solutions the coordination algorithms need to be adjusted, which means that it is not possible to apply them to all types of information in existing systems. The aim of our model is to create a generic solution to handle all information types. Similarly to approaches presented above in the discussion about Quality of Service, solutions based on information theory could be incorporated into our model in the definitions of specific information types.

Approaches trying to incorporate the notion of the knowledge of agents are gaining popularity. One of them is presented by Alshehri et al. [54]. They utilize a framework for epistemic reasoning [55, 56] on what to communicate in order to improve multi-robot coordination. Such work definitely has a huge potential improving the performance of multi-robot coordination. However, similarly to approaches based on information theory, the epistemic reasoning is very closely connected to the application at hand and to the knowledge about the world. Hence, for many practical systems it might be infeasible to incorporate such reasoning.

2.3 *Adapting to varying communication conditions*

Throughout this thesis, we have already mentioned multiple publications addressing the general problem of what information, when, and with which agents to share in MRS [3, 24–29]. Often they take into account communication limitations by introducing a limit of the amount of data that can be exchanged during some time period. We call such a limit *goodput constraint*. Whenever goodput constraints are introduced, it is assumed that the network conditions are known or are easy to measure. Hence, often adaptation to network conditions is not included in such approaches.

On the other hand, the problem of adapting to network condi-

tions is extensively studied and used in the field of video streaming [57–59]. However, the results are usually not directly applicable in robotic systems, as they tend to depend on measurements of available throughput over extended periods. In robotic systems, periods of inactivity followed by extensive use of the network are common (e.g., when an object of interest is detected), so measurements of available throughput are less reliable.

However, a previous work on video streaming in a robotic system conducted in our group [60] helped us identify that network latency measurements can serve as a good indicator for deciding how much throughput to use. Additionally, Li et al. [61] presented an interesting approach to video adaptation based on TCP congestion control algorithms. These two papers motivated us to investigate the applicability of delay-based congestion control methods to a generic problem of information distribution.

Latency-aware methods are often investigated and utilized in robotics. For instance, Böhmer et al. [62] show how to leverage an open communication stack to benefit from communication with predictable latency.

Multiple papers addressed the problem of congestion control on the application layer. For instance, Liu et al. [63] presented a congestion control algorithm designed for P2P networks. They utilize knowledge about throughput to detect over-use of the network — a decrease in achieved throughput suggests congestion and triggers a congestion avoidance algorithm that reduces the number of concurrent P2P connections. Conversely, an increase in achieved throughput suggests that the number of concurrent P2P connections can be increased. Kohler et al. [64] proposed a generic UDP congestion control method. Similar to TCP congestion control algorithms, they base their method on loss rate, which is periodically communicated from receiver to sender, introducing additional overhead. Both of these methods detect congestion when it occurs. Utilizing a delay-based approach could allow us to identify initial symptoms of congestion, thus avoiding it entirely. In particular, in this thesis, we take into account that robots usually have access to a synchronized time source and hence can measure end-to-end delay (also known as one-way latency). It allows us to detect early network congestion symptoms on the application layer without introducing additional communication overhead.

Delay-based congestion control algorithms have been known for years. TCP Vegas [65, 66] is a classic example thereof. However, the main issue of such algorithms is that their performance degrades when inter-operate in the same network with loss-based methods. Delay-based methods detect symptoms of congestion earlier. Thus,

they reduce the sending rate earlier than loss-based algorithms, resulting in an unfair network share. Recent approaches to delay-centric congestion control, e.g., DCCC [67] or BBR [68], address this problem. However, in robotic systems, this issue is not severe. Robots usually collaborate and have common goals, so there is no reason for such predatory behavior to emerge.

In this thesis, chapter 6 aims at transferring the state of the art of congestion control algorithms into the field of information distribution in MRS and applying it to the middleware introduced in chapter 5.

3 Technical infrastructure

Contents

3.1 Hardware platforms

Nowadays, there are many robotic platforms available for purchase. Out of the box, these platforms usually have a very specific and often limited set of features. At the beginning of the work on the KPK project, we needed to identify which of these features are the most important, choose a platform that has the most of them and extend it by adding other necessary capabilities.

This chapter describes the solution we have eventually decided to use. Although since the very beginning we were working in a project focused on drones, we decided to acquire also ground robots that will allow us to prototype quickly and then transfer the ready solutions to drones.

Raspberry Pi testbed

The first requirement our team formulated was that the platform should be able to run ROS. We all had a very positive experience with this framework and there is plenty of software already available that supports it. Furthermore, it facilitates the integration of different components. This aspect was important for us, because the KPK project involved multiple, initially independent parts, which were later on merged together.

However, most computers capable of running ROS are rather big and relatively expensive. The size means that a larger robot is required (which again increases the price) and high costs of a single robot in a multi-robot systems quickly add up.

These thoughts led us to the choice of *Raspberry Pi 3B+*. For a reasonable price (less than 40 EUR) and in a small form factor it packs good computational power (thanks to a 1.4 GHz quad-core processor) with great connectivity (802.11ac wireless LAN, Bluetooth 4.2 and multiple hardware interfaces, e.g., USB, UART, I2C).

But perhaps the most useful aspect of this platform is the size of

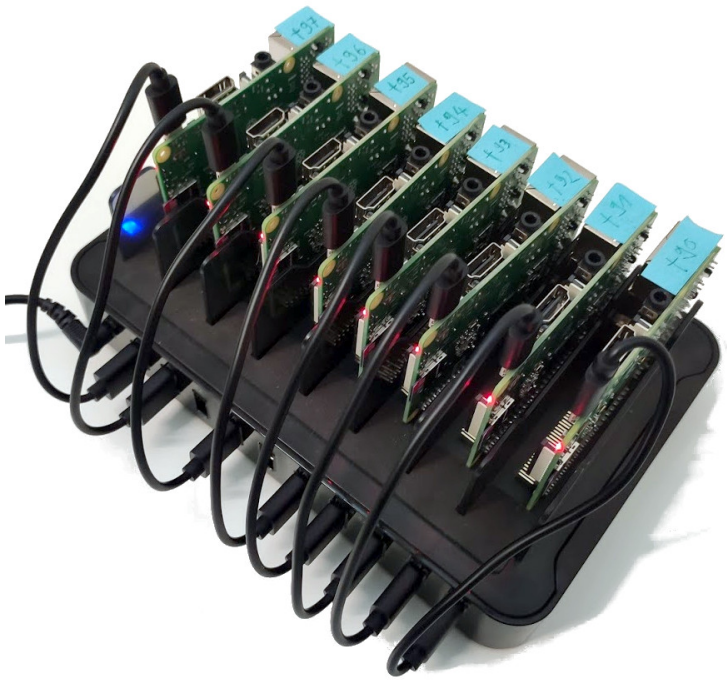


Figure 3.1: Photograph of the testbed consisting of eight Raspberry Pi B+.

the community around it. Most of the software we wanted to utilize was already tested on Raspberry Pi with many problems described and solved. We also expected it to be one of the first hardware platforms to have a ROS 2 version pre-compiled for and we were right.

At the end we included a Raspberry Pi in all our robots. In order to facilitate the development process we have also created a testbed (presented in figure 3.1) consisting of eight devices and a compact charger. All the devices are connected using a wireless network, so from the programming perspective the usage is similar to a situation when the devices are mounted on robots. At the same time it is more convenient to set up, cheaper and easier to maintain than robots.

The testbed was used to perform experiments for multiple publications during the KPK project [2, 7].

Pololu Balboa

The first multi-robot system (MRS) platform we have designed during the KPK project was in collaboration with prof. Christian Bettstetter from the Mobile Systems group at the University of Klagenfurt. The initial idea was focused around a demonstrator for synchronous behaviors of robots, but we decided to use this opportunity to create a platform that would serve as a middle-ground between stationary computers and UAVs: not as hard to work with and inconvenient as

UAVs, but mobile and with some additional hardware capabilities.

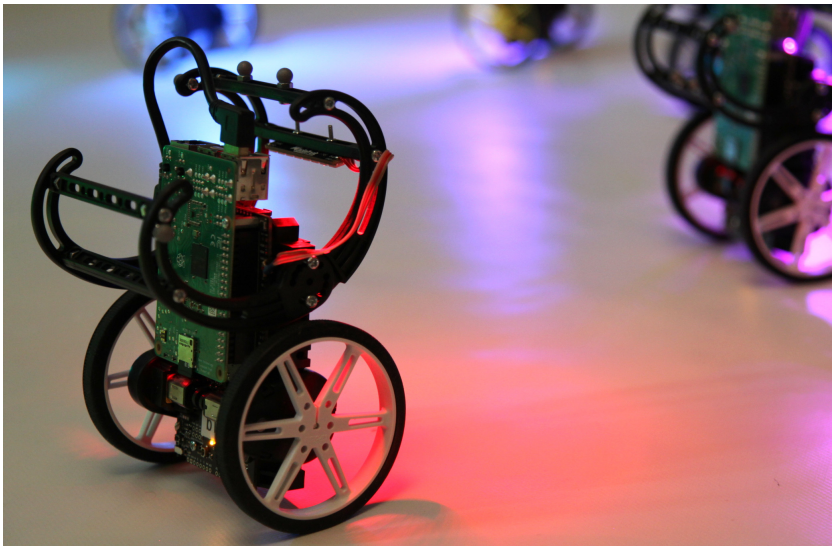


Figure 3.2: Photograph of Pololu Balboa robot taking part in the demonstrator of the Sync and Swarm article [6].

Hence, after conducting a market research we decided to use a Raspberry Pi on the *Pololu Balboa 32U4* robot. It is a self-balancing robot, so it is able to visualize synchronous behavior by swinging in unison with other agents. In fact, the first project realized using this platform involved a group of synchronizing robots demonstrating firefly synchronization algorithm [69].

The robot has on board an additional microprocessor (ATmega32U4), called by us *low-level controller*, which is responsible for driving and balancing. Such a setup is very similar to the one used on UAVs, where a low-level controller takes care of flying and a companion computer is responsible for deciding what to do. On Balboa robots the low-level controller can be easily programmed using Arduino.

Furthermore, the robots are equipped with an inertial measurement unit (IMU) and encoders measuring precisely the rotation of the wheels. This enables them to be used for projects requiring state estimation. Finally, each pin of the low-level controller is easily accessible thanks to connectors present in multiple places on the main board. This makes it very easy to enhance and extend the platform.

In order for the robot to fulfill our requirements we needed to perform a couple of modifications. First, we added a short USB cable to connect Raspberry Pi to the robot. It allowed us to communicate between low-level controller and Raspberry Pi and to reprogram the low-level controller remotely. Additionally, we have added an LED strip to visualize colors and a couple of reflective markers, which are used by motion capture systems for precise localization indoors.

Multiple robots were also equipped with *Raspberry Pi Camera Module V2* in order to perform image streaming for the information distribution optimization paper [3].

The robots were used in demonstrators of our research [3, 4, 6, 70] and were shown in multiple videos [71, 72]. A photo showing the robots taking part in one of the demonstrators is presented in figure 3.2.

TwinFOLD SCIENCE quadcopter

Figure 3.3: Photograph of twinFOLD SCIENCE drone.

The idea of the KPK project involved UAVs flying around and reconstructing the environment. Unfortunately, the same year as the project started the platform that was used by the university was discontinued. Hence, we needed to find a new UAV platform for the project. After considering multiple possibilities we decided to use a drone developed by *twins GmbH*[1] called *twinFOLD SCIENCE* UAV. The picture of this drone is presented in figure 3.3.

TwinFOLD SCIENCE is a quadcopter designed with the needs of researchers as a priority. Relatively small and easy to control, with plenty of space and mounting points for additional equipment. At its core there is a Pixhawk[2] flight controller, which is an open hardware standard for UAV design. It makes it possible to run a variety of software compatible with this standard, such as *PX4*[3] or *Ardupilot*[4] flight stacks, *QGroundControl*[5] ground control station and middle layers, e.g., *MAVROS*[6] for ROS interaction.

The detailed specification of the drone is available on *Dronehub K*[7].

[1] https://start.twins.co.at/

[2] https://shop.holybro.com/pixhawk-4_p1089.html

[3] https://px4.io/

[4] https://ardupilot.org/

[5] http://qgroundcontrol.com/

[6] http://wiki.ros.org/mavros

[7] https://medium.com/dronehub/introducing-the-twinscience-v1-7b56b0c592db

To highlight some of the more interesting features: with the diameter of around 52 cm it weights 0.8 kg and can carry a maximum payload of 1.2 kg. With the maximum payload and the 3 Ah version of the battery it flies for 10 min.

Furthermore, Raspberry Pi is a standard companion board included and tested with the platform, which makes it much easier for us to transfer experiments designed and tested on the previously described platforms to the UAVs.

Five twinFOLD SCIENCE drones were used in the final demonstrator of the KPK project. The demonstrator is described later in this thesis, in chapter 7. A video presenting it is available on-line[8].

[8] https://youtu.be/fw6bQhUfIsk

3.2 *Communication*

Communication technology has a major influence on the design and final performance of the MRS. This technology impacts many parameters of the system: available data rates, communication range, robustness, cost, etc. For instance, these parameters are then taken into account for mission planning, often being considered as a major component of the mission performance [73–75].

Hence, communication should be considered at an early design stage and advantages and disadvantages of each option should be thoroughly analyzed.

At the early stage of the KPK project, we have made multiple assumptions that influenced our technology choice:

1. We decided to pursue a fully distributed system without a single point of failure and with robots being able to join and leave the MRS freely. Distributed systems are usually more robust: they can handle failures of single agents or poor communication conditions.

2. Because we wanted to operate a larger group of robots (even up to 20) and had a limited budget, a communication device using licensed radio frequencies was not considered.

3. We introduced an assumption that robots are going to work in relative proximity (less than 1 km apart). Many practical applications can be realized with that assumption and it simplifies the choice of communication technology.

4. Our approach should be as generic as possible and easily applicable on other robotic platforms to make our research easy to reproduce.

These assumptions led us to the choices described in the following subsections.

Ad-hoc 802.11 WiFi

The `IEEE 802.11` standard is a widely used and available solution for wireless communication. It is also gaining popularity in robotic settings [76–78]. We were pursuing a fully distributed system, hence we decided to go with an ad-hoc network. Even though we run into many practical problems with support for drivers for wireless cards in Linux, eventually we have managed to configure our devices and run them in `802.11b/g` ad-hoc mode.

This technology has some drawbacks. For us, the greatest concern was related to ever-increasing amount of communication taking place in the unlicensed spectrum. The interference from other transmissions introduced the risk of reduced communication capabilities for our drones. Nevertheless, we have decided to choose it and try to mitigate the risk by having an option to switch to the 5 GHz band.

Multi-hop connections

One question that popped up frequently when we were describing our experimental setup during various meetings and conferences was about multi-hop communication. Even though in our demonstrators and applications we were describing, robots were always operating close to each other and were able to communicate directly, this question seemed to catch the interest of many and therefore needed to be addressed.

By default, a WiFi ad-hoc network supports only direct connections between peers. Even if multiple devices are connected to the same network, the communication is only possible between the pairs that are in communication range of each other. For instance, let us assume that devices A and C are not in range of each other, but the device B is in range of both of them. In standard ad-hoc mode, devices A and C cannot communicate, but if device B repeats everything what it receives, a link between A and C can be achieved.

Even though, normally, we are not describing solutions we did not decide to use, in this section we make an exception. The reason is that perhaps the most obvious choice for multi-hop communication is the use of the `IEEE 802.11s` mesh networking protocol, but we decided not to use it. It is widely supported in Linux and can be easily turned on just by a couple of commands. Furthermore, it builds on top of the other protocols carrying actual traffic (e.g., `802.11b/g/n/ac`), so it does not enforce a particular choice of the communication standard. The main reason for the decision not to use it was that visualization and debugging of such a solution is hard. We could not find any software that visualizes packet paths and allows to easily diagnose potential problems.

Instead, we decided to use the Babel[9] routing protocol and perform routing on the application layer. Such a solution is also very easy to configure and use on a Linux-based system, and because it operates on the application layer it is much easier to monitor and debug. An example of such a monitoring tool is presented in figure 3.4.

[9] https://www.irif.fr/~jch/software/babel/

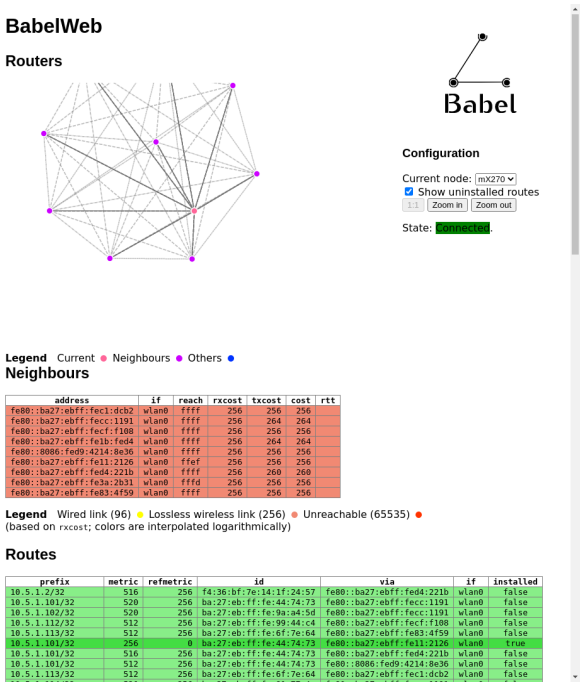


Figure 3.4: A screenshot from the Babelweb 2 software visualizing the routing table obtained by the Babel routing protocol during an operation of Balboa robots. It is included here to depict how this tool looks like.

Robot Operating System 2 (ROS 2)

The growing interest in robotics motivated a need for reusable software for robotic platforms. Hence, many robotic middlewares were created (e.g., Rock[10] in 2009 or the OROCOS[11] project which started in 2001). But perhaps the most well known and widely adopted middleware is the Robot Operating System (ROS)[12][79]. It aims at providing a unified ecosystem for robotic applications to work with, making them easy to implement and reusable. An outline of the motivation behind ROS is presented in figure 3.5.

[10] http://rock-robotics.org
[11] https://www.orocos.org
[12] https://www.ros.org

At the core of ROS is a communication method based on a publish-subscribe pattern. A user can define topics to which applications may subscribe and publish messages. Whenever a message is published on a topic, all applications subscribed to this topic receive the message. Unfortunately, customization of this communication scheme in ROS was very limited. Even though there were efforts to incorporate UDP communication (e.g., UDPROS), they never went outside of the development phase. Hence, in practice only reliable unicast

Figure 3.5: A symbolic representation of what ROS is. Image was used without modifications from the official ROS website (https://www.ros.org/about-ros/), licensed under CC-BY 3.0 (https://creativecommons.org/licenses/by/3.0/us/).

communication is possible.

The situation changed drastically with the introduction of ROS 2. Contrary to what the name might suggest it is not just a second version of ROS, but a complete redesign of the system from the ground up. In fact, for a couple of years new releases of both ROS 2 and ROS were published. The first experiments in the KPK project were conducted in early 2018 using ROS 2 Ardent Apalone. ROS 2 was back then considered a very immature project, in fact Ardent Apalone was a first version after the beta phase. However, the communication capabilities enabled by the redesign made it worth risking the switch to the new version.

ROS 2 is based on the Data Distribution Service (DDS) [80] standard for machine-to-machine data exchange. The standard aims at providing a unified way of implementation of the publish-subscribe pattern and is widely used in many fields, like, robotics, aerospace, autonomous vehicles, etc. Furthermore, ROS 2 allows to use any of multiple supported implementations of the DDS standard. This was a great setup for our research, as we were able to test multiple different implementations using the same code.

Furthermore, ROS 2 enabled some possibilities that were not there in ROS: multicast support, fully decentralized operation, interprocess communication. These features allowed us to realize our demonstrators and present our research in practice, in a widely used robotic framework.

3.3 *Deployment and management*

Maintaining a fleet of mobile robots operational is a hard problem which lacks well-established solutions and good software ecosystems. This is not surprising — after all, even ecosystems for individual robots (like ROS 2) are relatively new. Hence, creating a new MRS forces developers to solve many basic problems:

- How to set up robots in a scalable way, making it easy to add new ones in the future?

- How to get the working version of software to the robots?
- How to start a program on a specified subset of robots?
- How to change parameters of the program running on robots?
- How to get a work-in-progress version of software to the robots and debug it?
- How to manage updates and requirements?

Unfortunately, the lack of established solutions means that programmers often resolve to ad-hoc methods that solve the problem short-term, but might amplify it long-term. Such methods include manual code uploads, manual starting of programs on each robot individually, manual system updates etc. For instance, manually modifying code on the robot might result in different versions of software across the swarm with the inconsistencies being hard to track. This leads to cumbersome and time-consuming debugging. Even worse, any mechanical failure incurred by the robot might result in all the changes to the code being lost.

While working on the *Robots that Sync and Swarm* paper [6] we have separated MRS maintanance problems into two groups: software deployment and mission management. The former group includes all things that have to be done prior to starting a mission on robots: setting them up, managing requirements, uploading code, etc. The latter group manages robots during the mission: synchronized starting and stopping of programs, changing parameters, etc. For each of these groups we have came up with a solution described in the following two sections.

Software deployment for MRS

In order to keep our robotic platform up-to-date, easy to configure and easy to scale, we have used Ansible [81]. Ansible is used by system administrators to perform similar tasks with remote servers over the Internet. It is an open-source software maintained by the Ansible Community with commercial support provided by Red Hat. The first release was created in 2012 and currently it is a fairly known and well established method for server maintenance. Based on it we have developed a generic framework enabling us to remotely install, start and stop a program on a selected subset of robots simultaneously.

The solution was used for the first time in the *Robots that Sync and Swarm* paper [6] and published in the GitLab repository [82]. The repository also contains detailed description of usage and capabilities of this software. An example of the program execution is presented in figure 3.6.

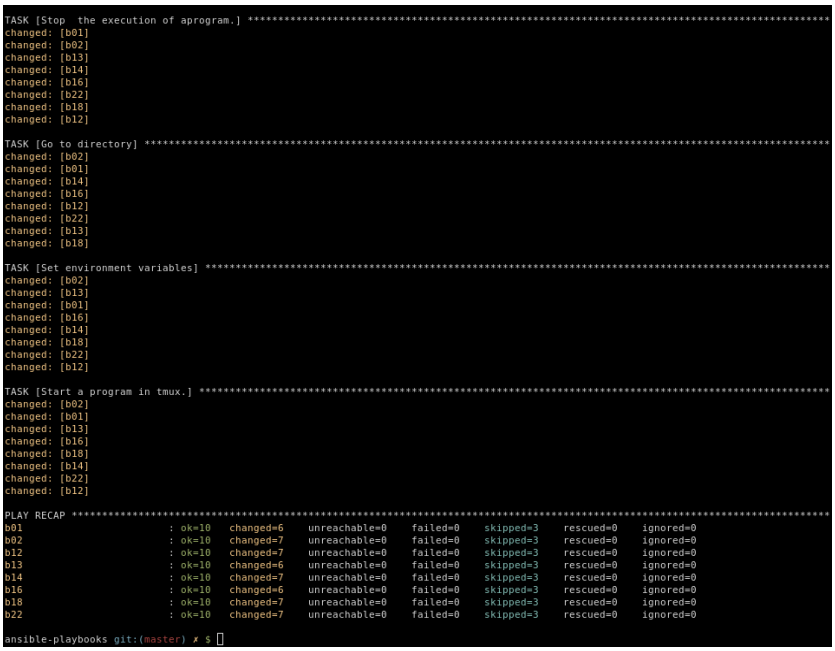


Figure 3.6: A screenshot presenting a part of output generated by a single run of the Ansible script used to start a mission on eight robots.

Throughout the duration of the KPK project we have successfully used this software to facilitate the work with robots and ease the configuration of their operating systems. Having such software incorporated in the workflow of the whole group was also helpful while working remotely: each member of the group knew that their configuration is exactly the same, which made debugging and helping each other easier. Additionally, one person could change the settings and let others apply them on their hardware, which made teamwork smoother.

Mission management

While working on one of our first presentations we have developed the main program that started the execution of the mission (i.e., started sending messages to other robots, making them drive around, etc.) immediately after being run. Soon we have realized that such design choice was a source of many unexpected problems.

First of all, the fact that robots start sending messages just after the program starts meant that some robots are still trying to connect to the network, while others already start transmitting data. The increased amount of transmissions during the starting process results in packet collisions and hence decreased performance. It significantly increased the connection time for ROS 2. This situation could be partially circumvented by starting robots sequentially, but

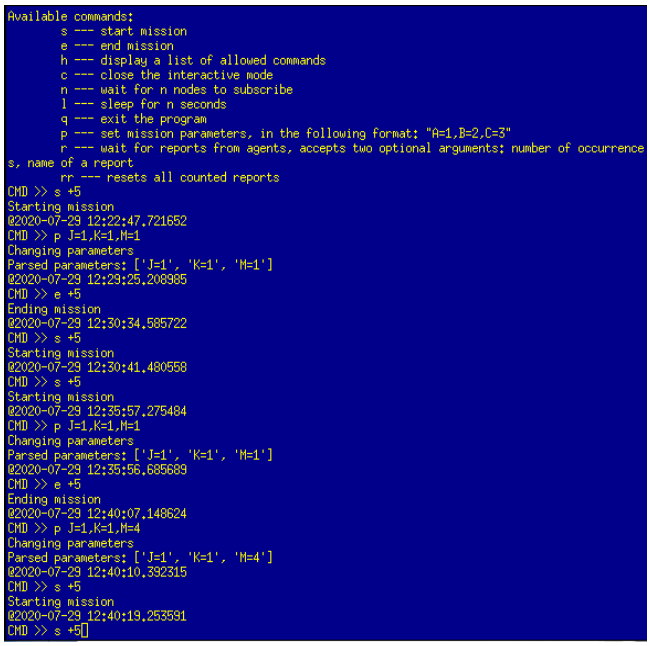


Figure 3.7: A screenshot of Mission Manager in action. A different color and font helps to easily distinguish it from other open terminals.

such a procedure would take even longer and does not fully solve the problem.

Furthermore, the time it takes a program to start was not the same on each robot. This fact added another source of randomness to our experiments making them harder to reproduce.

Hence, we have decided to implement a ROS 2 module named *Mission Manager* to facilitate synchronized starting and ending of the mission. The module defines multiple ways to execute a command with a time argument, e.g., *start a mission in 5s* or *finish a mission at 14:00*. It then translates the time into an absolute timestamp and communicates this timestamp together with a command to all robots. The client-side part of the module runs on robots, listens to the commands and executes them precisely at the given timestamp.

After successfully using the module in multiple experiments it was also enhanced with two additional functions. The first one allowed us to specify a set of parameters and apply them to robots on the fly, without re-starting the program. The second one enabled status reporting for robots. For instance, when a robot finished processing a task it reported mission completion and as soon as all robots did it, the program knew that the whole mission was finished and a new one could be started. Both of these functions helped us running robust and reproducible experiments.

A screenshot showing how the program could be used is presented in figure 3.7. It depicts a run when multiple times a mission was started with a delay of 5 s (*s* command), parameters were

changed (*p* command) and a command *e* was used to end a mission with a delay.

The program was incorporated in the majority of experiments conducted by our group. The most prominent example is the experimental study performed for the article published in the Sensors journal [2], where 29884 experiments were performed, each of them lasting for 100 s. In total that is over 840 hours! It was only possible thanks to automations built on top of the Mission Manager. Another good example is a proof of concept in the *Robots that Sync and Swarm* paper [6], where the ability to change model parameters on the fly was extensively used.

The source code of Mission Manager is available in the GitHub repository [13].

[13] https://github.com/zeroos/mission_manager

Final thoughts on MRS management

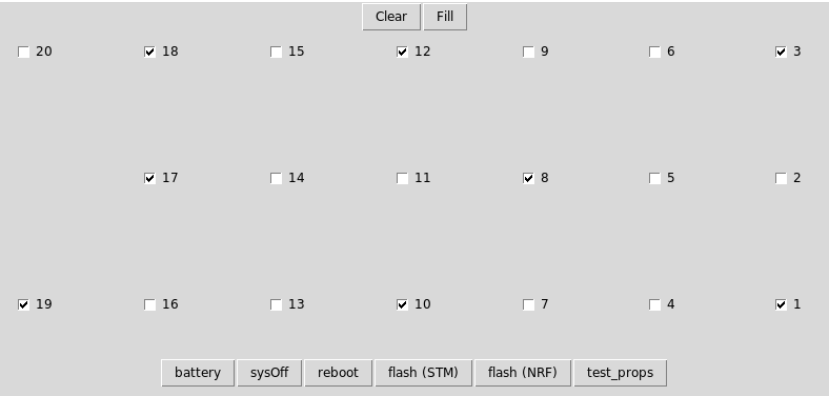


Figure 3.8: *Chooser* program from the *Crazyswarm* project [83].

In hindsight, it was definitely worth to invest a couple of weeks of work to develop additional software meant to facilitate our work with robots. It not only saved us time, but also helped to avoid bugs and reduce frustration of having to perform a multitude of repetitive tasks.

However, the problems we have had are not unique to our setup. Other designers of MRS also face them. For instance, figure 3.8 presents a program used to select which drones are to be used in the *Crazyswarm* project [83]. We provided this example, because the Crazyswarm is a popular implementation of a research-friendly multi-robot system platform and visualizes well the nature of the problem described in this section. In our setup, the choice of drones was done using Ansible using command line parameters.

As practical applications of MRS are becoming more and more popular, a software ecosystem designed to facilitate MRS mainte-

nance will surely appear and present a standardized way to solve such common problems.

4 Evaluation of information distribution

As a first building block of a generic communication solution described in this thesis, this chapter presents an analytical tool for evaluating information distribution. It allows comparing different communication schemes (or schedules), taking into account the amount of useful information distributed in the system. This model can serve as a tool to compare different systems that realize communication in UAV fleets or to provide optimization criteria for such systems. In particular, in chapter 5 we show a way to use this evaluation method in order to efficiently choose which information is worth communicating.

Contents

4.1 Concept

The presented solution is directly motivated by the KPK project (section 1.1), which aimed at multiple UAVs collaboratively performing a 3D reconstruction of an unknown environment. Dynamic communication conditions we faced while working on the project require continuous monitoring of the network status and evaluation of information distribution. Additionally, an agile fleet of UAVs presents an interesting challenge in terms of variable available throughput: sometimes the UAVs are able to exchange large amounts of data effectively, e.g., sending high-resolution images without problems, but at other times they might even struggle to exchange small messages, e.g., containing their positions. Therefore, it is not enough to handle just the message types that consume the most of the communication resources; when the communication quality is bad, even the smallest message brings value. This fact supports the need of a unified system that handles messages of all types comprehensively. The presented model addresses these needs. It is worth noting that the motivation and all examples provided in this article are related to UAVs, but the model may also be useful for other types of multi-robot systems.

The evaluation model introduced in this chapter is based on the following observations:

1. A message generated by a sender is meant to provide some information to the receiver, which benefits from this transmission. Hence, our model evaluates the communication from the receiver's perspective.

2. The usefulness of a message changes (increases or decreases) over time. For example, a map patch received at the beginning of a mission may be useful until the end, depending on whether and when the receiving UAV flies over the map region (useful) or moves elsewhere (not useful). Therefore, a message could add value throughout the whole mission and not only at the time of reception. This usefulness is, however, not known at the time of reception, it depends on the mission progress.

3. The information types can differ significantly and therefore are useful in different ways. For example, knowing the mission objective allows the UAV to plan its next task, whereas getting a new map fragment can help with path planning or localization. Thus, to express the characteristics of each information type it needs to be considered separately.

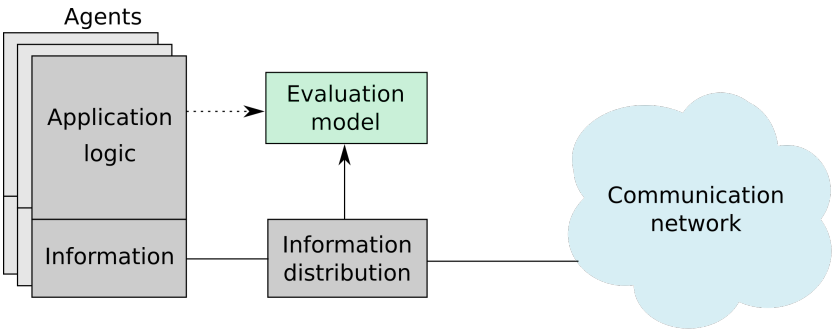


Figure 4.1: The proposed model aims at abstracting the mission from communication in order to evaluate information distribution.

The model serves as an abstraction between the mission logic and the communication mechanism. This concept is visualized in figure 4.1. The model is presented as a block working on top of the information distribution, that separates communication from the information that the agents act upon and exchange. Although the construction of a specific instance of the model is influenced by the application logic, the evaluation does not need to interact with the application during execution, and the only requirement from the communication side is a message-based protocol. It was designed specifically taking into account the capabilities and constraints of the hardware setup described in section 3.1.

Novelty

This approach enables a novel way of thinking about information distribution: the utilities provided by the evaluation model could serve as an input to an external algorithm for deciding which messages are more useful (e.g., when optimizing information distribution) or what communication solution performs better (e.g., for comparison purposes). Such an algorithm does not require any additional knowledge about the specific mission or information types in use. It only works on a stream of messages with defined message utility functions. Benefiting from this abstraction, such an external algorithm can be independent of the application and the mission, but at the same time the utilities reflect the value of each message for the mission.

The model only requires knowledge about the received information, which makes it easily applicable in real robotic systems. To that end, the model defines and uses *utility* as a function over time throughout the whole mission, which makes it easier to reason about the usefulness of a given information. For instance, let us analyze a message containing a map fragment that was received by a UAV. It is straightforward to define a function specifying if the UAV was making use (generating utility) of that map fragment at a given point in time, knowing the state of the UAV.

This chapter contains content which was initially presented at the Workshop on Mission-Oriented Wireless Sensor, UAV and Robot Networking 2019 (MiSARN), organized in conjunction with the IEEE International Conference on Computer Communications 2019 (INFOCOM) and published in the proceedings [1]. During the workshop, our work was presented as a first step towards the optimization of information distribution. We have realized that our approach is useful also for other aspects than just optimization of information distribution and can also be used as a separate component. Therefore, we have worked on an extension focused on emphasizing and improving the applicability of the model for other works. The extension was published in MDPI Sensors [2].

The following part of this chapter is organized as follows. In section 4.2 we present the definition of a utility-based model for the evaluation of information distribution schemes. Section 4.3 contains descriptions of multiple mission-related communication scenarios together with instances of the evaluation model for these scenarios (presenting the key properties of the model). Next, section 4.4 presents a theoretical analysis focused on finding out what kinds of scenarios can be described by the model. Finally, section 4.5 presents a proof of concept showing that the model can be implemented and applied on top of a well-established robotic framework, ROS2.

4.2 Model

In this section we present a common framework for defining information distribution evaluation models. The framework can be used to specify a model for a mission at hand. We call such a mission-specific definition of the model a *model instance*. Section 4.3 presents a variety of simple examples of how to define such instances.

The model's goal is to compare different communication schemes for a given course of a mission (also called "mission history" in the following) a posteriori. Since the communication is evaluated after the mission is finished, the model treats the mission history as fixed and assigns scores to sets of messages that could have been exchanged. Such an approach implies that we treat the mission history as being independent of the information being communicated and therefore the model is not able to express long-term influence on the mission goal of not sending or delaying a message. Despite this simplification, given a sound definition of the model instance, the approach is capable of assessing message usefulness (e.g., identifying the most and least important messages) limited to the extent analyzed in section 4.4, at the same time abstracting communication from the logic of the system.

Setup

An agent is a process taking part in the information distribution. For instance, an agent could be a communication process of: a UAV, an autonomous rover, a base station operated by humans, or a display visualizing a mission progress.

We evaluate the information distribution during a mission post factum, thus for each agent $g \in G$ we introduce a *history* of a mission H_g as a sequence of states of agent g during the mission. H_g is a function assigning a state of agent g to each time t during the mission execution; formally, $H_g : [0, t_{\text{end}}] \to S$, where S is the set of all possible states and the mission starts at time $t = 0$ and ends at $t = t_{\text{end}}$. A state is a tuple of properties of an agent. We do not define explicitly the structure of the state, as it may vary for each agent and for each mission, but intuitively it represents the knowledge of an agent. Some examples of properties potentially present in the state include: battery level, position, observed map area, etc. In order to successfully exchange a message, the communication protocol might also utilize additional meta data, headers, etc., but the introduced model is focused only on the part of the message that is useful for the application logic.

In order to compare communication schemes, we calculate a nu-

merical value (called *total utility*) $\mathcal{U}$ of a set of exchanged messages for a given mission history. Then we compare these values; the higher the utility, the more useful is the received information.

Let $\mathcal{M}$ be the set of all messages received by agents and $\mathcal{P}$ be the set of all information types. Each message $m \in \mathcal{M}$ has at least three properties: $m.\text{type} \in \mathcal{P}$ is the information type; $m.t_{\text{gen}} \in [0, t_{\text{end}}]$ is the time when the message was generated; and $m.t_{\text{rcv}} \in [0, t_{\text{end}}]$ is the reception time. Each message carries some data that can bring different information types. We use the term *generated time* instead of *sent time* in order to emphasize the fact that the actual time when the communication interface has sent the message is not important for the analysis. As a consequence, any delays in sending a message (e.g., caused by processing, interference or an ongoing transmission) should influence the final result of the evaluation.

Overview

The total utility of the system is the sum of all agents' utilities. Furthermore, the utility of each agent is the sum of utilities of messages containing information of all types received by that agent.

At the core of the model lies the *message utility function*. For each message, it describes how the utility of that message changes as the mission progresses over time. For example, we analyze the utility of the message containing the information about a map fragment for an autonomous UAV. Whenever the UAV flies above the area corresponding to this map fragment, the information is useful and the utility rises. However, as soon as it goes outside of that area, the message is not useful and thus brings no utility at that point in time. When the UAV comes back to this area, the utility of this message rises again. This example presents the continuous nature of the utility functions and their time-dependency. This function is heavily dependent on the mission. Even the same information type could have a different message utility function for different missions.

The utilities of messages containing the same information type are often related to each other. For instance, a UAV could have received a map fragment of the same area twice, in two separate messages. Then, whenever it flies above this area, not both of these messages are equally useful and they should not bring the same utility: the information is redundant and hence only one of the messages should not increase the total utility. To handle such situations, a way to aggregate utilities of information of a given type is needed. Therefore, for each information type we define a utility *aggregation function*, which quantifies how useful the transmission of information of a given type was throughout the whole mission. For each time t, it

takes the utilities of all messages containing the given information type and outputs the aggregated utility for time t. The total utility of the given information type is the integral of the utility aggregation function over time.

Definitions

The total utility of the system is given by

$$\mathcal{U}(\mathcal{M}) = \sum_{g \in G} \hat{\mathcal{U}}(\mathcal{M}_g, H_g), \tag{4.1}$$

where $\mathcal{M}_g$ is the set of all messages received by agent g. $\hat{\mathcal{U}}$ is the utility function for a single agent and is defined as

$$\hat{\mathcal{U}}(M, h) = \sum_{p \in \mathcal{P}} \omega_p \cdot \int_0^{t_{\text{end}}} \mathcal{A}_p(M, t, h(t))\, \mathrm{d}t. \tag{4.2}$$

where ω_p is an importance factor of information type p. The arguments M and h represent the set of messages, and the history of an agent respectively. Note that in equation (4.1) this function is used with the arguments $M = \mathcal{M}_g$ and $h = H_g$.

Before defining the aggregation functions we introduce the following notations: $M^p_{\text{rcv}}(t)$ is a subset of set M. It includes all messages containing information type p received up to time t:

$$M^p_{\text{rcv}}(t) = \{m \in M : m.t_{\text{rcv}} \leq t \wedge m.\text{type} = p\}. \tag{4.3}$$

It can be thought of as an operator over the set M that filters it based on the reception time and information type. Similarly, $M^p_{\text{gen}}(t)$ is a subset of set M, including all the messages containing information type p generated up to time t.

All aggregation functions require three arguments: M, t and $s = h(t)$. M is the set of messages to be considered, t is the time when aggregation takes place and s is the state of an agent at time t. In this work we introduce three types of aggregation functions: *sum*, *max*, and *last*.

The *sum* aggregation function treats all received messages independently by summing their utilities:

$$\mathcal{A}^{\text{sum}}_p(M, t, s) = \sum_{m \in M^p_{\text{rcv}}(t)} \mathcal{U}_p(m, t, s). \tag{4.4}$$

The *max* aggregation function at time t takes only the utility of a message with a maximum value:

$$\mathcal{A}^{\text{max}}_p(M, t, s) = \max_{m \in M^p_{\text{rcv}}(t)} \mathcal{U}_p(m, t, s). \tag{4.5}$$

The *last* aggregation function at time t evaluates only the utility for the most recently generated message:

$$m_{\text{last}}(t) = \underset{m \in M^p_{\text{gen}}(t)}{\arg\max}\, m.t_{\text{gen}}, \tag{4.6}$$

$$\mathcal{A}^{\text{last}}_p(M, t, s) = \mathcal{U}_p\left(m_{\text{last}}(t), t, s\right). \tag{4.7}$$

The *last* aggregation function takes into account the last *generated* message. This allows us to describe situations, when generating a new message invalidates the information contained in a previous message, i.e., as soon as a new message is generated, the previous one does not bring any utility anymore. Unfortunately, it introduces a new challenge for the evaluation of the model from the receiver's perspective. If a message m is generated but not received, we cannot influence the aggregated utility based on the time it was generated. Thus, this kind of utility should only be used in conjunction with a reliable communication protocol.

The defined aggregation functions can be applied to any information type. For example, if we want to use the *sum* aggregation function for information type *test*, we will write: $\mathcal{A}_{\text{test}} = \mathcal{A}^{\text{sum}}_{\text{test}}$.

4.3 *Mission examples*

In this section, we specify the utility model for five different information types in order to provide examples of how the utilities of various information types can differ and how different aggregation functions can be used. The examples are motivated by various research efforts at our university on UAV applications [9] and networking [8]. The choice of examples is not incidental: each of the examples presents a significantly different approach to utility definition. These examples should provide intuition on how the model can be instantiated (i.e., defined for a specific mission) and could serve as a basis to define similar utilities for a variety of information types. We focus our analysis on simplified definitions in order to make them easier to understand and analyze. For certain use cases, where such simplifications are not applicable, more complex definitions could be introduced, e.g., taking into account additional properties of exchanged messages, correctness, or age of information (information decay).

For each information type, we present an example of the mission scenario and a plot of the message utility and aggregation functions. All of them follow the same convention (see example in figure 4.2): message utilities are plotted with dashed lines, each message is represented with a different color. The aggregation function is plotted with a solid line and the area under its plot is filled. Additionally,

each moment when a message was generated is marked with an arrow pointing up and the reception time is marked with an arrow pointing down.

Agents' properties

In this section we analyze two types of information, both of them representing robots' properties: battery level and position in 2D. The definitions of utility of messages containing information of these types, depend on the way the information provided by these messages is used. For example, consider using the position of another robot to estimate how far it is from a given point. We can assume this distance in the worst case will change linearly over time, based on the maximum speed of the robot. However, if we use the last received position in order to estimate where exactly the agent is now, the probability of estimating the exact position, based only on the maximum speed and the last received position decreases quadratically over time in the 2D case (the agent can go into any direction on a plane) and cubicly over time in the 3D case (it potentially travels in any direction in space).

This distinction in the interpretation of position data serves as a good illustration of one of the most important properties of the introduced model: the ability to abstract communication from mission logic. Once the utility functions are defined, the evaluation model allows us to treat all the messages in exactly the same way, independently of the way the message is used by the mission.

Battery level

We assume agents are exchanging messages containing their battery levels. The algorithm running on each agent directly uses the data from the most recent message and assumes it to be correct, even if it is outdated. From this message, it extracts the information about the sender's availability, i.e., if the sender's battery is depleted, it is no longer available.

In order to introduce the message utility function for this scenario, we define ρ as a pessimistic battery depletion rate, i.e., a rate that is higher than the maximum possible battery consumption rate. Hence, the battery level after time $\hat{t}$ drops in the worst case by

$$R_{\text{batt}}(\hat{t}) = \hat{t} \cdot \rho. \tag{4.8}$$

The utility of a message reflects the ratio between a pessimistic real battery level at a given time and the message data.

Here, the *max* aggregation function is used, because the algorithm running on an agent always uses the most useful message, which in

this case is always the most recent message received by the agent.

The message utility and aggregation functions for the battery information are defined as follows:

$$\mathcal{U}_{\text{batt}}(m,t,s) = \max\left(0, 1 - \frac{R_{\text{batt}}\left(t - m.t_{\text{gen}}\right)}{m.\text{batt\_level}}\right) \tag{4.9}$$

$$\mathcal{A}_{\text{batt}} = \mathcal{A}^{\max}_{\text{batt}} \tag{4.10}$$

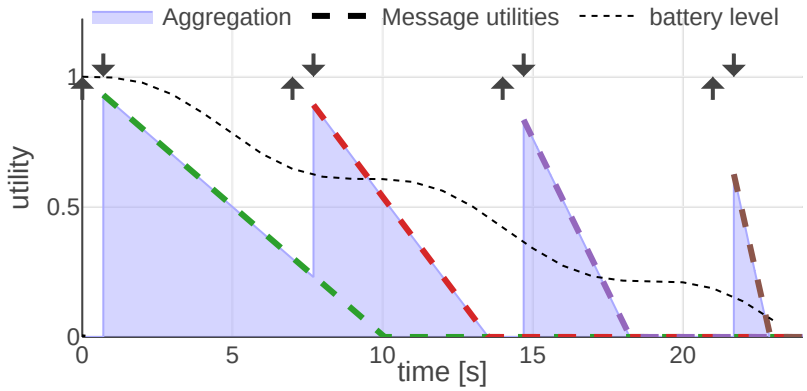


Figure 4.2: $\mathcal{A}_{\text{batt}}$ and $\mathcal{U}_{\text{batt}}$ functions plotted for an instance of the given mission.

By studying figure 4.2 presenting the plot of equations (4.9) and (4.10) for an example mission, we can observe some properties of these utilities. First of all, a message is useful for the receiver, starting from its reception time to the time when the battery would be completely depleted in the worst case (assuming pessimistic depletion rate). The utility decreases over time as soon as the message is generated. This is based on the intuition that we have the perfect information only at the time of measurement. The more time has passed, the less precise information we have about when the battery is going to deplete. Additionally, we can observe that the utility falls faster when the measured battery level is low.

Position

In this example, the agents exchange position information. This position information is handled in a similar way as the battery level information, with two small differences. First, instead of using a battery depletion rate, we define the area where the agent can be after time $\hat{t}$ as

$$R_{\text{pos}}(\hat{t}) = \pi v^2_{\max} \cdot \hat{t}^2. \tag{4.11}$$

Second, we define an area A, which could either be the maximum area of a mission or an area big enough that knowing the agent is somewhere in that area does not bring any valuable information. The utility of a message over time is defined as a ratio between the

complement of the area where the agent can be and A. Such a definition maintains the following property: if the exact position of an agent is known, the utility equals 1 and it drops to 0 when there is no information about the agent's position (it can be anywhere in A). Similarly to the battery level example, the position information also uses the *max* aggregation function:

$$\mathcal{U}_{\text{pos}}(m,t,s) = \max\left(0, 1 - \frac{R_{\text{pos}}(t - m.t_{\text{gen}})}{A}\right) \tag{4.12}$$

$$\mathcal{A}_{\text{pos}} = \mathcal{A}^{\max}_{\text{pos}} \tag{4.13}$$

Example utility and aggregation functions from equations (4.12) and (4.13) are plotted in figure 4.3. The result is very similar to the battery type, but the utility decreases quadratically over time. Additionally, the utility is independent of both the message content (last known position) and the state, so the shape of the message utility function is always the same.

Mission status commands

During a mission, it might be required to communicate a status of the mission to all agents involved in it. Examples of statuses include: *waiting for mission start*, *mission in progress*, *abandon mission*, *mission finished*. The main characteristic of this type of information is that, as soon as the new message is generated, the mission status changes and all the old messages are not valid anymore. We express that with a constant message utility function and the *last* aggregation function:

$$\mathcal{U}_{\text{status}}(m,t,s) = \begin{cases} 1 & \text{if } t > m.t_{\text{rcv}} \\ 0 & \text{otherwise} \end{cases}, \tag{4.14}$$

$$\mathcal{A}_{\text{status}} = \mathcal{A}^{\text{last}}_{\text{status}}. \tag{4.15}$$

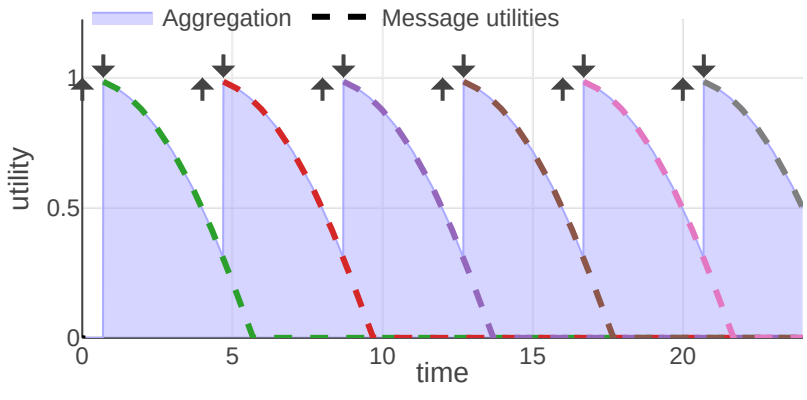


Figure 4.3: $\mathcal{A}_{\text{pos}}$ and $\mathcal{U}_{\text{pos}}$ functions plotted for a mission example.

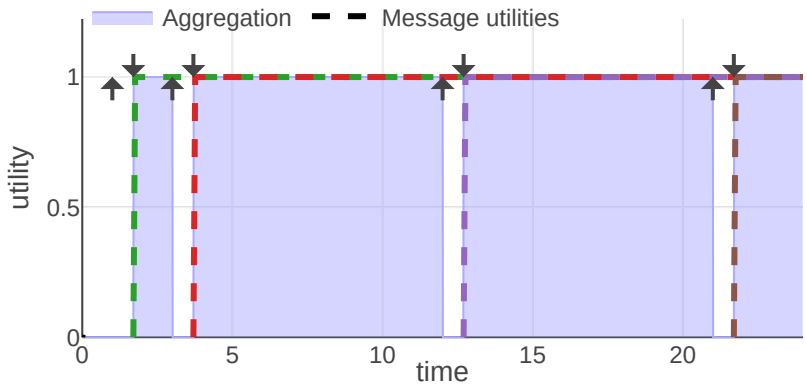


Figure 4.4: $\mathcal{A}_{\text{status}}$ and $\mathcal{U}_{\text{status}}$ functions plotted for a mission example.

Figure 4.4 presents a plot of the aggregation and utility functions for an example communication scenario. An agent aggregates the utility when it knows the current mission status. Each delayed message means that the agent does not know the current status; thus, the final utility is decreased. In the figure this shows as a lack of aggregated utility between the generation and reception of each message. Note that the utility of each message never drops, e.g., the utility of the green message is equal to 1 for almost the whole mission. It is the aggregation function that decides which message should be considered at which point in time and introduces the areas of no utility.

It can be easily observed that, for this information type, a communication scheme that minimizes transmission delays is preferred. The fact that the communication is evaluated from the receiver's perspective makes such a definition possible—the sender would not have access to information about delays without any additional protocol.

Because the *last* aggregation function is being used, the approach is only useful to evaluate the performance at the end of the mission assuming a reliable communication channel. Sometimes it might be beneficial to compute the utilities live, while the mission is in progress. Unfortunately, because of the receiver-centric nature of the model, in that case we cannot use the generation time of a new message, because that message might not have been delivered yet. Instead, we can change our modeling approach and define utility as a probability that the status of the mission did not change since the message we have received was generated.

We assume that changes of the status of a mission could be modeled as a Poisson point process, i.e., a process where events (in this case status changes) appear independently at a constant average rate λ. Then, a time between events is modeled using exponential

distribution. From a memorylessness property of that distribution we know that we can also use it to express a distribution of a time between an observation of a status (i.e., the time when a message is generated) and a change of that status. So the probability that an event occurred since the last observation is expressed using a cumulative distribution function (CDF) of exponential distribution: $1 - e^{-\lambda t}$. We are interested in an inverse of this event (i.e., that the status did *not* change), so we can define the utility as:

$$\mathcal{U}_{\text{status}'}(m, t, s) = e^{-\lambda(t - m.t_{\text{gen}})}, \tag{4.16}$$

$$\mathcal{A}_{\text{status}'} = \mathcal{A}^{\max}_{\text{status}'}. \tag{4.17}$$

We use the *max* aggregation function, because we are interested in the message that gives us the maximum probability that a status did not change since it was received. These functions are plotted in figure 4.5 for an example of a mission with $\lambda = 0.5$. We can observe that the utility falls down rapidly. Therefore, an information of this kind should be repeated periodically to increase the probability that it is still valid.

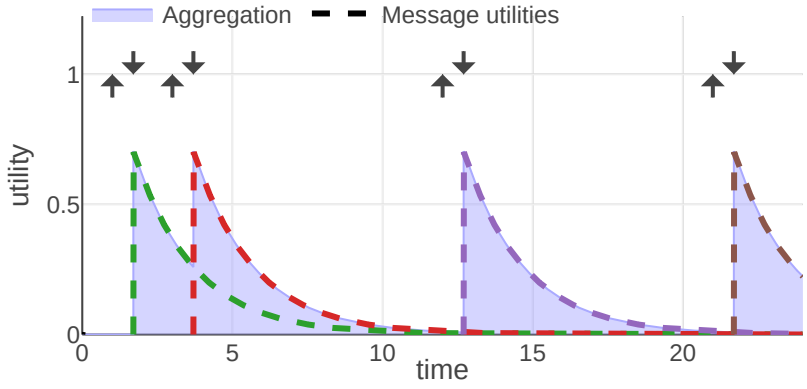


Figure 4.5: $\mathcal{A}_{\text{status}'}$ and $\mathcal{U}_{\text{status}'}$ functions plotted for a mission example.

Mission objectives

Mission objectives, although conceptually similar to mission status, are distinctively different from them. Examples of objectives include: *explore sector A*, *monitor sector B*, *map sector C*. They define goals to be pursued during the mission by agents.

We say that an agent is *working on* an objective if it actively tries to fulfill it. For instance, if it received a message *explore sector A*, immediately it does not receive any utility, but it knows that traveling through this sector is beneficial for the mission. We say that it starts to work on this objective as soon as it decides to go to this sector and explore it and this is the time when the utility of this message rises.

However, it does not necessarily mean that the agent has to explore the whole sector; it can contribute some work and then continue with a different objective.

This example demonstrates that the message utility is related to the agent's state (in this case with respect to the objectives that the agent is trying to fulfill) and that it varies over time. It is worth noting that the utility of the message is not necessarily decreasing over time; in this case, it will be equal to 0 at the time of reception and will rise as soon as the agent decides to work on the objective.

An agent cannot start the work on an objective before a message introducing this objective is received by it. It is, however, possible that an agent will work on multiple objectives at the same time. Therefore, we will treat the utilities associated with each objective independently. Hence, the *sum* aggregation function is used.

For the sake of simple presentation, we assume all objectives have equal importance and each unit of time spent working on the objective is equally useful. Thus, we define the utility of a message that contains objective o as 1 whenever the agent is actively working on objective o and 0 otherwise:

$$\mathcal{U}_{\text{objective}}(m,t,s) = \begin{cases} 1 & \text{if } m.o \in s.O_{\text{pursued}} \\ 0 & \text{otherwise} \end{cases}, \tag{4.18}$$

$$\mathcal{A}_{\text{objective}} = \mathcal{A}^{\text{sum}}_{\text{objective}}, \tag{4.19}$$

where $m.o$ is the objective introduced in message m and $s.O_{\text{pursued}}$ is a set of objectives on which the agent was actively working in state s.

The assumptions about equal importance and constant utility are not required. For some missions it might be more practical to assume the message utility is given by a decreasing function (e.g., if joining at the beginning makes an agent more useful). We could also assign higher utilities to the more important objectives.

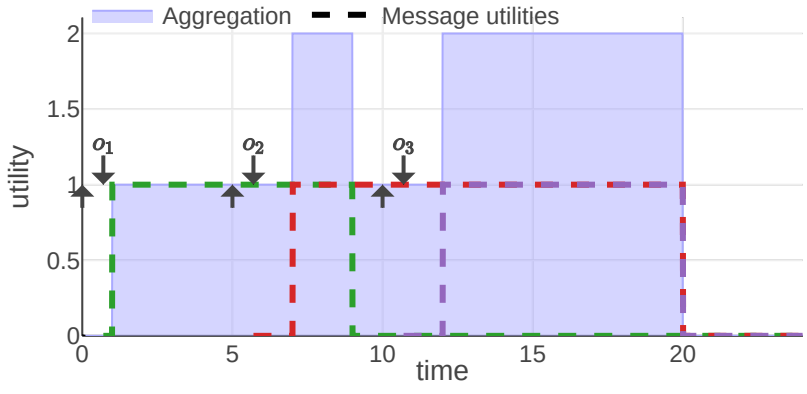


Figure 4.6: $\mathcal{A}_{\text{objective}}$ and $\mathcal{U}_{\text{objective}}$ functions plotted for a mission example.

Figure 4.6 presents a plot of aggregation and message utility functions for an example communication scenario. In this scenario, the agent was working on three objectives: o_1 between time 1 and 9, o_2 between time 7 and 20 and o_3 between time 12 and 20. During time periods $[7, 9]$ and $[12, 20]$ the agent is working on two objectives simultaneously. The aggregated utility at each given time is the number of objectives the agent is working on at that time.

If the message corresponding to a particular objective is not received, the agent does not know about it and cannot work on this objective. The utility will not be generated. Therefore, a communication scheme that prioritizes the objectives that are more important to be worked on by the agent will perform well in the evaluation framework.

Localization using a map

The next type of information analyzed in this work is a map used for localization. We assume each message m contains a sector of a map covering area $m.A_{\text{map}}$. Each agent maintains a union of all of the received sectors and stores the area of this union in its state as $s.A_{\text{map}}$. We will call this union a *gathered map* of an agent. At each point in time an agent observes the area $s.A_{\text{obs}}$ around it and tries to localize itself on the gathered map. The bigger the intersection between the observed area and the gathered map, the better the localization result.

We define the added area of message m as the difference between $m.A_{\text{map}}$ and all the areas previously received:

$$N_m = m.A_{\text{map}} \setminus \left(\bigcup_{\substack{m' \in M^{\text{loc}}_{\text{rcv}}(m.t_{\text{rcv}}) \\ m' \neq m}} m'.A_{\text{map}} \right). \tag{4.20}$$

The utility of a message m at time t equals the ratio of the common part of the observed area and N_m to the whole observed area. The proposed model is presented in equations (4.21) and (4.22).

$$\mathcal{U}_{\text{loc}}(m, t, s) = \frac{N_m \cap s.A_{\text{obs}}}{s.A_{\text{obs}}}, \tag{4.21}$$

$$\mathcal{A}_{\text{loc}} = \mathcal{A}^{\text{sum}}_{\text{loc}}. \tag{4.22}$$

An example of a mission introduced to analyze this information type is visualized in figure 4.7. The plot visualizes the 1D position of an agent over time with a dashed line. Its observed area is marked with a light gray color and the borders of this area are plotted with

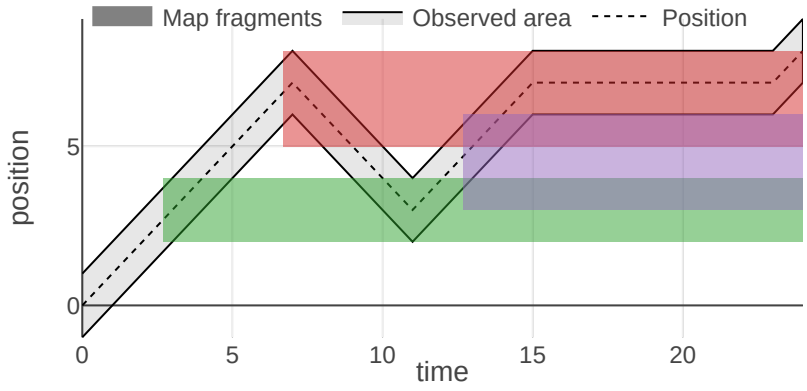


Figure 4.7: Plot of the agent position and the received map data in 1D.

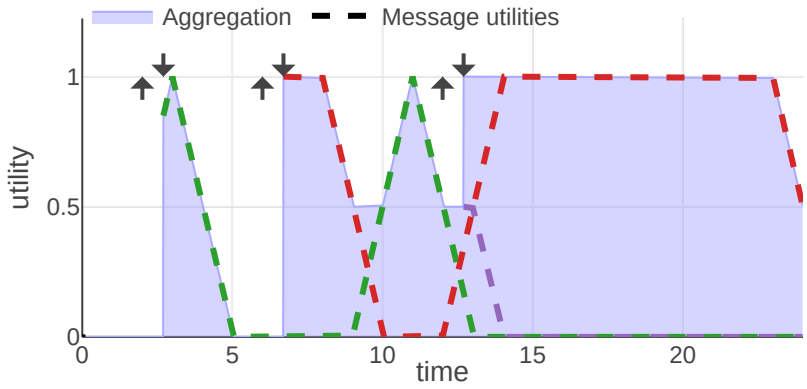


Figure 4.8: $\mathcal{A}_{\text{loc}}$ and $\mathcal{U}_{\text{loc}}$ functions plotted for a mission example.

a solid black line. During a mission, the agent received three map fragments, at times 2.7, 6.7 and 12.7. These messages are visualized with a filled rectangle of different colors representing the area of a 1D map associated with the message. The fragment is valid indefinitely, hence the area is marked since the reception time until the end of the mission. The color of each map fragment matches the color of a message utility function plotted in figure 4.8.

In figures 4.7 and 4.8 we can observe how message utilities can change throughout a mission. For instance, the first map fragment (drawn in green) is useful just after the reception, then at time 5 it brings no utility since the agent cannot observe this sector anymore. Around time 9 the agent comes back and the map fragment is useful again. These figures also present nicely the properties of the model, namely that the utility of information is state-related (in this case it depends on the agent's position and thus its observed area) and that utility of each message is independent of the utilities provided by other messages. The third map fragment (drawn in purple) overlaps with previously known areas, thus, bringing utility only for the newly introduced part. This demonstrates how the possible redun-

dancies of the information can be handled.

Image streaming

Another interesting kind of information that could be distributed between agents is a stream of images, for instance being captured by a camera of one of the agents. The analysis is motivated by a surveillance mission, where multiple agents (e.g., drones) are monitoring an area and taking pictures of some targets. They exchange these pictures in order to offload some computations and share the knowledge about targets. We focus on situations, where a ground personnel may connect to such drones in order to preview the exchanged image stream. We assume each message contains a single image. We call a stream of such messages viewed by humans a video.

Clearly, there is a wealth of techniques aiming at optimizing video streaming performance. The role of this model is to evaluate those approaches. The simplified example presented here focuses only on one aspect of Quality of Experience (in this case a perceived frame rate) but it could be used as a basis for more complete solutions.
In this article, the image streaming information type serves as an opportunity to demonstrate a different approach to defining the utility model instance. So far, message utility functions were defined in a structural fashion, i.e., we were arguing about the amount of information that each message brings at a time instance and constructing the functions accordingly. This time, we first specify some properties of the resulting total utility and based on that we come up with a form of message utility function.

We base our definition of the model on an assumption that the human perception of a frame rate of a video can be modeled by the Weber-Fechner law [84]. The law relates the subjective perception of human senses (p) to the objective stimulus intensity (f) and claims that this relationship is logarithmic:

$$p = \alpha \ln \frac{f}{f_0} \tag{4.23}$$

for some α and f_0, where f_0 is the smallest stimulus that can be perceived.

In our case, the observed subjective quality is the smoothness of the video and an objective measure is expressed in frames per second (FPS). The utilities provided by the model should be proportional to the perceived quality, hence in the definition of the model we aim at having the logarithmic relationship between utilities and FPS. Therefore the model should have the following property:

$$U(\mathcal{W}_f) = \hat{\alpha} \ln \frac{f}{f_0} \tag{4.24}$$

where $\mathcal{W}_f$ is a message set which results in a video being streamed at f FPS, formally:

$$\mathcal{W}_f = \{\, m \mid m.t_{\text{gen}} = \frac{n}{f} \wedge n \in \mathbb{N} \,\}. \tag{4.25}$$

Even though in the original formulation of Fechner's law (equation (4.23)) the natural logarithm is used, we can change the base of the logarithm by substituting $\hat{\alpha} = \frac{S}{\log B}$, where B is a new base of the logarithm and S is the parameter of the model. We have decided to use $B = 2$, which is often used in IT applications. The final form of the utility function is:

$$U(\mathcal{W}_f) = S \log_2 \frac{f}{f_0} \tag{4.26}$$

Next, we introduce a message utility function and we show that the obtained model fulfills the requirement defined in equation (4.26). For the sake of simple presentation, in this analysis we assume the absence of delays (i.e., the frame is received at the same time it is being generated). We take delays into account in the definition of message utility, so they will lead to a lower total utility value, but we do not examine their impact. Naturally, we use the *max* aggregation function since the last frame always brings the most useful information. Let us assume a message utility function has the following form:

$$\mathcal{U}_{\text{vid\_ideal}}(m, t, s) = -\log_2(t - m.t_{\text{gen}})k + l \tag{4.27}$$

By solving a system of equations (4.26) and (4.27) we can find out that the proposed model indeed results in a pursued relationship for $k = S$ and arbitrary l. Unfortunately, this definition is impractical: it might result in an infinitely large or negative utility value, which might mean that receiving some messages introduces loss or disproportionately huge gains.

Therefore, we have decided to limit the values of equation (4.27) by capping them to $[0, 1]$, while still using the *max* aggregation function. It results in the following model:

$$\mathcal{U}_{\text{vid}}(m, t, s) = \max\left(0, \min\left(1, -\log_2(t - m.t_{\text{gen}})k + l\right)\right), \tag{4.28}$$

$$\mathcal{A}_{\text{vid}} = \mathcal{A}_{\text{vid}}^{\max}. \tag{4.29}$$

The maximum value of total utility per second is limited to 1, by such an approach. We assign this value to an exchange that results in a video streamed with frame rate of $f_{\max}$ FPS. Any exchange that results in a higher frame rate should receive the same utility, formally:

$$\forall f : f > f_{\max}. \quad \mathcal{U}(\mathcal{W}_f) = 1. \tag{4.30}$$

If we had tried to pursue the same approach as we did with the $\mathcal{U}_{\text{vid\_ideal}}$ (equation (4.27)) utility function, i.e., tried to solve a system of equations (4.26), (4.28) and (4.30), there would have been no solutions. The model received in that case no longer results in the logarithmic relationship we were pursuing. However, if instead of looking for solutions for any f in equation (4.26) we pick just one frequency f_1, we get the following results:

$$l = 1 + \log_2 \left(\frac{1}{f_{\max}} \right) k \tag{4.31}$$

$$k = \frac{S}{1 - \frac{f_1}{2 f_{\max} \ln 2}} \tag{4.32}$$

In practice it means that the relationship is not logarithmic anymore. In the "ideal" case, each time the frequency was divided by B, the utility was lowered by S. In the new model this property holds only for a specific frequency f_1. However, in practice it should not be a problem, as usually the system will aim at streaming videos close to some predefined frame rate and we will be evaluating exchanges that result in frame rates similar to that one. Additionally, the relationship we have achieved is indeed very close to logarithmic.

We have calculated the values of parameters k and l for $S = 0.1$, $f_{\max} = 24$ Hz, and $f_1 = 12$ Hz. Then, we have simulated missions in which images were sent with different frame rates over 100 s. In practice, it means that we were evaluating the model for different $\mathcal{W}_f$. The utilities generated during these simulations are presented in figure 4.9. The blue line represents an ideal model while circles correspond to utilities achieved by simulating the modified version. We find the resulting model satisfactory, because it is motivated by an abstraction of human perception, which itself often does not strictly adhere to the logarithmic relationship.

Figure 4.10 presents a plot of the utility and aggregation functions from the introduced model. We can observe that exactly for $\frac{1}{f_1}$ the function is flat. If messages were to be exchanged with a higher frequency than f_1, the result will just be a constant function. After that time period, the function decreases logarithmically. The slope of that decrease can be adjusted with the parameter S.

In the following paragraphs, we examine the possibility to include source adaptation in the model. Most types of data can be adapted in order to reduce the amount of data sent while sacrificing the information content. A video stream is a good example, because image compression techniques can achieve a very good compression rate

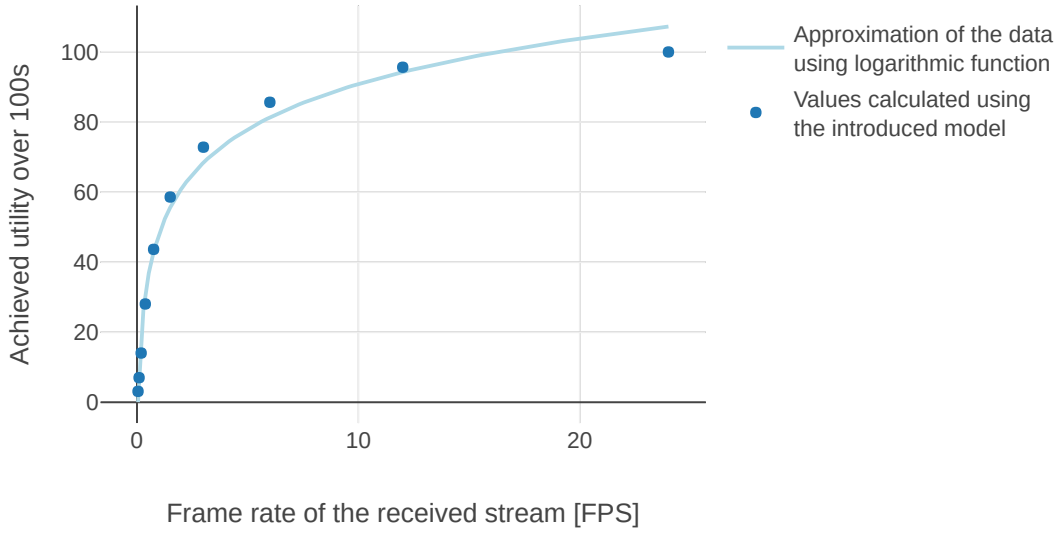


Figure 4.9: Total utility achieved by evaluating message exchanges which resulted in different constant frame rates using the introduced model.

while the stream still looks good for a viewer. Of course, an important part of video adaptation is reducing its frame rate. However, this part is already taken into account by the model, as it assigns lower utilities when the message (an image) is not received, therefore penalizing lower FPS streams. In the following, we will only examine how to incorporate into the model the information about a lower quality of a transferred message (e.g., a lower resolution).

We model adaptation in the following way. First of all, we define the notion of a *version of a message*. Each message could be available in multiple versions. They are all generated at the same time and they all represent the same knowledge, but the information content might differ, i.e., for any message, a sender can decide to send an adapted (e.g., lower quality) version of that message. We call "the *best version* of a message" a version that has the highest information content. For instance, in the case of a video stream it could mean that the same image can be sent in multiple resolutions. The best version of that message will be the image with highest resolution available for the sender.

The evaluation model should take the existence of those versions into account and assign lower utilities to versions of messages that contain less information. At any point in time the sender can de-

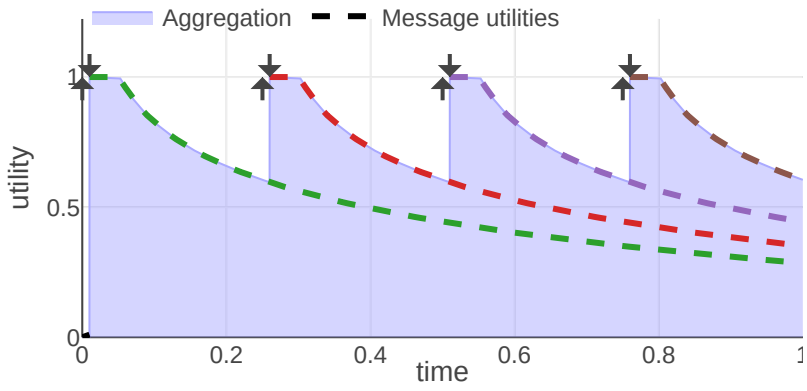


Figure 4.10: $\mathcal{A}_{\text{vid}}$ and $\mathcal{U}_{\text{vid}}$ functions plotted for a video streaming mission example.

cide to re-transmit the same message in a different version (e.g., in a higher resolution). As soon as a better version is received, the evaluation model should use it instead of the previous one.

In order to implement these features, we make two changes to the model introduced in section 4.2. First of all, the model did not include the notion of versions of messages. Hence, in the case of the *max* aggregation function, we redefine each message in the model as being the best version of that message at a given time (all other versions are not used). It means that the definition of $M^p_{\text{rcv}}(t)$ from equation (4.3) changes and should contain only the best versions of all messages available at time t. Second, we need to introduce a version of the message utility function that takes adaptation into account. Our example is based on video streaming messages, because it uses the image resolution, but a similar parameter of a message can be used for any other type. The adaptation is realized by changing the resolution of images. We assume the utility of the adapted image is proportional to its diagonal, expressed in number of pixels. The message utility function takes the following form:

$$\mathcal{U}_{\text{vid}'}(m,t,s) = \frac{m.d}{d} \cdot \mathcal{U}_{\text{vid}}(m,t,s), \tag{4.33}$$

where $m.d$ is a diagonal of the image provided in message m and d is a diagonal of a frame in a maximum available resolution on the sender's side.

4.4 *Theoretical analysis of the model*

In this section, we analyze the universality of the model by addressing the question of what kinds of communication history can be assessed and compared by the model. It aims at increasing the understanding of its limitations. Obviously, it depends on the definition

of the instance of the model (message utilities for given information types), but we focus only on the general formulation of the model (equations (4.1) and (4.2)) and analyze what constraints are introduced by this definition.

We start by showing how to define an instance of the model that mimics the behavior of other evaluation models that just assign a single numerical value to each message. Then, we analyze a more complicated and general case of message filtering. The first subsection presents a generalized use case that is much easier to express than the mission examples introduced in section 4.3. On the other hand, the second subsection argues about use cases that are harder (or even impossible) to express than the presented mission examples.

External message utility value

Sometimes the message utility is already given as a numerical value, for instance, by another evaluation algorithm that is specialized for a given information type. Each message then gets a single value assigned to it and the values of all exchanged messages are summed up. This is a very simple scheme that might often be useful, but some situations cannot be captured by it; for example, it cannot capture applications in which the utility of one message depends on the successful transmission of another message. It might still be useful to incorporate such a simple scheme into our model, e.g., in order to have one unified model for all information types or to use an optimization technique that is based on this model. Let us assume there is a function V that assigns scores to messages and we would like to define a model instance for information type p that preserves the scores of V. This could be achieved using the following message utility function:

$$\mathcal{U}_p(m, t, s) = V(m) \cdot \delta\left(t - m.t_{\text{rcv}}\right) \tag{4.34}$$

where δ is Dirac delta.

Such definition allows us to integrate state-of-art evaluation methods that are specialized for some type and are specifying the utility of a message as a single value. Moreover, this approach could be easily modified to take into account the state of the robot at the time of reception, delay of the message, etc. It means that creating an instance of the model for the whole system at hand (i.e., for all information types) might be simplified. For instance, let us assume there is a system that exchanges drones' positions, coordination information, and video stream between UAVs and our goal is to create an instance of the introduced evaluation model for such a system. Good evaluation algorithms for coordination information and video stream

might already be available. We could integrate them into the system using the solution proposed in this section. The only thing left would be to define message utilities for UAVs' positions and weights for each information type.

Message filtering

Dropping some messages is one of the most efficient ways to reduce the amount of data exchanged in a system. In this section we mark the set of all potential messages as $\mathcal{M}$. An evaluation model should be able to help us identify a subset of a predefined number of messages of $\mathcal{M}$ that we can drop with a minimal performance loss.

More specifically, given any order $\leq_{\mathcal{P}(\mathcal{M})}$ on subsets of $\mathcal{M}$ (i.e., filtered messages), one should be able to construct an instance of the model that is able to represent it: the resulting total utilities should maintain the same order. We also handle separately one corner case: no set can be lower than an empty set. It is reasonable: rarely we would prefer receiving no information at all throughout the whole mission over any message. This intuition is formalized using the equation:

$$\forall \mathcal{M}.\, \forall \leq_{\mathcal{P}(\mathcal{M})}.\, \forall \mathcal{M}', \mathcal{M}'' \subseteq \mathcal{M}.\, \exists \mathcal{U}. \quad \mathcal{M}' \leq_{\mathcal{P}(\mathcal{M})} \mathcal{M}'' \implies \mathcal{U}(\mathcal{M}') \leq \mathcal{U}(\mathcal{M}'') \tag{4.35}$$

Another way to interpret this equation is to introduce an adversary knowing all potential messages $\mathcal{M}$ and telling us that some particular subset of exchanged messages is better than another. The adversary specifies a set of such relations between two subsets, which together form the order $\leq_{\mathcal{P}(\mathcal{M})}$. Our role is to find such an instance of the utility model $\mathcal{U}$ that maintains the order for each pair of subsets given by the adversary.

Unfortunately, the following theorem is true:

Theorem 1. *Equation* (4.35) *does not hold in general.*

Proof of theorem 1. Let's consider the following four subsets of $\mathcal{M}$:

$$\begin{aligned} \mathcal{M}^{I} &= \mathcal{M}_1 \cup \mathcal{M}_2 \\ \mathcal{M}^{II} &= \mathcal{M}_1 \\ \mathcal{M}^{III} &= \mathcal{M}_2 \\ \mathcal{M}^{IV} &= \emptyset \end{aligned}$$

where $\mathcal{M}_1$ and $\mathcal{M}_2$ are sets of all messages received by, subsequently, the first and the second agent. We will show that the ordering

$$\mathcal{M}^{I} \geq_{\mathcal{P}(\mathcal{M})} \mathcal{M}^{II} \geq_{\mathcal{P}(\mathcal{M})} \mathcal{M}^{IV} \geq_{\mathcal{P}(\mathcal{M})} \mathcal{M}^{III}$$

cannot be expressed using the model. The model sums up utilities of all agents, so it can be simplified:

$$\mathcal{U}(\mathcal{M}) = \sum_{g \in G} X(\mathcal{M}_g)$$

for some function X. This means that the utility function has to have the following form for the considered sets:

$$\begin{aligned} \mathcal{U}(\mathcal{M}^I) &= x_1 + x_2 \\ \mathcal{U}(\mathcal{M}^{II}) &= x_1 + p \\ \mathcal{U}(\mathcal{M}^{III}) &= x_2 + p \\ \mathcal{U}(\mathcal{M}^{IV}) &= 2p, \end{aligned}$$

for some x_1, x_2 and p (note that $p = X(\emptyset)$).

By applying equation (4.35) we get:

$$x_1 + x_2 > x_1 + p > 2p > x_2 + p,$$

therefore

$$x_1 + x_2 > x_1 + p \implies x_2 > p,$$

but

$$2p > x_2 + p \implies p > x_2,$$

which leads to contradiction. □

Because of the receiver-centric design of this model, it is not possible to freely order subsets that differ in messages received by multiple agents. For instance, consider a mission in which UAVs are photographing an area and sending images to two base stations. It does not matter to which base station a UAV sent a message, but there is no use of sending a photo of the same area to both base stations. Such situations cannot be expressed using the presented model. In real life, we can get around this problem: the base stations could exchange between themselves some meta-data about the area already photographed and based on this the utility of images could be reduced.

Even though there is a family of orders that cannot be expressed using the model, we formulate the following theorem that specifies what can be expressed:

Theorem 2. *Equation* (4.35) *holds with the following constraint:*

$$\forall \mathcal{M}', \mathcal{M}'' \subseteq \mathcal{M}. \left| \{\, m.\text{receiver} \mid m \in \mathcal{M}' \triangle \mathcal{M}'' \,\} \right| > 1 \implies \mathcal{M}' \not\preceq_{\mathcal{P}(\mathcal{M})} \mathcal{M}'' \tag{4.36}$$

where $\triangle$ *is the symmetric difference of two sets, i.e., for any sets* A *and* B, $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

Proof of theorem 2. For each agent $g \in G$ there exists a monotone function $f_g : \mathcal{P}(M_g) \to \mathbb{R}$ that fulfills the following requirements: (1) $\mathcal{P}(X)$ is a power set of X, (2) f_g should preserve the order $\leq_{\mathcal{P}(\mathcal{M})}$ on its domain. We know such a function exists because $\leq_{\mathcal{P}(\mathcal{M})}$ is a partial order on $\mathcal{P}(\mathcal{M})$.

Observe that all f_g have disjointed domains, therefore we can construct a function $f : \mathcal{P}(\mathcal{M}) \to \mathbb{R}$ that uses appropriate f_g functions for parts of its domain:

$$f(M) = \sum_{g \in G} f_g(M_g), \tag{4.37}$$

M_g is a subset of M containing only messages received by agent g.

We set all information type weights ω to 1 and use the following aggregation function:

$$\mathcal{A}_p^{\text{proof}}(M, t, s) = \delta(t) \frac{f(M)}{|\mathcal{P}|} \tag{4.38}$$

where δ is Dirac delta. We calculate the utility given this aggregation function:

$$\mathcal{U}(\mathcal{M}) = \sum_{g \in G} \sum_{p \in \mathcal{P}} \omega_p \cdot \int_0^{t_{\text{end}}} \mathcal{A}_p^{\text{proof}}(\mathcal{M}_g, t, \emptyset) = \sum_{g \in G} f(\mathcal{M}_g). \tag{4.39}$$

Because $\mathcal{M}_g$ contains only messages received by agent g we can simplify it as:

$$\mathcal{U}(\mathcal{M}) = \sum_{g \in G} f_g(\mathcal{M}_g). \tag{4.40}$$

Given this definition of the model let us consider any $\mathcal{M}'$ and $\mathcal{M}''$ such that $\mathcal{M}' \leq_{\mathcal{P}(\mathcal{M})} \mathcal{M}''$. Lets define $G_\triangle = \{ m.\text{receiver} \mid m \in \mathcal{M}' \triangle \mathcal{M}'' \}$, which contains all agents for which there are different messages in sets $\mathcal{M}'$ and $\mathcal{M}''$. Based on that definition, we can observe that

$$\mathcal{U}(\mathcal{M}') = \sum_{g \in G \setminus G_\triangle} f_g(\mathcal{M}'_g) + \sum_{g \in G_\triangle} f_g(\mathcal{M}'_g) = \sum_{g \in G \setminus G_\triangle} f_g(\mathcal{M}''_g) + \sum_{g \in G_\triangle} f_g(\mathcal{M}'_g). \tag{4.41}$$

Because of the constraint introduced in equation (4.36), we know that the set $G_\triangle$ has zero or one element.

If it has zero elements, then:

$$\mathcal{U}(\mathcal{M}') = \sum_{g \in G \setminus G_\triangle} f_g(\mathcal{M}'_g) = \sum_{g \in G \setminus G_\triangle} f_g(\mathcal{M}''_g) = \mathcal{U}(\mathcal{M}'') \tag{4.42}$$

If there is exactly one element in $G_\triangle$, we can mark it as $\hat{g}$. Because $f_{\hat{g}}$ maintains the order $\leq_{\mathcal{P}(\mathcal{M})}$ we deduce that

$$f_{\hat{g}}(\mathcal{M}'_{\hat{g}}) \leq f_{\hat{g}}(\mathcal{M}''_{\hat{g}}). \tag{4.43}$$

By applying this result to equation (4.41) we finally see that

$$\mathcal{U}(\mathcal{M}') \leq \mathcal{U}(\mathcal{M}''). \tag{4.44}$$

□

This theorem means that the order $\leq_{\mathcal{P}(\mathcal{M})}$ can only compare subsets in which all of the changed messages were received by the same agent—messages received by other agents need to stay the same. Specifically, the theorem implies that we can express all situations in which an optimal solution for the whole group of agents could be made by choosing an optimal solution for each agent independently. It also shows that the model can describe all systems in which each agent is aware only of its own knowledge and does not know what messages were received by other agents.

The obtained result assures that even with the added constraint, the model can describe a variety of situations and should be adjustable to any mission at hand. Note that the results are achieved with an unspecified aggregation function. As soon as we introduce some specific aggregation to the model, its expressive power can be vastly reduced in exchange for usability and simplicity granted by that aggregation. In summary, the results mean that for each situation there exists a combination of aggregation functions that gives the model high expressive power.

4.5 *Model application*

The aim of this section is twofold: (1) it shows the usefulness of the model by employing it in a simple mission, and (2) it presents a proof of concept demonstrating a way to integrate the presented model into a robotic system. Additionally, the section could serve as an example of how similar models (used for optimization or for evaluation) can be tested and show what technologies are available.

In order to present the applicability of the model, we show how it can be used to address a real-life problem related to UAV communication. In our scenario, UAVs perform some mission and each of them needs to periodically report its battery level to the ground station. They have two ways to achieve this: either by sending a message over a direct radio link or by using the cellular mobile network and sending the data "over the Internet".

The possibility of evaluating the model in a real robotic mission is shown by implementing it on top of ROS 2, in a virtualized environment that is as close as possible to a real one. A robotic platform was not used because we wanted to present the performance of the model in extreme communication conditions, which might be very hard to control in the real world.

We assume the following characteristics have the most influence on the communication: the Internet connection is reliable (all messages are delivered), but introduces some variable delay; the direct connection has a constant low delay of 10 ms, but sometimes it fails to deliver a message. The UAV can actively measure these characteristics, i.e., it knows the current message delay and drop rate. Based on them, it can choose which communication channel should be used for sending messages. In order to make this decision, it uses the introduced evaluation model.

The model for battery information is specified as in equations (4.9) and (4.10) with the pessimistic battery depletion rate equal to 1% of full capacity per second. The simulated mission takes 100 seconds and during that time the actual battery level depletes much more slowly than in the pessimistic case. We assume a linear depletion, with battery level starting from 100% and reaching 88% at the end of the mission.

The implementation of our example mission logic is very simple: each agent is periodically publishing its battery status once per second. All of the other agents are subscribed to receive these messages. At the end of the mission, the devices are calculating the accumulated utility and saving it for further analysis.

Experiment setup

The proposed model is able to evaluate any exchange of information implemented using any message-based communication protocol and provided all agents use a synchronized time. Such low requirements make the model applicable for many technologies, e.g., built on top of a publish-subscribe communication paradigm, using pull-based approaches, etc. The experiment setup was designed in order to emphasize these properties by making use of frameworks and technologies popular in practical robotic applications and it allows us to directly use the same software on real hardware. At the same time, it allows us to control the communication conditions in order to run reliable and repeatable experiments.

We use *Docker*[1] as a virtualization technology. It improves realism of the system because the interactions between processes running on different agents is limited. Additionally, it forces the solution to work on separate (virtualized) hosts, which from the implementation perspective is very similar to an actual multi-robot system.

[1] https://www.docker.com/

In order to be able to examine a wide spectrum of parameters, the experiments were performed using a network emulator. This allows us to adjust network parameters like communication delays or packet drop rate and run thousands of tests automatically. At the same time,

we benefit from being able to use exactly the same communication solutions as in a real robotic system. The use of a network simulator (like *NS-3*[2,3]) was also considered, but it is hard to integrate such a simulator with popular robotic frameworks; eventually, the aim of this example is to showcase the applicability of the introduced model for robotic systems.

[2] https://www.nsnam.org/

[3] NS-3 can also be used as an emulator, but then the choice between it and Containernet is a subjective opinion.

The network emulator used in the presented experiments is a fork of *Mininet*[4] with added support for Docker containers: *Containernet*[5]. As wireless connections are not supported in Containernet, we decided to model the network in our scenario as a router to which all devices (drones and a base station) are connected (a star topology). In the described case, this simplification is acceptable, because the single communication line between the router and the base station assures only one packet can be transmitted at a time. In real life, channel access methods used in both LTE and 802.11 serve a similar role as this router. The link bandwidth was always set to the minimum available value of 1 Mbps.

[4] http://mininet.org/

[5] https://containernet.github.io/

The simulation framework developed by us first starts a Docker container for each agent in the system, connects them through a virtualized network, and runs a single instance of the experiment. When it detects the end of the experiment, it shuts down all containers and cleans the network configuration. It is also possible to run multiple experiments in parallel. In order to avoid high CPU usage, which might make the results less reproducible, we limit ourselves to run 8 experiments at a time. After each experiment all outputs from the programs are stored together with a log of all captured packets and the experiment's metadata. This data can be then further analyzed.

The experiment running on each agent was implemented in *Python* and *Robot Operating System 2 Crystal Clemmys*[6] with two *Data Distribution Service* (DDS) implementations being tested: *eProsima Fast RTPS*[7] and *RTI Connext DDS*[8].

[6] https://index.ros.org/doc/ros2/

[7] https://github.com/eProsima/Fast-RTPS

[8] https://www.rti.com/

The analysis of the results was conducted using Python in *Jupyter Notebook*[9] and *Wireshark*[10]. The results are visualized with the *Plotly*[11] plotting library.

[9] https://jupyter.org/

[10] https://www.wireshark.org/

[11] https://plot.ly/

Results

Even though the main goal of this section is to show a simple example of how the model can be applied in real life, we decided to use the opportunity and also see how the changes in our setup influence the final outcome. The simulation-based study was conducted in order to compare performance with different number of agents, DDS configurations, and DDS implementations. All of them are run with the assumption introduced at the beginning of this section: small

messages are being exchanged relatively rarely (once per second) in two types of experiments simulating different communication channels: (1) with drop rate varying between 0 and 0.3 and constant delay, presented in figure 4.11; and (2) with drop rate equal to 0 and delay in the range $[0, 1]$ seconds, presented in figure 4.12.

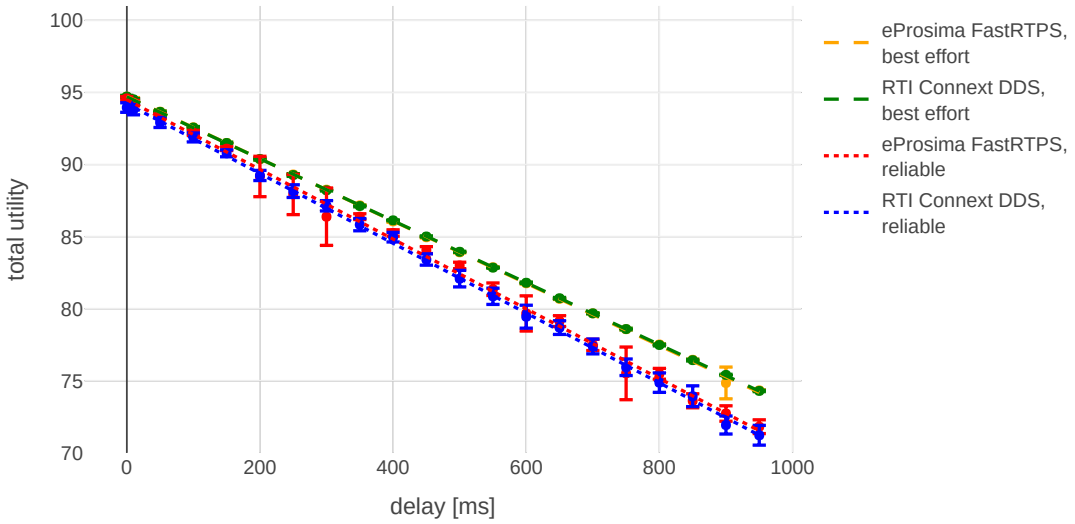


Figure 4.11: Total utility aggregated during the simulation of the exemplary mission (100 s) for connections with different delays.

In both plots, error bars represent 95% confidence intervals and dots mark mean utility values for a given experiment configuration. For each sample point we run at least 50 experiments, often more. The variance of number of experiments is caused by the fact that during some experiments (especially done using lossy setups) the connection was not established at all. We have not included those results in our analysis, i.e., we analyzed only the situations were initial connection was successful.

At the beginning we elaborate about the implementation- and configuration-specific results without getting into details what the specific setting does. Interested readers can find the explanations in the DDS specifications and manuals of the corresponding implementations. At the end of this section we present more generic results, not bound to our configurations, together with ideas how they could be applied.

We compared the following *Quality of Service* (QoS) settings for both DDS implementations in our mission: reliability, durability, and history. The reliability setting allows us to choose between a reliable and best-effort connection. In the best-effort case, each message will just be sent once and no efforts will be taken to re-transmit it. The reliable setting re-transmits messages and tries to ensure all of them

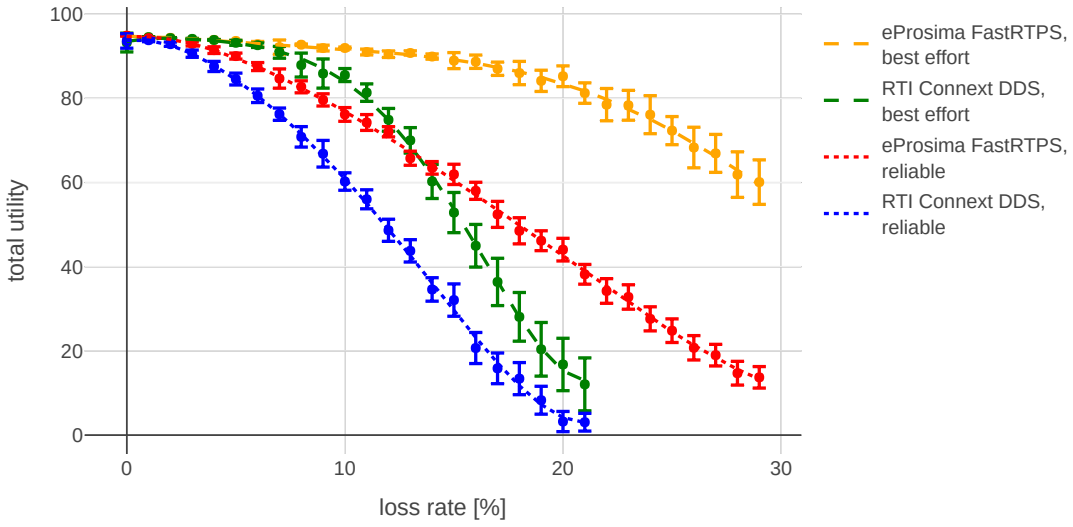


Figure 4.12: Total utility aggregated during the simulation of the exemplary mission (100 s) for a lossy connection. The experiments with Connext DDS with loss rate higher than 21% were infeasible, because the discovery phase was consuming most of the mission time, therefore the communication between robots was often not established at all.

will be delivered in the right order. The durability parameter decides if publishers should remember old messages and send them to late-joining subscribers. The history setting allows us to choose how many past messages are kept in the memory for re-transmissions. Changing any of durability and history parameters did not yield any observable difference. This is probably due to the small packet size compared to available throughput, which means that all messages can be delivered as soon as possible and there is no need to prioritize them. Reliable connection always introduces some overhead. Hence, it has a slightly worse performance than best effort in scenarios where dropping some single messages does not influence the mission a lot. These results can be observed in figures 4.11 and 4.12.

A larger number of agents taking part in the mission on the other hand has a huge effect on the time it took for the setup of the connections between agents. Both DDS implementations are using custom discovery protocols to establish connections between participants. Even with as few as 8 agents, it could be observed that this phase is consuming a significant part of the mission time (around a minute), whereas for 3 to 4 agents it was almost instant (a couple of seconds). This result was expected, because the discovery protocol needs to exchange $O(n^2)$ messages, where n is the number of agents, and was in fact observed by us also in a real-world experiments using robots in one of our previous works [6]. However, after the discovery period is finished the system runs flawlessly for any number of agents between 1 and 10, which is explained by the small amount of data exchanged during the mission. No experiments were conducted with more than

10 devices, because a long discovery period (over a minute) for both implementations of DDS made it infeasible to run a huge number of experiments. It is possible to decrease the discovery time by tweaking the parameters of DDS implementations, but all the experiments were done using the default values and higher number of devices taking part in the mission does not contribute much to the main goal of the described experiments.

We have identified some differences between DDS implementations. In this particular scenario FastRTPS was doing a bit better than Connext, especially over a lossy connection. This is probably caused by smaller packet sizes used by FastRTPS and therefore a lower chance of packet loss. For example, in order to send a message containing a battery level by FastRTPS in a best-effort mode, it transmits 334 bytes "on wire". Connext DDS for the same message utilizes 774 bytes. In both cases, packets contain mainly protocol-specific information. Those differences can be observed in figures 4.11 and 4.12. We are reluctant to make any generic claims about the performance of these implementations based on these results, because of the small message sizes and the fact that the differences start to show up only when packet loss is higher than 0.1. Furthermore, the experiments were performed in the first half of 2019. In the following months ROS 2 and both DDS implementations were very actively developed and improved.

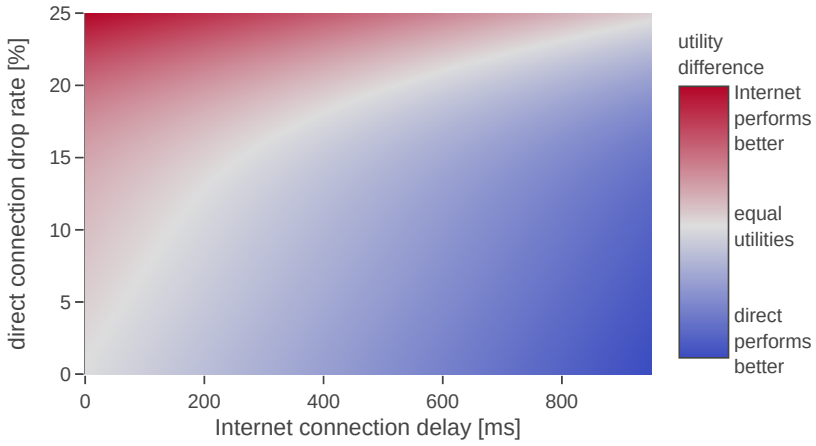


Figure 4.13: Results of the simulation of an exemplary mission.

Based on the presented results, we can construct a 2D plot that visualizes the difference between the utilities of direct and Internet connection. An example of such a plot is given in figure 4.13. In order to create it we have used the parameters that achieved the best performance, i.e., best-effort communication using FastRTPS.

With the blue color (lower right corner) we mark configurations in which the direct connection has higher utility than the Internet connection; deeper blue means bigger difference. The red color (upper left corner) represents that the Internet connection has higher utility. The generated data could be used by a UAV to decide which communication channel to use.

4.6 Summary

Motivated by the fact that the usefulness of a message changes over time and different information types are useful in a different way, we have introduced a utility-based model for evaluation of information distribution in multi-robot systems. The proposed model provides the following benefits: First, it is able to handle different types of exchanged messages and information conveyed by these messages in a uniform way. This allows us to evaluate information distribution independently of the mission using a unified framework. Second, it provides a layer of abstraction between the mission and algorithms working on the information distribution scheme (e.g., evaluating or optimizing it), providing a common representation of the usefulness of each message for the mission. Third, it takes into account only the received messages. This makes the model applicable to real-life robotic systems, even if the communication channel is not reliable. Fourth, it does not depend on the feedback about the message reception, thus it does not cause any additional overhead in the network.

Furthermore, the presented application of the model demonstrates its usefulness and serves as a proof of concept, showing how the evaluation model can be integrated into robotic systems and which technologies can be used in order to achieve that. A practical system based on the model is presented in chapter 7.

We believe the work presented in this chapter could be useful not only in the specific use-case presented in the rest of this thesis, but could universally serve as a solid foundation for a variety of research topics (from a theoretical analysis of different communication solutions to practical applications in multi-robot systems). In particular, the model presented here has already been used to decide when to re-transmit outdated information in one of the works conducted by our research group [7].

Other interesting topics for future work include an effort to categorize utility functions in order to facilitate model definition for system engineers. For instance, one could prepare a set of functions that a system engineer can choose from and provide a default set of parameters together with instructions which function can be used when. Then, a person integrating the model into a new system would just

choose from the predefined options instead of considering all possible functions. Of course, this approach could also be merged with a generic one, where a function could be specified freely. Stretching the idea even further, future work could consider research aiming at automatic definition of message utility functions based on mission objectives or even based on data gathered during mission execution (in the spirit of reinforcement learning).

Extensions of the model could also be considered. The fact that the information might have been corrupted during communication could be taken into account. An interesting extension could include an assumption that the agents are not cooperating and some of them should not receive some information.

In the next chapter we will show how the model can be used with an optimization method to create a generic and automatic way of choosing which messages are worth sending and which ones can be dropped.

5 *Optimization of information distribution*

In this chapter, we used the previously introduced evaluation model to actively decide which information to share. Our solution consists of two blocks, marked with green color in figure 5.1. The first block is a generic optimization method that can be transparently incorporated into any application. The second one is the aforementioned evaluation model. An evaluation model taking into account the specifics of a mission at hand can bring application-awareness into the information distribution middleware. To the best of our knowledge, the method proposed by us is the first one which holds all of the following three properties: it is transparently integrable into existing systems; it jointly considers exchange of various information types; and it is able to work in real-time (online) on robots.

This chapter describes the following scientific contributions.

- Design of an online and transparent information distribution middleware that comprehensively (for all information types) optimizes information exchange in an MRS taking into account non-myopic, mission-aware message utilities (section 5.1).

- Introduction of two techniques to reduce the decision space of the proposed method, thus facilitating its use in practical applications (section 5.3).

- Method for mission characterization based on the difficulty of the information distribution optimization problem in a given mission (section 5.4).

- Proof of concept implemented in Robot Operating System 2 (ROS 2) on mobile robots and a simulation study performed using a network simulator (ns-3) (section 5.4).

Major parts of this chapter are taken from the work already published in Frontiers in Robotics and AI [3].

Contents

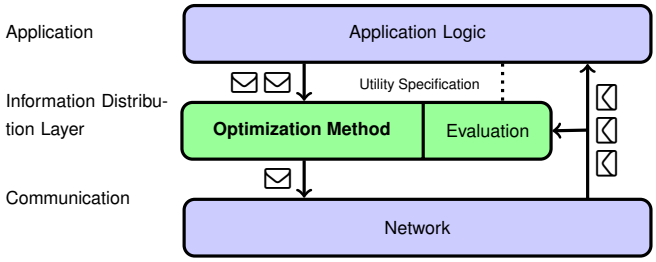


Figure 5.1: Design of the proposed information distribution middleware. Application logic generates messages that are then intercepted by the introduced information distribution layer. The optimization method decides which ones are worth forwarding to the communication layer (i.e., sending). In order to do so, it utilizes the evaluation model, which can be adjusted for the mission at hand using utility specifications.

5.1 *Problem formulation*

We focus on deciding which messages should be sent to maximize the obtained utility, constrained by the limited available communication resources, such as energy or throughput. Formally, such an objective can be expressed as the following optimization problem to be solved collaboratively by the MRS:

$$\begin{aligned} &\underset{M_{\text{rcv}} \subseteq \mathcal{M}}{\text{maximize}} && \mathcal{U}(M_{\text{rcv}}) \\ &\text{subject to} && \text{network constraints (equation (5.5))}, \end{aligned} \tag{5.1}$$

where M_{rcv} is a set of messages received by any agent, $\mathcal{M}$ is a set of all messages that can be generated, and $\mathcal{U}$ is an evaluation model that returns a utility value. The evaluation model $\mathcal{U}$ is an important part of this approach. In practice, many formulations of $\mathcal{U}$ employ compromises to allow for efficient methods to solve the maximization problem. Even though our approach was inspired by the evaluation model introduced in chapter 4, the optimization method could work for any definition of $\mathcal{U}$. Specifically, we assume the evaluation model can assign any value to any subset of messages, which makes our optimization method generic. Such a formulation of the problem has a complexity growing exponentially with the number of messages [85]. The messages can influence each other in arbitrary ways, thus all possible subsets need to be considered. Moreover, as the number of messages is usually high and the decisions need to be made fast, looking for an optimal solution is infeasible. Hence, most research efforts tackling this problem (including this paper) are focused on identifying heuristics that are applicable in practice [29].

System design

The system setup presented here is similar to the other ones introduced earlier in this thesis and highly motivated by the available hardware (described in section 3.1). It consists of multiple agents (e.g., robots) exchanging messages to pursue a common objective. The agents broadcast messages over a single wireless channel giving

them knowledge of all communication in the system. It is assumed that agents are working within each other's communication range, though in principle multi-hop connections could also be used. We assume an efficient channel access method that guarantees a fair usage of the communication medium and resolves potential message collisions. The robots are connected using an IEEE 802.11 wireless network in an ad-hoc mode. This setup is motivated by a realistic application addressed in the KPK project (described in section 1.1).

We design the solution as a transparent middleware that can be integrated into existing systems. A diagram of the design is presented in figure 5.1. The solution could be perceived as an additional layer introduced between the application logic and the network. It consists of two parts: the optimization method and an evaluation model. The evaluation model is aware of all messages exchanged in the network, but it does not modify them in any way. In principle, a very generic model can be used, possibly completely independent of the mission at hand. The evaluation model (chapter 4) is designed specifically to be used in this setup. It allows the optimization method to make mission-aware decisions and is able to consider both myopic and long-term impact of a message.

All messages are handled as follows. First, a message is generated by the application logic. Second, it is processed by the information distribution middleware deciding to either drop it or forward it to the network layer (i.e., send it), which finally transmits it to the other agents. The objective of the information distribution middleware is to make this decision in such a way that the received utility, as defined by an evaluation model, is maximized, taking into account the constraints.

Our approach requires the knowledge of the generation time of all messages (also the ones that are yet to be generated in the future). Often reliable mission-specific estimations for these times can be designed. For instance, an agent could be sending a message whenever it enters a warehouse or its battery is low. In this work, we estimate the generation time by assuming the messages are generated periodically. This assumption often holds in practical situations. For example, robots generate sensor data, such as their position, with a predefined, fixed rate. As soon as the first position message is received, we can estimate the generation times for all the future position messages.

5.2 *Monte Carlo Tree Search*

The method proposed by us is based on Monte Carlo Tree Search (MCTS) [86, 87]. MCTS is a heuristic decision algorithm based on

Monte Carlo method. It is widely used in the design of artificial intelligence, for instance, in order to play games. In fact, the method was first introduced to play Go [86] and since then almost dominated the field [88]. Perhaps the most remarkable program utilizing this method is *AlphaGo* developed by *DeepMind Technologies* (formerly *Google Deepmind*), which was the first computer program to beat a professional Go player [89]. The method shown its applicability also in other problems, like playing nondeterministic video games [90], planning [91], and the problem of information distribution in MRS [37].

MCTS is based on classic Monte Carlo methods, where the object of study (e.g., a continuous field) is randomly sampled in order to estimate some properties of this object. In the case of MCTS the object of study is a decision process and it is being randomly sampled by traversing a tree where each node represents a decision.

Base of operation

The following description explains the generic base of operation of the basic version of the MCTS algorithm. The description is intertwined with an example that presents how this algorithm can be used in order to solve the problem of information distribution in MRS.

The MCTS algorithm starts with a tree consisting of a single root node. This node represents the state where none of the decisions has been made yet.

Example. *Let us assume we have a set of* 4 *messages:* $\{m_1, m_2, m_3, m_4\}$ *and we need to decide which ones of them are worth sending. We need to make* 4 *decisions: for each message we should decide if it should be sent or dropped. This state is represented by the root node.*

The method involves running a number of simulations. Each simulation involves testing a different option for each decision and examining the outcomes. The more simulations, the better the estimation.

For each simulation the algorithm performs four steps: Selection, Expansion, Simulation, and Backpropagation. The steps are described in the following subsections and summarized in pseudocode in algorithm 1. Additionally, in section 5.2 we provide references to the Python source code used in our experiments.

Selection

The algorithm starts from a root node and traverses the tree down until a leaf node is reached. A leaf node is a node without any children. For each node it chooses the child to go to based on the UCT

formula (given in equation (5.3)). The selection step is described with pseudocode in line 8 of algorithm 1 and an example of this step is presented in figure 5.2.

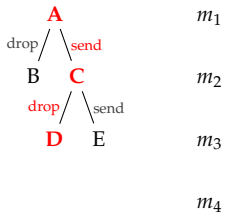


Figure 5.2: Example of the selection step. Node D was eventually selected, which represents a state in which message m_1 is sent, m_2 is dropped and the decision about the other messages is not yet made.

Expansion

If in the leaf node there are still some unresolved decisions (e.g., messages for which we did not decide if they should be sent or not), in the expansion step we choose one of them and create as many children as there are options for the considered decision. In principle the decision could be chosen at random, but usually some heuristic is utilized to first consider decisions that are more significant. In our implementations the decisions (i.e., messages) are ordered by generation time to assure deterministic outcomes.

Then, one of the newly created children is chosen randomly and considered in the next step. We will call this chosen child the *expanded child*.

The pseudocode for this step is provided in line 13 of algorithm 1.

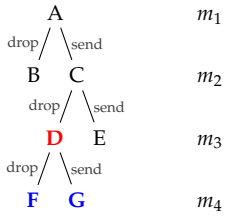


Figure 5.3: Example of the expansion step. Node D is being expanded and nodes F and G are the newly created children nodes.

Example. *This example is visualized in figure 5.3. Let us assume we are in a leaf node D. There are still two messages for which the decision was not made: m_3 and m_4 generated at, respectively, times 3 s and 4 s. Message m_3 is generated earlier, so we will consider options related to it first. We can either send it or not. Hence, we create two children nodes — one representing a situation when m_3 is sent (G), the other when it is not (F). In both of these children the decision regarding message m_4 is still unknown.*

Simulation

The goal of this step is to estimate the value of the expanded child form the previous step. In classic MCTS at this point all of the unresolved decisions will be made randomly. Then, the resulting state would have to be evaluated which would result in a numerical value being assigned to it (for instance, 1 for a won game and 0 for a lost game).

However, often the set of decisions might be big and hence even performing them randomly might be computationally intensive. An alternative approach to this step is to asses the value of a given state without considering future decisions. This assessment could be done, for instance, by utilizing domain-specific expert knowledge or be based on machine learning.

This step is presented in line 21 of algorithm 1.

Example. *Let us assume node F was chosen as the expanded child. We can utilize some evaluation method in order to assess the situation in which message m_1 is sent and messages m_2 and m_3 are dropped. The method could,*

for instance, assume that bigger messages are more useful. As a result a numerical score should be assigned to node F.

In our implementation, this step is realized using the evaluation model described in chapter 4.

Backpropagation

Executing the simulation step provided a new information about the expanded node. This information needs to be propagated to all its ancestors in order to improve their estimated values. The ancestors of node *F* are marked in figure 5.4 and the pseudocode of this procedure starts in line 23 of algorithm 1.

In order to make the final decision, we start from a root node and then always go to the child that was visited the most times. When the child that represents the decision we are interested in is reached, we have the result. This procedure is described with pseudocode in algorithm 2.

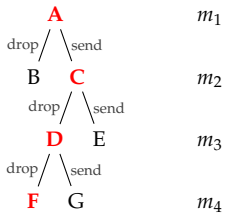


Figure 5.4: Example of the backpropagation step. All nodes that should be updated are marked with red color.

Implementation

In order to use MCTS in our experiments we have utilized a free and open-source Python implementation available on Github[1]. We have forked it and ported to Cython[2] in order to improve performance. Additionally, we changed the selection step to be deterministic in order to make our experiments reproducible.

[1] https://github.com/ImparaAI/monte-carlo-tree-search
[2] https://github.com/zeroos/monte-carlo-tree-search

Algorithm 1: The MCTS algorithm. The main procedure starts in line 28.

```
 1: class Node
 2:     Node parent
 3:     Node[] children
 4:     bool expanded := False
 5:     Decision[] unresolved_decisions
                                  ▷ e.g., a set of messages to decide on
 6:     int visits := 0
 7:     float value := 0

 8: function SELECT(tree_root)
 9:     Node node := tree_root
10:     while node.expanded do
11:         node := child of node with the highest UCT score
12:     return node

13: function EXPAND(node)
14:     decision := choose one of the unresolved decisions in node
15:     option := choose one of the options for decision
                                                  ▷ e.g., drop a message
16:     expanded_child := create a child representing option
17:     node.children.APPEND(expanded_child)
18:     if all options for decision were considered then
19:         node.expanded := True
20:     return expanded_child

21: function SIMULATE(node)
22:     return estimated value of node

23: function BACKPROPAGATE(node, value)
24:     node.value + = value
25:     node.visits + = 1
26:     if node is not tree root then
27:         BACKPROPAGATE(node.parent, value)

28: function MCTS(decisions)
29:     Node tree_root
30:     tree_root.unresolved_decisions := decisions
31:     while insufficient number of simulations is performed do
32:         Node node := SELECT(tree_root)
33:         Node expanded_child := EXPAND(node)
34:         float value := SIMULATE(expanded_child)
35:         BACKPROPAGATE(expanded_child, value)
```

Algorithm 2: Procedure that allows us to make a decision based on a tree constructed using MCTS simulations.

```
1: function DECIDE(node, decision)
2:     best_child := the most visited child of node
3:     if node represents decision then
4:         return a result of decision in best_child
5:     else
6:         return DECIDE(child_node, decision)
```

5.3 *Information distribution optimization with Monte Carlo Tree Search*

In the problem of information distribution, MCTS enables an efficient traversal of a decision tree, considering more promising options first. It executes a pre-selected number of iterations:. the higher the number of iterations, the better is the result, eventually leading to the optimal solution. This allows us to easily balance the exploration depth (thus better optimization results) and computation time needed for each decision. This ability was the main reason why we have decided to utilize this method. Furthermore, the fact that MCTS eventually explores the whole tree is also beneficial for the problem of information distribution. Other heuristic approaches sometimes discard parts of the solution space as not worth exploring, which in general might result in a sub-optimal solution even with unlimited computational resources.

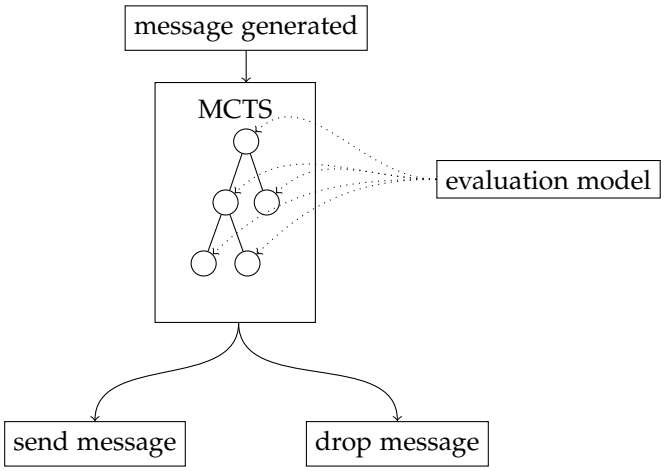


Figure 5.5: Flow of the messages visualized using a diagram.

A diagram visualizing how the MCTS is incorporated into the message flow of the system is presented in figure 5.5. Each time an agent needs to make a decision whether it should send a message or not, it builds a new MCTS tree. All generated messages from all

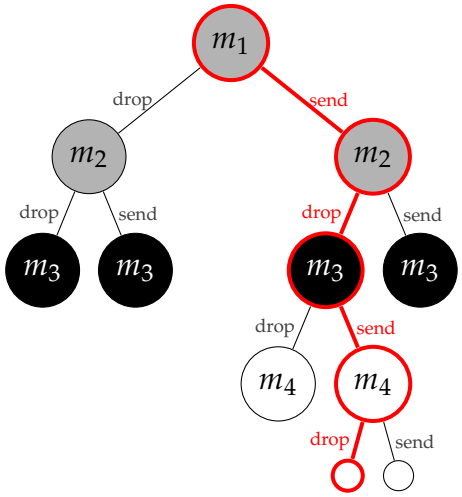


Figure 5.6: Example of a tree constructed by MCTS. The black nodes represent the message to decide about. Grey nodes represent messages generated in the past that might have not been received yet. White nodes represent future messages.

agents are incorporated into this tree. Hence, the agent takes into account other agents, which allows it to make decisions optimized for the whole system. The information about messages generated by other agents is not always available and has to be estimated (for instance, in the case of future messages). In this work the estimation is done using the assumption about periodic messages (see section 5.1) and their expected utilities are estimated using mission-specific estimators (see chapter 4 for the analytical model and chapter 7 for the practical realization). A similar approach would also work for other setups where message generation times are known or can be reliably estimated.

Each node at the i-th level of the tree represents a decision: whether the i-th message should be sent or dropped. To take into account the messages that might still be in transfer, we construct the decision tree including also messages generated in the past. The maximum age of messages to consider is based on the current network latency.

An example of a decision tree obtained using MCTS is visualized in figure 5.6. For the sake of the example, we assume that each message is generated by a different agent. In order to make the example easy to follow, it consists of only four messages and shows a result of a relatively low number of iterations of the MCTS algorithm. This particular tree is constructed in order to decide if message m_3 should be sent or not. In practice the tree would have been explored much further into the future. However, even in such a simplified setup it is possible to observe the asymmetrical growth of the tree. It occurs because the MCTS explores more promising options first. The red outline marks the path that is the output of the algorithm. The decisions represented by this path are likely to provide the highest utility. Based on it the agent would choose to send message m_3.

Even though the MCTS provides decisions for all considered messages, we use it only to decide about the one just generated. There are multiple reasons for this design choice. First, before the next message is generated, the agent's knowledge might change. This could influence the estimated utility values and hence invalidate the previous decisions. Second, a message considered in the tree might be generated by another agent, which is able to better estimate its value. Algorithm 3 explains in pseudo-code what happens when a message is generated or received by an agent.

Algorithm 3: Pseudocode summarizing what happens when a message is generated or received.

```
1:  class Agent
2:      MessageSet messages := empty set
3:      On message received do
4:          messages.APPEND(message)
5:      On message generated do
6:          predicted_messages := a set of messages predicted to be
    send in the future
7:          decisions := messages ∪ predicted_messages ∪ {message}
8:          tree := construct MCTS containing decisions
9:          based on tree decide if message should be send or not
10:         if message should be send then
11:             messages.ADD(message)
12:             send message
13:         else
14:             drop message
```

The key design decision in an MCTS-based method is how to assign rewards for final states and how to calculate scores of intermittent states. Typically, after each tree expansion step, the MCTS algorithm simulates the whole execution of the decision process with random future decisions. However, in our problem, this is infeasible. Thus, we utilize another common approach: instead of performing the MCTS simulation step, we score the states using a scoring function. The scoring function is a problem-specific heuristic that provides a simple way to determine which states are better than others. We decided to formulate it based on the ratio between the value of utility in the given state and a maximum value of utility, achieved by receiving all messages. Specifically, we use the following scoring function:

$$S(v) = \frac{\mathcal{U}(M_v)}{\psi\mathcal{U}(\mathcal{M})}, \tag{5.2}$$

where S is a function assigning a score to a node v of the decision

tree, $\mathcal{U}$ is the evaluation model returning the utility of exchanged messages, M_v is the set of messages exchanged up to node v, $\mathcal{M}$ is the set of all messages that could ever be transmitted during the mission and ψ is a value representing the progress of the mission. In our experiments, we set $\psi = \frac{t}{t_{\text{end}}}$, where t is the generation time of the message considered in v and t_{end} is the duration of the mission. In the simulation-based study (section 5.4) the value of $\mathcal{U}(\mathcal{M})$ is known, but in the practical experiments (chapter 7) we estimate it.

For now, we did not explicitly consider any network constraints. Without this, MCTS could decide to transmit all messages. An interesting way to include the network constraints would be to make them a part of $\mathcal{U}$. Such a $\mathcal{U}$ could utilize a network simulator (e.g., ns-3) in order to estimate the expected value of sending the given subset of messages and, for instance, taking into account message drop rate and delays that could occur in a network. Assuming a reliable network simulator, the solution would take into account that sending too many messages results in potential loss and latency increase, which in turn decreases utility. Even though we find such solution interesting, it might be infeasible given the overhead associated with network simulation. Hence, in the implementation we estimate the future latency as an average observed latency and use the technique described in section 5.3 to introduce network constraints.

An inherent part of such decision problems is a balance between exploration and exploitation[3]. To give some intuition, in our problem an exploitation-only solution would be a greedy method that sends a message whenever it increases the utility, whereas an exploration-only approach would prioritize trying previously untested decisions. To maintain a balance between these two extremes in MCTS, often the UCT formula [87] is used to decide which node to go to in the MCTS selection step (cf. section 5.2). Always the node with the highest UCT value is chosen. The UCT formula is:

[3] The concept refers to the situations in which an algorithm needs to decide to choose a solution that is known to provide good results (exploit) or test a new option to look for a better solution (explore).

$$\text{UCT}(v) = \overline{S} + c\sqrt{\frac{\ln n_p}{n_v}}, \tag{5.3}$$

where $\overline{S}$ is a mean score (equation (5.2)) of all descendants of v, n_p is the number of visits of the MCTS algorithm to the parent node of v, n_v is the number of visits to node v, and c is called a discovery factor. High values of c result in a more exploratory behavior, whereas low values promote exploitation of the already constructed tree. In our experiments, we fix c to 0.35 in order to prioritize high-valued decisions (i.e., sending the message).

Reduction of decision space

In the following, we propose two methods to reduce the decision space with the aim of increasing the performance of the optimization middleware. The first method considers network constraints and is applicable to all information types, albeit providing only a small performance boost. The second method, on the other hand, is applicable only to specific types of information, but is able to reduce the tree growth from exponential to polynomial.

Reduction based on network constraints

So far, our approach considered any subset of messages to be sent, even if transmitting them would cause obvious overuse of resources (e.g., causing network congestion). Unfortunately, it is difficult to predict how the network will perform under such stress. In principle, this could be done using a network simulator. However, such an approach has two major problems. First, simulating transmission of many messages at each node of the decision tree requires considerable computational resources if an advanced simulator is used, whereas using simple simulations provides less reliable results. Second, in practice other processes (i.e., not considered by the information distribution middleware) might also run in the network and their behavior might be impossible to simulate (e.g., a system using aggressive retransmission schemes could cause congestive collapse of an overloaded network).

To address these problems we introduce the following reduction. In addition to avoiding resource overuse, it enables simple simulations to be used and at the same time reduces the decision space of the MCTS. At the core of the reduction lies the following predicate:

$$\forall t \in [0, t_{\text{end}}].\ |W(t, M_{\text{sent}})| \leq A(t), \tag{5.4}$$

where M_{sent} is the set of sent messages, W is the windowing function that returns a subset of all messages affecting resource utilization at time t (i.e., being transmitted at time t), operator $|\cdot|$ measures how much resource is used by a subset of messages and $A(t)$ specifies how much resource is available at time t. Intuitively, such a predicate is false if and only if there exists a moment when the resource is overused. To give an example, if $A(t)$ is a constraint specifying the number of messages that can be simultaneously transmitted, then the operator $|\cdot|$ is simply a power of subset returned by a windowing function. In such a case, the predicate is false if at any time t there are more messages being transmitted than is allowed by $A(t)$.

However, in practice, evaluating such a predicate for each time instant is infeasible. Therefore, we introduce an additional assumption:

a message consumes a constant amount of resources for the whole time of transmission, e.g., it is being transmitted with a constant speed. This allows us to reduce the predicate from equation (5.4) to the following one:

$$\forall t \in \{\, m.t_{\text{sent}} \mid m \in M_{\text{sent}} \,\}.\ |W(t, M_{\text{sent}})| \leq A(t), \tag{5.5}$$

where $m.t_{\text{sent}}$ is the moment when m is sent. This predicate allows us to check the same condition but can be evaluated only in discrete moments, i.e., when a message is sent.

Such a predicate can be defined for a variety of resources, e.g., network goodput, or energy. For each tree node in the decision process, we check the truth values of such predicates. If any of them is false, we do not consider the node nor its descendants in the decision process. The additional computational complexity of this check depends on the specific definition of a predicate, but for many practical predicates (e.g., limited network goodput) it can be realized in amortized constant time[4] (for details see our implementation [92]). In the following paragraphs we present analytical results summarizing the efficiency of the proposed constraint.

[4] This means that the check would on average require constant time, but sometimes might incur more computations. In our case, the first check requires a number of operations proportional to the size of the window and the next ones can be done in constant time.

Many of such predicates can be simplified to a constraint that allows to send only less than r messages in a window. We denote the number of nodes at the i-th level of the tree as $F(i)$. k is a number of messages generated in a window. We assume the constraint states, that only $r-1$ messages can be sent in one window. We number the levels in the tree starting with level 0 (i.e., in figure 5.6 message m_1 is at level 0, m_2 at level 1, etc.). This means that level k represents the decision about k messages.

Then, we can formulate F as a recursive function:

$$F(i) = \begin{cases} 2^i - \sum_{j=r}^{i} \binom{i}{j}, & \text{for } i \leq k \\ 2F(i-1) - \binom{k-1}{r-1}, & \text{for } i > k. \end{cases} \tag{5.6}$$

After expanding it, we get:

$$\begin{aligned}
F(i) &= 2F(i-1) - \binom{k-1}{r-1} = \\
&= 2\left(2F\left(i-2\right) - \binom{k-1}{r-1}\right) - \binom{k-1}{r-1} = \\
&= 2^2 F\left(i-2\right) - (2+1)\binom{k-1}{r-1} = \\
&= 2^{i-k}F(k) - \left(\sum_{j=0}^{i-k-1} 2^j\right)\binom{k-1}{r-1} = \\
&= 2^{i-k}F(k) - \left(2^{i-k}-1\right)\binom{k-1}{r-1} = \\
&= 2^{i-k}\left(2^k - \sum_{j=r}^{k}\binom{k}{j}\right) - \left(2^{i-k}-1\right)\binom{k-1}{r-1} = \\
&= 2^i - 2^{i-k}\sum_{j=r}^{k}\binom{k}{j} - \left(2^{i-k}-1\right)\binom{k-1}{r-1}.
\end{aligned}$$

In general, the decision space with such constraints still grows exponentially with the number of messages. However, if the resource is very limited, the reduction can be significant. On the other hand, when the resource is abundant, there is almost no impact on the size of the tree. To give some intuition: in the case when all messages can be sent, the constraint has no impact, i.e., the number of nodes stays the same and is equal to 2^i. When $r \approx \frac{k}{2}$, the number of nodes is approximately 2^{i-1}, so it allows us to consider one more message with the same computational effort. Finally, if only a few messages can be sent (specifically, $r << k$), the number of tree nodes is reduced to below 2^{i-k}, allowing us to consider up to k more messages with the same computational cost.

To derive the above results, we approximated the value of F with $r \approx \frac{k}{2}$ and with r much smaller than k. To obtain the first of these approximations, we first simplified the computations by considering only even k. The order of magnitude of the obtained result is not affected by this assumption. By calculating the value of F we obtain:

$$\begin{aligned}
F(i) &= 2^i - 2^{i-k}\sum_{j=r}^{k}\binom{k}{j} - \left(2^{i-k}-1\right)\binom{k-1}{r-1} = \\
&= 2^i - 2^{i-k}\sum_{j=\frac{k}{2}}^{k}\binom{k}{j} - \left(2^{i-k}-1\right)\binom{k-1}{r-1} = \\
&= 2^i - 2^{i-k}2^{k-1} - \left(2^{i-k}-1\right)\binom{k-1}{r-1} = \\
&= 2^{i-1} - \left(2^{i-k}-1\right)\binom{k-1}{r-1} < 2^{i-1}
\end{aligned}$$

The second approximation is derived in the following way:

$$\begin{aligned} F(i) &= 2^i - 2^{i-k} \sum_{j=r}^{k} \binom{k}{j} - \left(2^{i-k} - 1\right) \binom{k-1}{r-1} = \\ &= 2^i - 2^{i-k} \left(2^k - \sum_{j=0}^{r-1} \binom{k}{j}\right) - \left(2^{i-k} - 1\right) \binom{k-1}{r-1} = \\ &= 2^i - 2^i + 2^{i-k} \sum_{j=0}^{r-1} \binom{k}{j} - \left(2^{i-k} - 1\right) \binom{k-1}{r-1} = \\ &= 2^{i-k} \sum_{j=0}^{r-1} \binom{k}{j} - \left(2^{i-k} - 1\right) \binom{k-1}{r-1} \end{aligned}$$

by applying the binomial theorem we can limit it to:

$$\begin{aligned} F(i) &\leq 2^{i-k}(1+k)^{r-2} - \left(2^{i-k} - 1\right) \binom{k-1}{r-1} \\ &< 2^{i-k}(1+k)^{r-2} \end{aligned}$$

Hence, if the number of messages that can be sent tends to 0 (only a few messages can be sent), the number of nodes tends to being lower than 2^{i-k}.

In our experiments, we use one such predicate for a *network goodput constraint*: we define A as a constant function returning fixed maximum network goodput, operator $|\cdot|$ as the sum of the sizes of all messages in a subset divided by the window length, and use the following windowing function:

$$W(t, M_{\text{sent}}) = \{\, m \in M_{\text{sent}} \mid t \leq m.t_{\text{sent}} \leq t + T \,\}, \tag{5.7}$$

where T is a parameter defining the window length. Low values of T result in a constraint that does not allow many messages to be sent at similar times. High values of T provide fewer restrictions, but might result in a temporary resource overuse. For the experiments we have empirically set this value to $T = 0.5\,\text{s}$, since it provides a middle-ground between the two described extremes and works well in our experimental setup.

Reduction for Markovian information types

Many information types used in MRS exhibit a Markovian property, i.e., only the most recent message of such a type is useful (brings utility). In this section, we present a reduction method that significantly limits the decision space for such information types.

Let us recall that the messages in the tree are ordered by generation time. We define the *state* of a node in the decision tree as a set

$\{ \tau_p \mid p \in \mathcal{P} \}$, where $\mathcal{P}$ is the set of all information types and

$$\tau_p = \begin{cases} m_{p,\text{last}} & \text{if } p \text{ is Markovian,} \\ M_p & \text{otherwise,} \end{cases} \tag{5.8}$$

where $m_{p,\text{last}}$ is the most recently received message of type p and M_p is the set of all messages of this type received so far.

If two nodes at the same level of the decision tree are sharing the same state[5], we will call them *tantamount nodes*. We observe that any decision made in a descendant of one tantamount node has exactly the same influence on the mission as a corresponding decision in any other tantamount node. Consequently, the utilities achieved from these future decisions are also equal. Therefore, it is enough to consider a tantamount node that brings the highest utility, as the other ones will for sure not perform better. It means that all tantamount nodes with utility lower than the highest one do not have to be expanded anymore.

[5] For completeness, the state should also contain an information about the network constraint utilization. It is not included for the sake of simple presentation. In particular, the presented considerations hold without changes if the constraint from equation (5.5) is used with window size smaller than the publishing periods of Markovian message types.

At each level of the decision tree there is a finite number of states equal to:

$$2^{\hat{n}} \prod_{p \in P_M} n_p, \tag{5.9}$$

where $\hat{n}$ is a number of messages from non-Markovian information types, P_M is the set of Markovian information types and n_p is the number of messages of type p generated so far. For each state, all tantamount nodes are reduced to a single node. Thus, after this reduction the number of nodes in the tree grows exponentially only with respect to messages of non-Markovian information types. For Markovian information types it grows polynomially.

This observation can be incorporated into the MCTS algorithm by maintaining a set ξ_i of the best nodes for all states at each level i of the decision tree. Each time the tree is expanded, the algorithm checks if the newly expanded node v is better than its tantamount node v' in the set ξ_i of the current level. If this condition holds, the descendants of v' are attached to v, v' is replaced by v in ξ_i, and all statistics of the tree are updated to accommodate this change. In a case where v achieves worse utility than v', the expansion is canceled and not considered in the future.

This modification introduces only a small computation overhead of maintaining a set of states after each simulation of MCTS. To analyze computational cost we assume there are n messages in the tree. If, for the implementation of this specific set, a tree-based set is used, additional $O(\log(n))$ operations per simulation are needed. Whereas, when using hash table, the expected number of additional operations is just $O(1)$ and in the worst case it is $O(n)$. Both of these

solutions introduce a negligible cost in practice, because each simulation has to perform $O(n)$ operations anyway. The overhead is small compared to the benefit of greatly reduced decision space.

5.4 Evaluation

The results related to this chapter are twofold. First, in a simulation study we show that the introduced method produces good results in a multitude of applications. Second, in chapter 7, through a robotic proof of concept, we present the applicability of the proposed method in a real-world scenario.

However, arguing about the performance of a solution to an information distribution problem is inherently complicated due to the multitude of applications and conditions to consider. For instance, a method adjusted for distributed photogrammetry might not work well for a multi-robot search and rescue mission. Nevertheless, state of the art publications addressing the problem of information distribution are usually ignoring this aspect and evaluating the solutions using specific applications [24–26, 28–30]. Hence, in the next subsection, we propose a novel evaluation methodology that presents the performance of our solution in different scenarios. Then, we use it to present the rest of the results.

Evaluation method

Instead of focusing on a specific application, we introduce a mission *characteristic* that intuitively describes the difficulty of the information distribution problem for a mission. Then, we present three scenarios with different characteristics and show how our approach performs in them.

As a characteristic of a mission, we could use a distribution of utilities of all generated messages in the mission: what percentage of messages provided high utility, what low, etc. Such a characteristic would allow us to identify how important it is to optimize information distribution for a given application. However, sending one message influences the utility of other messages. Thus, it is not straightforward to obtain such statistics. For instance, in one execution of the mission a message could be very valuable, whereas in the other it could provide almost no utility, because the same information was shared using a different message. Hence, instead of using a distribution of utilities we approximate the distribution of expected utilities of all messages. In order to acquire the proposed statistics, we first perform experiments using an algorithm that randomly drops messages with different probabilities. Then, for each message

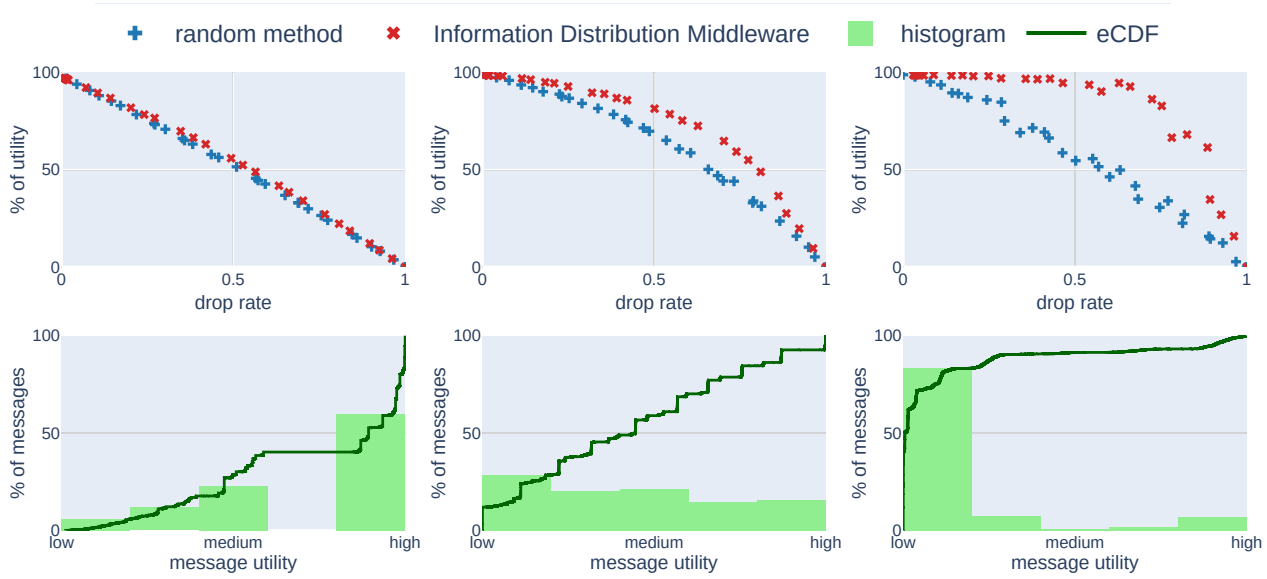


Figure 5.7: Results of comparison between the baseline random method and the Information Distribution Middleware in the simulation study. In scenario (a) most of the exchanged messages are of high utility. In scenario (b) there is a uniform distribution of messages with respect to utility. Finally, in scenario (c) most of the messages are of low utility. These plots can be reproduced by running the `DropRateVsUtility` experiment in our software framework [92].

exchanged during any of the experiments we compute how much utility it provided on average for the mission and plot the distribution of these values. The results in figure 5.7 are obtained by running 30 simulations with different message drop rates, resulting in around 12000 exchanged messages.

We consider three scenarios. In *scenario (a)* a majority of messages are very important for the mission, hence have a high utility value. An example of such a mission could be an application of unmanned aerial vehicles (UAVs) to map an unknown area and share with others map fragments that have not yet been explored. Then, each map fragment is equally important and hence brings similarly high utility. In such a mission even if the information distribution optimization is realized perfectly, the benefit is very low, because even a random method would choose to send most of the valuable messages. On the other hand, in *scenario (c)* a majority of messages is not useful and there are just a few very important messages. For this case an example could be a search and rescue mission performed by UAVs, which are periodically sharing their status. For most of the time UAVs are just reporting the fact that they did not locate a victim. However, the messages stating that the victim has been found are much more valuable, but also relatively rare. In this scenario the main goal of the information distribution optimization is to find and deliver these important messages. For such applications it is fairly simple to showcase the superiority of some information distribution approach over a random scheme. Even a very simple solution that transmits only

the high-valued messages would provide results much better than a random approach. Apart from these two extremes one can think about scenarios in between, with similar numbers of messages of different utility. We will refer to this middle-ground as *scenario (b)*. An example of such a scenario could include UAVs measuring pollution around a building with an assumption that measurements made closer to the building are more important. The characteristics of these three scenarios are depicted in the bottom row of figure 5.7 as histograms. The expected utilities of messages are normalized w.r.t. the message with the highest utility value and characterized with adjectives "low", "medium" and "high". Additionally, the empirical cumulative distribution function (eCDF) for this data set is plotted with a green line.

Simulation

Each scenario is implemented and evaluated in the network simulator *ns-3*. We have configured the simulator to be as close as possible to our robotic setup, i.e., using an ad-hoc IEEE 802.11b/g network with messages broadcast in a non-reliable manner using UDP. In all simulated scenarios there are 10 agents — this number is motivated by the practical applicability of systems of this size and similar to the number of robots we used for practical experiments. The longer the experiment, the more consistent the results. Hence, instead of performing multiple experiments and averaging results, for each set of parameters we run an experiment for a predefined time. We have empirically determined the mission duration of 100 s to provide consistent outcomes. The software is open source and available online to make the results reproducible [92].

Throughout the whole experiment, each agent generates with a frequency of 1 Hz a piece of information about itself, e.g., its battery level or position. The actual semantics of a message is not important, as we are pursuing an indirect approach and only consider the utility provided by an evaluation model. As the evaluation model, we use the approach described in chapter 4. Message utility functions are set to be linearly decreasing functions with variable slope and initial value. This allows us to specify for each message how long it generates utility, how much, and express the fact that the information could be overwritten by another message. They are not based on real-world information types. This definition allows us to introduce a variation between utilities of single messages and also to vary how messages influence each other.

The difference between three scenarios is introduced by splitting all messages into ten sets, numbered from 0 to 9. Then the utility

provided by the message from i-th set is multiplied by weight ω_i. For scenario (a), $\omega_i = 3^{i+1}$; for scenario (b), $\omega_i = 1000i + 1$; and for scenario (c), $\omega_i = i^{0.1} + 1$.

The MCTS is performing 1500 iterations for each decision, which in our implementation takes around 0.3 s using a personal laptop computer (Intel Core i7–7500U, 2.7 GHz). This value has been chosen to keep the duration of experiments manageable. Higher number of iterations provides better results, at the cost of longer decision time.

The results are presented in the top row of figure 5.7. The y axis, labeled *% of utility* refers to the ratio between achieved utility and maximum possible utility, i.e., obtained by exchanging all generated messages without delays. We compare our optimization method (red $\times$) with a randomized one (blue $+$). Each sample is a result of a single 100 s long experiment. For each algorithm we performed 30 experiment runs. In the case of a random method each experiment is run with a different probability of dropping a message. For our method the number of messages to send is constrained using equation (5.5) and also varies with different experiments. The plotted value is based on the received messages, so the loss naturally occurring in the network is also taken into account.

We observe that our method performs very well in comparison to a method that randomly drops messages if there are many low-value messages (scenario (c)). It means that our approach is able to identify which messages bring high utility and send them. On the other hand, in scenario (a), most of the messages bring high utility. It does not matter which ones are chosen, hence our algorithm cannot perform much better than the randomized method. Scenario (b) presents a middle-ground between the other two extremes.

Proof of concept using mobile robots

In contrast to deterministic simulations, in the real world the changing environment makes it very hard to reproduce similar experiments on robots in a reliable fashion. On the other hand, practical tests allow to confirm that the assumptions we made are correct and that there were no oversights. With that in mind, we tested our approach in practice using mobile robots in multiple different setups and with different platforms. For the sake of conciseness, we decided to present the robotic experiments that showcase all contributions presented in this thesis in chapter 7.

5.5 Summary

We presented a generic method to optimize information distribution that can be integrated into any robotic system that exchanges messages. The proposed method allows the agents to select and exchange the most relevant information, increasing the mission performance. In addition to the performance benefits, the proposed method is easy to integrate. The only step needed to benefit from it involves defining the utility functions for each used information type. The proposed method shows good performance in diverse scenarios and is demonstrated both in simulation and in experiments involving mobile robots.

We see multiple ways how this work could be developed in the future. First of all, there is certainly a need for a unified benchmark or a data set in order to make comparisons with other approaches quantifiable. The mission characteristics presented in this work could be the first step towards a more general solution. Furthermore, the comparisons with state of the art methods are relatively hard, because usually no code is provided and the evaluation is done using hand-crafted applications. Hence, a standardized set of experiments could facilitate the future work and make similar research more reproducible.

In addition, the properties of different information types can be investigated and exploited to develop efficient methods for information distribution. Ideally, such methods could be then incorporated into a generic system to increase its performance for that given information type (similarly to the approach presented in this paper for Markovian information types).

Automating the generation of utility specifications and estimators would minimize the user effort to integrate the proposed middleware into existing applications. One potential approach could be to utilize statistical or machine learning-based methods to autonomously determine utility specifications and obviate the need for any additional application-related modifications.

Finally, while performing the experiments for the work presented in this chapter, we have realized that commonly used formulations of network constraints are not very user-friendly. They require a lot of manual tuning and do not work well when the communication conditions are changing. In the next chapter we present a solution to this problem.

6 *Adapting to varying communication conditions*

Contents

While working on the middleware presented in chapter 5 we have realized that there is one important component in multi-robot information distribution that is not trivial, but often taken for granted. Intuitively, this component answers the question *how much* information can be exchanged between robots.

More specifically, in section 5.3 we define a *network goodput constraint* and show how the middleware could take it into account. We assumed that the available goodput (i.e., the amount of data that can be transferred during a specified time) is a resource, which amount could be easily estimated. Sometimes it is true. When a network performance is stable and no additional processes are using it, the available goodput could be measured and then used for the constraint definition. However, having to perform such procedure each time a system is changed is inconvenient. Furthermore, in MRS it is rarely the case that the network's performance would stay stable throughout the operation. Furthermore, other processes might be appearing in the network. For instance, a person operating the MRS could choose to turn an additional diagnostic software. A system that adapts to the varying network conditions would allow optimal network utilization and obviate the need for manual tuning.

The main contribution presented in this chapter is the design of the *adaptive goodput constraint* (section 6.2). It is suitable for long-term information distribution planning in MRS because its value is not changing unnecessarily and smoothly adjusts to varying communication conditions. We provide an extensive discussion of our approach's theoretical background to enable applying our findings to similar problems (section 6.1). The constraint is based on an observation that an uncoordinated nature of communication in MRS (for details see section 6.1) enables us to model and predict network behavior. Our approach is validated using a network simulator (ns-3) (section 6.3). Additionally, in chapter 7 the applicability of adaptive goodput constraint is demonstrated in practice using robots.

Parts of this chapter were taken from the work published in the

proceedings of the Wireless Days 2021 conference [4].

6.1 Background

The choice of communication method we focused on while pursuing this work is described in details in section 3.2. To summarize the most important aspects: the robots communicate using a wireless network, e.g., IEEE 802.11 WiFi. All messages are broadcast on a single channel using an existing collision avoidance technique, e.g., CSMA/CA. Robots operate in proximity, so multi-hop connections are not considered, though in principle could also be utilized. However, the method proposed in this chapter does not rely strongly on any characteristic of this particular setup. Therefore, it can also be adapted to other systems and communication techniques.

Communication constraints

Solutions similar to the *goodput constraint* introduced in chapter 5 are often used in the work on information distribution [24–26, 28][1]. Intuitively, such a constraint limits the amount of data to be transmitted by the application.

[1] Other publications also refer to it as a *throughput constraint*.

To formalize this, we use the same notation as in section 5.3. However, for the sake of completeness and we briefly recap the concepts also here. *Windowing function* $\mathcal{W}(t, M)$ specifies a subset of messages from a message set M to be considered at time t:

$$\mathcal{W}(t, M) = \{\, m \in M \mid t - T \leq m.t_{\text{sent}} \leq t \,\}, \tag{6.1}$$

where T is a parameter representing the window length and $m.t_{\text{sent}}$ represents the moment when message m was sent. Similarly to the previous chapters, we use the m.property notation to emphasize that a particular value is a property of message m. This windowing function allows us to choose a subset of messages sent within an interval of length T before time t.

Average achieved goodput in a window is expressed using the following operator:

$$\|\mathcal{W}(t, M)\|_G = \frac{1}{T} \sum_{m \in \mathcal{W}(t,M)} m.\text{size}, \tag{6.2}$$

which eventually leads us to the definition of *goodput constraint* as:

$$\forall t.\, \|\mathcal{W}(t, \mathcal{M})\|_G \leq \xi(t), \tag{6.3}$$

where m.size is the size of message m, $\mathcal{M}$ is a set of all messages exchanged during a mission, and $\xi(t)$ is the goodput value not to be

exceeded by the system. Often in other approaches [24, 25] ξ returns a constant value (representing network capacity) that has to be tuned according to the system in which the constraint is used.

Using this notation, we can formalize the problem addressed in this chapter as a problem of finding a value of ξ such that the network is optimally utilized (i.e., neither under- nor over-used) throughout the mission. In our approach, ξ is continuously adapted during the operation to account for nondeterministically varying network conditions.

Congestion control

Our solution is inspired by the approaches pursued in delay-based congestion control methods [65, 68], which arise from Kleinrock's theoretical analysis [93]. Kleinrock shows the existence of an optimal operating point for networked systems and characterizes it as a state when the data is being transmitted at the maximum rate while keeping buffers empty. Sending even the smallest amount of additional data would result in increased buffer utilization and thus increased latency. This observation indicates the importance of two values: latency of a network with empty buffers δ and maximum data rate ζ.

We reproduce this behavior in our simulation setup (described in detail in section 6.3). The messages of equal sizes are published with the specified rate r. However, their sending times are disturbed by a random value drawn from a normal distribution with standard deviation σ, i.e., the i-th message is sent at $\mathcal{N}(\frac{i}{r}, \sigma)$. For each publishing rate, we run simulations for 10 s and record two network performance metrics: the reception rate and latency of the received messages. The results are visualized in figure 6.1. The solid blue line (labeled $\sigma = 0.001$) represents a relationship between publishing (sending) rate and the network performance metrics in a scenario where messages' send times are almost evenly distributed. Hence, almost no collisions occur, and the network operates close to its minimum latency. If we were to define a constraint based on this relationship, we would require the publishing rate to be always lower than ζ (i.e., $\xi(t) = \zeta$) at which latency starts growing (bottom plot), or, equivalently, at which reception rate peaks (top plot)[2]. Unfortunately, it implies that observing ζ is only possible when the publishing rate exceeds ζ, which the constraint does not allow. Congestion control methods can circumvent this problem by rapidly controlling the amount of data in-flight and oscillating around the optimal point. Unfortunately, to be useful for long-term communication planning and information distribution, the constraint cannot change so often and in a nondeterministic manner. This suggests that state of the art

[2] In the BBR paper [68], these results are presented on a graph with the amount of data in flight on the horizontal axis instead of publishing rate because it allows capturing the fact that it takes some time to fill up network buffers. The observations presented here hold in both cases.

congestion-control techniques cannot be applied to our problem.

Figure 6.1: Relationship between publishing rate and reception rate (top), and latency (bottom) in our simulated setup. Each sample is the average value measured during a simulation.

Uncoordinated Communication in MRS

However, the reasoning presented in the previous paragraph makes one crucial assumption — the messages are sent one after another with enough time in between to avoid collisions. Specifically, in a non-congested network, when a new message enters a buffer, there should be no other message in that buffer. However, in MRS, messages are usually transmitted as soon as a sensor measurement or computation result is available. Hence, messages are not evenly distributed over time. Thus, multiple messages might enter a buffer

simultaneously. Furthermore, the higher the network load, the higher the chance for such an event to occur. Such communication patterns could also be characterized as an *uncoordinated communication*. This situation is demonstrated in figure 6.1 with the dashed red line ($\sigma = 0.01$) and the dotted green line ($\sigma = 0.1$). It shows that in a practical setting, average network latency starts growing much earlier than the event of congestion. This result suggests that, in a practical setting where scheduling is imperfect, a delay-based modification of goodput constraint might be conceivable.

Definitions

In this work, we have decided to utilize the following value motivated by TCP Vegas[3] in order to characterize network performance:

[3] In the TCP Vegas publication [65], a similar value is denoted as *Diff*.

$$\|W\|_\Delta = \frac{\|W\|_G \cdot \|W\|_L}{\hat{\delta}} - \|W\|_G, \tag{6.4}$$

where $\hat{\delta}$ is an estimation of δ. We assume a static network topology, so this value can be estimated as the minimum observed latency for all exchanged messages. Cardwell et al. [68] additionally update this value based on a moving time window in order to allow for changing network conditions. Operator $\|\cdot\|_L$ denotes the average latency of messages in window W, specifically:

$$\|W\|_L = \frac{1}{|W|} \sum_{m \in W} m.t_{\text{rcv}} - m.t_{\text{sent}}, \tag{6.5}$$

where $m.t_{\text{rcv}}$ is the reception time of message m and $|\cdot|$ denotes set cardinality.

The value of $\|W\|_\Delta$ is the difference between an optimistic data rate that would have been achieved in a given network if the latency was minimal and the actual data rate. Observe that increased message delays ($\|W\|_L$) cause the value of $\|W\|_\Delta$ to grow, which serves in our approach as an early indicator of congestion. Hence, this method is considered to be delay-based.

Brakmo and Peterson [65] give an intuitive interpretation of $\|W\|_\Delta$. They describe it as being proportional to the average backlog in the network, i.e., the average size of data kept in network buffers. This backlog is a direct reason for increased latency.

6.2 *Adaptive goodput constraint*

To simulate variable communication conditions, we introduce *background traffic* to the network, denoted by S. For instance, this value can be thought of as an amount of traffic present in the network from

sources that we cannot control. An example of background traffic might involve another application running in the same network and transmitting varying amounts of data or other processes starting in the system and exchanging some data. For instance, in a robotic system it could be a diagnostic application running in parallel to the main mission process and periodically gathering statistics from the system.

We start with a scenario where background traffic is constant (or non-existent). In such a simple case, there is no need to adapt to the network dynamically. It is sufficient to define ξ as a constant function and determine its optimal value. This value can be obtained by plotting graphs similar to those presented in figure 6.1 and manually choosing ξ. In practice, the choice almost certainly requires a compromise between message loss probability, average latency, and available bandwidth.

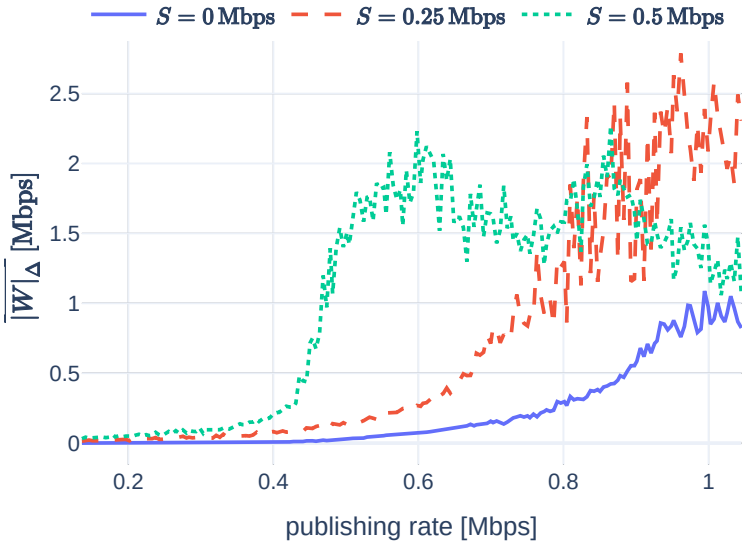


Figure 6.2: Results presented in the plot are obtained in the same way as the ones in figure 6.1 with σ fixed to 0.01 and added background traffic S. $\overline{\|W\|_\Delta}$ is an average of $\|W\|_\Delta$.

A more challenging scenario involves dynamic background traffic. Frequent changes of such traffic in a nondeterministic way depreciate the value of communication planning. Hence, we assume that background traffic changes are slow or infrequent, i.e., there is enough time for our method to adjust to the new network conditions.

The challenge in this scenario lies in the necessity to adjust to the available bandwidth dynamically. Additionally, the application performing information distribution might generate much less data than is needed to saturate the network, but the goodput constraint should still be valid for planning purposes. Therefore, a solution that just detects congestion and then lowers the publishing rate is not perfect. Instead, we should be able to estimate the optimal publishing

rate even when the network is far from congestion.

To explore this further, figure 6.2 presents the network's behavior depending on the amount of introduced background traffic. Intuitively, the more background traffic there is, the faster (for lower publishing rate) the network degrades.

We use $\|W\|_\Delta$ as an indicator of upcoming congestion, independent of the amount of background traffic. We introduce parameter α, and at each point in time t we search for a maximum publishing rate $\xi(t)$ that aims at maintaining the following constraint:

$$\forall t. \|\mathcal{W}(t, M)\|_\Delta \leq \alpha. \tag{6.6}$$

Property α has very similar characteristics to the choice of the value of ξ: lower values prioritize low latency, whereas higher values result in a higher publishing rate. For example, let us consider again a system presented in figure 6.2 and set $\alpha = 0.5\,\text{Mbps}$, i.e., specify that $\|\mathcal{W}(t, M)\|_\Delta$ should not exceed 0.5 Mbps. Then, we can observe in the plot that for the scenarios with $S = 0\,\text{Mbps}$, $S = 0.25\,\text{Mbps}$ and $S = 0.5\,\text{Mbps}$ background traffic, the application can publish with the maximum data rate of, respectively, around $\xi(t) = 0.9\,\text{Mbps}$, $\xi(t) = 0.65\,\text{Mbps}$, and $\xi(t) = 0.42\,\text{Mbps}$.

In practice, the background traffic is rarely known. Furthermore, the exact influence of the background traffic on $\|W\|_\Delta$ also depends on the publishing pattern used by the source of this traffic (e.g., if data packets are evenly distributed over time, or what kind of congestion control is being used by this source). Hence, the relationship between publishing rate and $\|W\|_\Delta$ is unknown and needs to be estimated. We denote this relationship as $R(\|W\|_\Delta) \approx \|W\|_G$ and refer to R as the *data rate model*.

We observed that when the network is not congested (i.e., for small values of $\|W\|_\Delta$), R can be reliably estimated using a linear function[4]. To obtain an estimator of R we utilize an online learning approach by constantly adjusting a linear model to the observed data.

We implemented this approach utilizing the Stochastic Gradient Descent (SGD) regression algorithm[5]. It has multiple features that are beneficial in our use case. It is easily applicable in an online setting with a low memory footprint and computational requirements independent of the number of messages or the window size. Hence, we can incorporate each received message into a model with very low, constant overhead and the model automatically adjusts to the varying communication conditions. Furthermore, it is possible to incorporate an initial belief into the model to perform well from the very beginning of the operation. Finally, the speed of adjustments to the changing background traffic can be controlled by tuning the algorithm's learning rate.

[4] Even though the plots suggest that an exponential or higher-degree polynomial relationship could fit the data better, in our experiments (especially the ones conducted on robots where the data was very noisy) the linear function served its purpose and is much easier to tune.

[5] Specifically, Stochastic Gradient Descent Regressor from scikit-learn [94].

To tune the SGD algorithm, multiple parameters have to be defined. The most important one is the learning rate. Low values result in a very stable model that is good for planning but is slow at adapting to new communication conditions. High values result in frequent changes and oscillations. We have also observed that using a quadratic loss function allowed us to adjust to new conditions much faster. It also results in outliers having a higher impact on the model, so this solution might not be applicable to networks with high jitter. Another important parameter that indirectly influences the model is the length T of the windowing function. Small values of T result in noisy data but at the same time allow for faster reaction to changing communication conditions.

Having the data rate model at our disposal allows us to define an adaptive $\xi(t)$ value:

$$\xi(t) \approx R_t(\alpha), \tag{6.7}$$

where R_t is the estimation of R obtained using SGD at time t. After substituting this equation into equation (6.3), we get the following *adaptive goodput constraint*:

$$\forall t. \, \|\mathcal{W}(t, \mathcal{M})\|_G \leq R_t(\alpha). \tag{6.8}$$

These results can be used in systems capable of limiting their publishing rate in order to enable them to dynamically adapt to varying communication conditions.

6.3 Results

The adaptive goodput constraint was implemented and applied in the information distribution middleware introduced in chapter 5. In principle, if the goal was just to present the adaptive goodput constraint, a much simpler method could be used: for instance, an application that generates messages with the rate determined by the constraint. However, by utilizing a method that incorporates long-term mission planning, we show that the constraint is suitable for such applications. In contrast, a constraint based on aggressive adaptation (e.g., directly implementing traditional congestion control methods) would change too often and thus would not be suitable. Our implementation is open-source and available online[6].

[6] https://github.com/zeroos/infdist/

Congestion avoidance

We start by showing that our method is indeed able to avoid congestion. For this purpose, we perform a simulation-based study, and we show that the publishing rate chosen by our approach is reasonable,

i.e., maintains a balance between low latency (which implies no congestion) and high publishing rate. The simulations were conducted using ns-3. Each simulation lasts for 100 s. Eight agents are *generating* messages of constant size with a frequency of 6 Hz. The setup is analogous to the one presented in section 6.1 but with parameter $\sigma = 0.01$.

For each generated message, the information distribution scheme chooses if it should be sent or not based on the introduced adaptive goodput constraint. If the decision is positive, the message is broadcast to all other agents using the UDP protocol. The network was configured as an 802.11g ad-hoc WiFi with 1 Mbps data rate. Such a low data rate was used to facilitate simulation-based experiments. Experiments with high data rates are presented in section 7.3 on robots. Additional 0.5 Mbps of background traffic were added to the network to show that the method performs well even when a significant portion of the network traffic is not part of the information distribution process and cannot be influenced. Whenever windowing functions are used, we set $T = 2.5$ s.

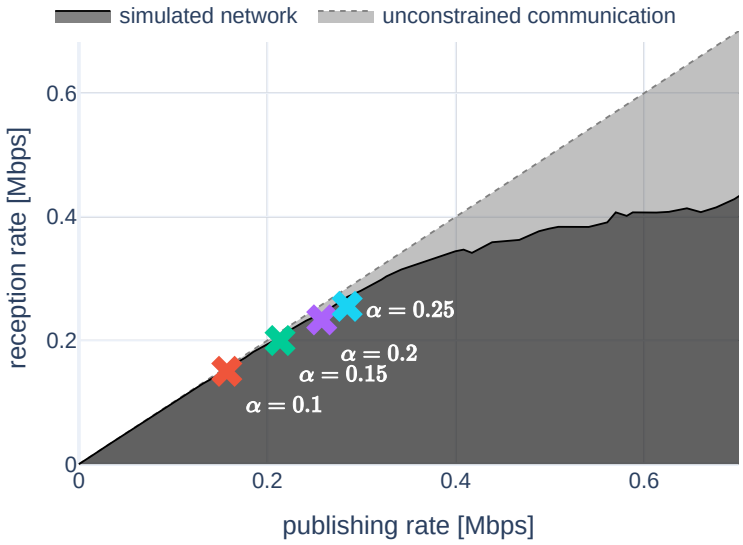


Figure 6.3: Relationship between parameter α and the publishing rate compared to the reception rate. The light-gray dotted line represents an ideal scenario where all sent messages, independent of the amount of data, are successfully received. The dark-gray solid line represents simulations that send messages with a predefined publishing rate, with no adaptation.

The simulation results are presented in figures 6.3 and 6.4. Each data point marked with $\times$ corresponds to a single run of the simulation performed using our method with a specified parameter α. The simulations show that higher values of α result in a higher drop rate (figure 6.3) and increased latency (figure 6.4) but allow higher publishing rates. These results show that our approach can select an effective publishing rate, and the selection can be tuned using parameter α. It is worth noting that all of the values of α annotated in the figures could serve as a valid choice depending on the application

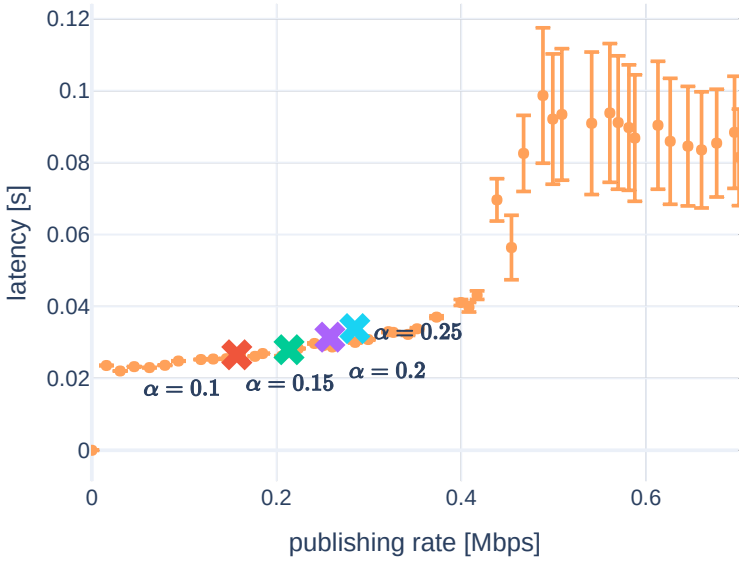


Figure 6.4: Relationship between parameter α and the publishing rate compared to communication latency.

at hand. Values higher than those presented would result in an even higher loss probability, which is usually undesirable.

Adapting to varying communication conditions

This section presents a qualitative case study highlighting how our approach adapts to varying communication conditions. For this purpose, we utilize the network setup identical to the one used in the previous experiment, but this time with only four agents. Instead of publishing messages all the time at the same rate, we introduce some periods of inactivity for each agent. This results in the periodic traffic pattern visible in the first 25 s of the *application goodput* plot in figure 6.5, marked with the green dashed line. The same pattern repeats four times. For instance, this kind of behavior could be observed in practice when an agent detects some object of interest, which requires agents to exchange more messages, followed by a period of lower activity until the next object is found. More importantly, in this work, such an example allows us to demonstrate the problems that the adaptive goodput constraint faces. Additionally, during the first half of the experiment, 0.1 Mbps of background traffic is introduced and increased to 0.25 Mbps in the second half. During the experiments presented in this section, we set $\alpha = 0.1$, which prioritizes low message loss.

The top plot of figure 6.5 shows how our adaptive constraint smoothly adapts to the new conditions. This behavior is the most prominent at 50 s when the background traffic increases. Another attractive property of our method is that it adjusts to the new con-

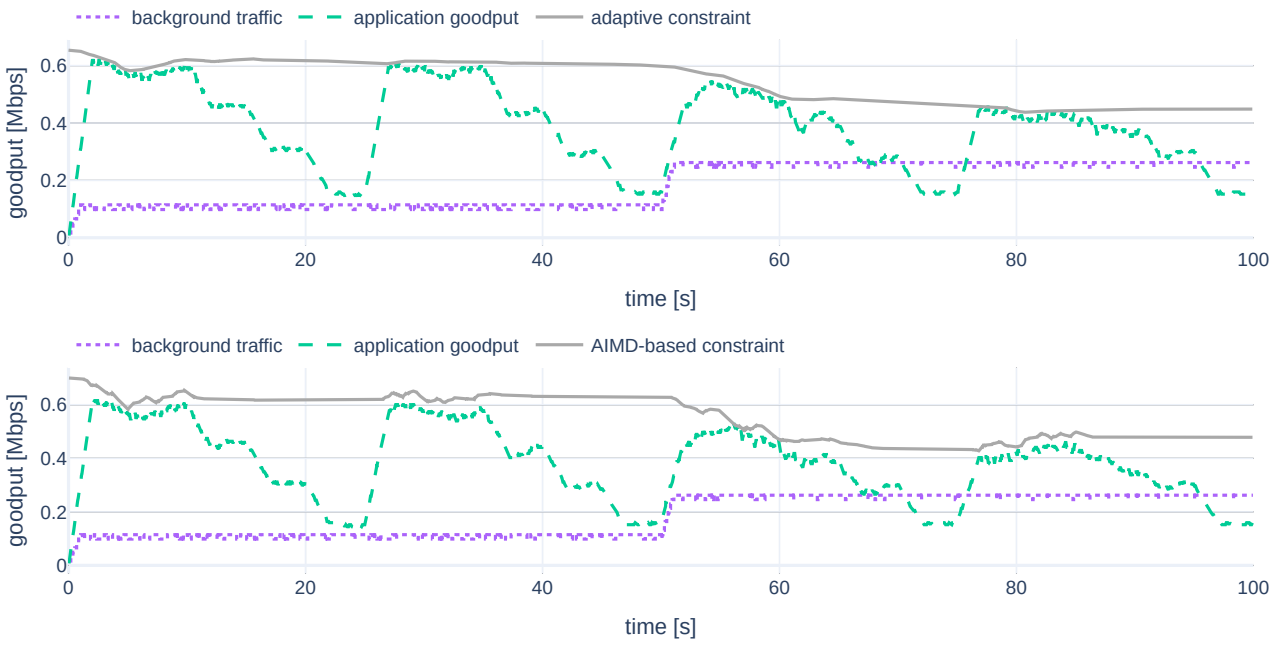


Figure 6.5: A simulated mission showing how the constraint changes over time with varying application goodput and background traffic. The top graph presents the results obtained with the adaptive goodput constraint and the bottom one with a constraint based on additive-increase/multiplicative-decrease (AIMD) method. It shows that a constraint based on AIMD is not suitable for long-term planning.

ditions even when the application goodput is much lower than the constraint. It is visible during the time segment between 65 s and 75 s, when the constraint slowly decreases even though the application traffic is much lower than the constraint.

Furthermore, at around 5 s and 30 s, it can be observed that the constraint is slightly disturbed. This happens because the message publishing times are randomized. However, such incidents do not considerably affect the constraint.

To compare our method, we have constructed a constraint based on the additive-increase/multiplicative-decrease (AIMD) method [95]. Similar methods are widely used, for example, in TCP congestion control. We are aware that the role of congestion control in TCP has different objectives than in communication planning. Nevertheless, we found the AIMD method to be the closest candidate for comparison.

Whenever a new message is received, the current AIMD constraint $\xi_{\text{AIMD}}(t)$ is updated according to algorithm 4. If the $\|W\|_{\Delta}$ is higher than threshold α, the constraint is decreased by multiplying it by a parameter $b \in (0,1)$. Otherwise, if the constraint is close to the current application goodput (not further than the parameter β), the constraint is increased by $a \in (0,\infty)$. If both of the previous statements are false, the constraint remains unchanged.

In our experiments, we set $a = 2048$ bytes, $b = 0.998$, and

Algorithm 4: AIMD algorithm used as a baseline for our approach. $\xi_{\text{AIMD}}(t)$ is the new value of the constraint and $\xi_{\text{AIMD}}(t^-)$ is the old value.

if $\|W\|_\Delta > \alpha$ **then**
    $\xi_{\text{AIMD}}(t) = \xi_{\text{AIMD}}(t^-) \cdot b$
else if $\xi_{\text{AIMD}}(t^-) - \|W\|_G < \beta$ **then**
    $\xi_{\text{AIMD}}(t) = \xi_{\text{AIMD}}(t^-) + a$
else
    $\xi_{\text{AIMD}}(t) = \xi_{\text{AIMD}}(t^-)$

$\beta = 0.05\,\text{Mbps}$ to keep the dynamics of changes similar to the adaptive constraint. Often AIMD-based methods set the b parameter to lower values to more aggressively adapt to changes. However, in our application that would make the constraint even less usable for planning due to rapid changes[7].

[7] We validated this by simulating several more aggressive AIMD-based methods (lower b values).

The bottom plot of figure 6.5 presents the results of the same experiment as the top one, but this time ran with the AIMD-based constraint. Perhaps the most notable difference is that the AIMD-based algorithm is more susceptible to noise and its behavior is less predictable under noisy conditions. It is not just a matter of tuning — the main reason is that the AIMD-based algorithm is based on a binary threshold. In contrast, our method adjusts more gradually to observations that are close to the modeled state and more dynamically to observations that do not match the current model. Another difference is visible when the application goodput is much lower than the constraint. Then, the value is not changing and is determined by the last moment when the application goodput was close to the constraint. This situation can be observed on the plot around 90 s. We can see that the constraint is set to a higher value than it was at 70 s, but the communication conditions did not change. This is especially unsuitable for communication planning, as the constraint might end up being an arbitrary value during periods with less communication.

6.4 Summary

The adaptive goodput constraint introduced in this work could serve as a method to incorporate adaptation into any robotic application capable of limiting its maximum transmission rate. We base the solution on a model of network performance in MRS exploiting the uncoordinated nature of communication in MRS. The model is continuously being fit (online machine learning) to the observed data, which enables its adaptive behavior.

So far, the theoretical principles of this method have been laid down and it was presented that the constraint works in simulation. In the next chapter, we present a robotic proof of concept we have used to show the practical applicability of the introduced constraint.

Naturally, there is always room for improvement. For future work, a more sophisticated data rate model could be beneficial. In particular, a higher-degree polynomial could closer fit the data. Such an approach would allow to faster adjust the constraint when the network is only partially used. Additionally, the ability to forecast the change of communication conditions and then adjust the constraint more rapidly would be handy for communication planning. Specifically, having multiple data rate models trained and switching between them could be beneficial for many applications.

7 Real-world applications: Adaptive image streaming

To validate our assumptions and show that the proposed solution is working in practice, we implemented it on two robotic platforms and used them for experimental work. In this chapter, we will describe how the experiments are set up, their results, and how the information distribution middleware was incorporated into the KPK demonstrator.

Contents

7.1 Experimental setup

The experiments were prepared using two robotic platforms. The first one consists of eight *Pololu Balboa* robots (described in detail in section 3.1, also shown in figure 7.1). The second one included four TwinFOLD Science UAVs (see section 3.1 for details). We decided to implement the demonstrator on the ground robots first, because they are much easier and safer to use. Then, we moved the working prototype to UAVs.

Figure 7.1: The Balboa robots used for some of the experiments.

Both of these platforms are based on Raspberry Pi (version 3B+ for the Balboa robots and version 4B for UAVs), which facilitated the migration of software from one platform to another. As cameras, we used *Raspberry Pi Camera Module V2*, because they integrate well with the computing board. The Raspberry Pi is set up to work in an IEEE 802.11 b/g ad-hoc network with multi-hop connection support provided by the *Babeld* protocol. The software is built in *Python* on top of the *Robot Operating System* 2 (ROS 2) robotic framework with a default *eProsima Fast RTPS* implementation of the *Data Distribution Service* (DDS) standard. Our initial experiments (described in section 7.2) were conducted with ROS 2 Eloquent Elusor. Then, when Foxy Fitzroy was realesed, we modified our software to work with that version. The robots' real-time clocks are synchronized using *Chrony* with a root mean square (RMS) clock discrepancy offset below 0.1 ms.

By using an ad-hoc network together with ROS 2 we create a fully distributed system, i.e., without any single point of failure enabling robots to join and leave the mission freely.

7.2 *Optimization middleware*

The aim of developing the first demonstrator is to provide the proof of concept to show that the optimization of information distribution (described in chapter 5) is applicable in practice on affordable hardware and to qualitatively confirm the results of the simulation-based study. We also utilize this opportunity to present how the layered design of our method enables fast and easy integration into existing systems.

For that, we designed an application, which simulates an exploration mission. The robots move around a single 70 m^2 room and send a stream of 640×480 px images from their cameras to the operator's computer. The aim of the system is to monitor red objects. This is a simplification made in order to clearly present our goals, but a similar system could additionally classify objects and look for some particular classes, e.g., cars. Alternatively, an infra-red camera could be used. Then, the red color could suggest presence of people or animals.

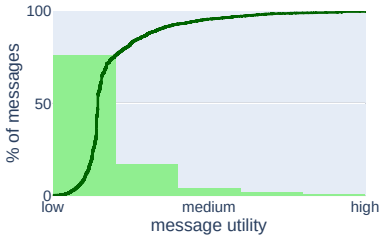


Figure 7.2: Characteristics of the proof of concept, which show that a random algorithm would not perform well in this setting as most of the messages are of low utility. For more information about mission characteristics, see section 5.4.

Figure 7.2 presents the characteristics of this mission obtained using data containing all generated messages from a single experiment and processed in the same way as in section 5.4. We present it mainly to demonstrate that the introduced concept of characteristics can be used to analyze practical missions, but also to show that even in a simple application an algorithm randomly dropping messages would not perform well.

As a baseline, the system has been implemented purely in ROS 2 without the use of our information distribution middleware. The performance was mediocre. When pictures were sent rarely enough not to overflow the network, the refresh rate was low (around 1.5 Hz). Setting a higher refresh rate resulted in a network congestion. Even though we used a communication scheme without retransmissions (non-reliable setting of DDS), delays of image frames increased above 3 s and the variation of frame rates was high. This resulted in a situation where it was possible to completely miss a red object, even when it was positioned in front of a robot for a couple of seconds.

Then, we have incorporated our method in three simple steps: intercepting image messages sent by the original system, defining utility functions for them to be dependent on the amount of red pixels in the image, and specifying that an image is expected to have the same number of red pixels as the previous one. Specifically, we have used the following utility function:

$$\mathcal{U}_{\mathrm{m}}(m,t) = \begin{cases} 0 & \text{if } t < m.t_{\mathrm{rcv}} \\ \mathcal{U}_{\mathrm{vid}}(m,t)(1+\alpha\, m.\mathrm{red}) & \text{otherwise} \end{cases}, \qquad (7.1)$$

Figure 7.3: Screenshot of the demonstrator in which the robots are exchanging images and our method maintains a limited throughput constraint while giving preference to images containing more red pixels.

where m is a message, t is a moment at which we want to evaluate the utility function, $m.t_{\text{rcv}}$ is the message reception time, and $m.\text{red}$ is a percentage of red pixels in the transmitted image. $\mathcal{U}_{\text{vid}}$ is introduced in equation (4.28) and is a utility of a message that ensures a good quality of experience taking into account human perception. It assumes that doubling the frame rate of an image stream improves the experience of a person observing it by a constant factor. Low values of parameter α prioritize frequent updates from each robot, whereas high values prioritize the images with red pixels. In the experiments $\alpha = 10$, which we determined empirically to provide a good viewing experience. The number of MCTS simulations needed to be tuned and set to 300, which resulted in a reasonable balance between low latency and good system performance.

These modifications greatly increased the system usefulness. A screenshot of a running application is presented in figure 7.3. It can be seen that a robot seeing a red fire extinguisher transmits images the most frequently. The robots that see some red pixels (e.g., wheels of other robots) are also sending information more often than other ones, but overall the link is shared fairly. As soon as a robot sees a red object, the frame rate of its video stream increases. We cannot show this in a picture, but it can be observed in video 7.1 presenting the demonstrator. Note that this behavior emerges autonomously, no additional logic was programmed.

On the downside, our method increases latency of the system because of the added computational overhead. This overhead depends mainly on the number of simulations performed by MCTS. In the presented scenario, with MCTS performing 300 simulations, the la-

Video 7.1: The video shows how the information distribution middleware performs in a robotic demonstrator. Eight Balboa robots are exploring the environment and search for red objects. It is available on-line at https://youtu.be/p7Td6KUIksk.

tency caused by the optimization method was around 0.1 s for each message. Before the introduction of the middleware, when the network was not overloaded, images were received approximately 0.2 s after being generated. Afterward, the latency grew to approximately 0.3 s. For the presented scenario, this is not a big issue: if one is looking for a red object, it is better to see it with a delay than to miss it completely. Furthermore, the proof of concept was not programmed with performance as the main priority and makes use of inefficient technologies, e.g., the Python programming language.

Finally, the goal of the demonstrator is not to show this particular behavior. After all, a similar application could be realized in a more efficient way if it was designed specifically for this use case. We show that our method is applicable in practice and can easily transform a relatively simple application into a much more sophisticated one. Furthermore, the simulation-based study shows that the method provides similar benefits to more complex systems, for which designing a specific information distribution optimization scheme might be infeasible.

7.3 *Adaptation to varying communication conditions*

Next, to show that the adaptive goodput constraint introduced in chapter 6 works, we implemented it and added it to the demonstrator described in the previous section. This demonstrator serves as an excellent candidate to show the applicability of the constraint introduced in chapter 6 for at least two reasons: (1) it involves exchanging significant amounts of data, which might lead to a network congestion, and (2) it can maintain a goodput constraint (similarly

to most approaches to information distribution). In contrast to the simulation-based evaluation (section 6.3), in this setup, there is no need to disturb the publishing time by a random value. The noise naturally occurring in the network is high enough to enable observing the behavior described in section 6.1.

The only modification of the demonstrator involved using the adaptive goodput constraint instead of the previously utilized goodput constraint. As the constraint operates on the application layer (i.e., it measures the latencies observed by the application, including processing time introduced by the application), it had to be tuned for this specific application. Specifically, parameters α and SGD learning rate needed to be adjusted. In comparison to the simulations, we have used a lower value of $T = 0.5\,\mathrm{s}$. This change was done to match the window length of the constraint with the window length of the information distribution method used on the robots. The information distribution scheme used in this demonstrator continuously adjusts the number of exchanged messages to meet the adaptive goodput constraint.

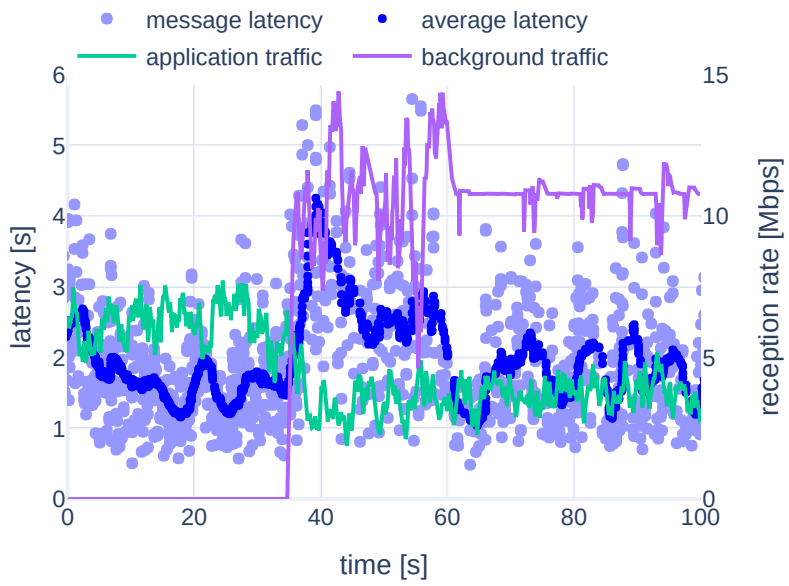


Figure 7.4: Reception rate and latency measurements during the experiment using robots with background traffic added at around 35 s. The average latency graph presents a weighted average of message latency within a window of 3 s.

To show that the adaptive goodput constraint works, we have introduced 10 Mbps of background traffic at an arbitrary point in time during the operation of the demonstrator. The traffic was generated using *iperf*[1]. The server reported successfully receiving an average of 10.5 Mbps.

[1] https://iperf.fr/

The system starts from a stable state with almost no background traffic. Just after the additional background traffic was introduced, the preview of images from the cameras froze and became responsive

again after a couple of seconds. This situation was captured using *Wireshark*[2] running on the operator's computer, and the obtained measurements are visualized in figure 7.4. Note an immediate drop in the application traffic (green line) just after the additional background traffic (purple line) was introduced. It occurred because the network could not transmit all the data. Then, over the next 20 s the latency (blue markers) is decreasing and finally stabilizing, which shows that the constraint is successfully adapting to the new conditions and the publishing rate of the application is lowered.

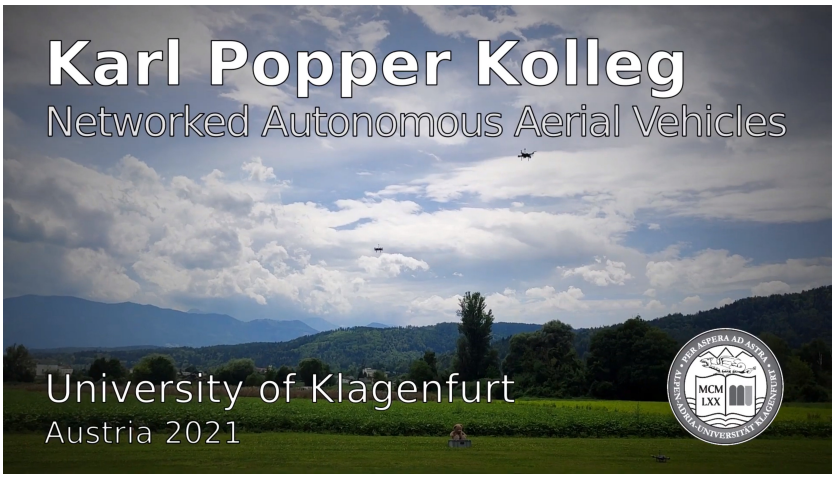
[2] https://www.wireshark.org/

7.4 KPK demonstrator

Finally, the contributions of this thesis were presented in the context of the KPK project (see section 1.1). To briefly recapitulate, the aim of the project was to create an autonomous system being able to construct a 3D reconstruction of an unknown environment using a group of UAVs. The UAVs are exploring some predefined perimeter. When they identify a new object, they send images from different perspective of this object to the base station, where the 3D reconstruction is performed. The demonstrator is presented in video 7.2.

The work described in this thesis allows the images captured by drones to be transmitted to the base station without overloading the network. We have incorporated it into the demonstrator in the following way. During the operation, drones are all the time sending the images using the information distribution middleware. Thanks to the adaptive goodput constraint, the network is shared fairly and utilized efficiently.

To maximize the number of images of the reconstructed objects,



Video 7.2: The video presents the final KPK demonstrator with four TwinFOLD Science UAVs performing a 3D reconstruction mission. It is available on-line at https://youtu.be/fw6bQhUfIsk.

the utility of images containing these objects was increased. We use a slightly modified version of the message utility function defined in equation (7.1). We treat the images containing the object to be reconstructed in the same way as we would treat an image that is fully red according to the previous definition:

$$\mathcal{U}_{\mathrm{m}}(m,t) = \begin{cases} 0 & \text{if } t < m.t_{\mathrm{rcv}} \\ \mathcal{U}_{\mathrm{vid}}(m,t)(1+\alpha) & \text{if } m \text{ contains the reconstructed object} \\ \mathcal{U}_{\mathrm{vid}}(m,t) & \text{otherwise} \end{cases}. \tag{7.2}$$

Figure 7.5: A still frame from video 7.2 showing how the middleware was utilized during the KPK demonstrator. Images taken by cameras mounted on UAVs are constantly streamed to the base station. The images of the object to reconstruct are prioritized. On the operator's screen, a red border is drawn around images with high utility.

The result is presented in figure 7.5. It shows that the images from the first drone, which do not contain the object to be reconstructed, are streamed with lower frequency than the images from the other three drones. In the background we can see the three drones rotating around the photographed object below them.

8 Conclusions and outlook

This thesis tackled a problem of information distribution in multi-robot system. It introduced a new way of approaching the problem with three significant contributions. First of all, we described an evaluation model that allows to characterize the application at hand and define which information is important in this context. Then, we proposed a generic middleware that uses the aforementioned evaluation model and could be transparently incorporated into existing systems to improve the communication performance. Finally, an adaptive goodput constraint allows for even easier deployment of the middleware on real robots.

Contents

8.1 Conclusions

The main focus of this thesis is on making the state of the art of information distribution in multi-robot systems applicable in practice in robotic systems. Thus, we focused on generalization and abstraction of the existing solutions and practical experiments. We believe that the contributions described in this thesis allow to bridge the gap between theoretical research and real applications and allow the engineers to benefit from the state of the art methods for intelligent information distribution.

Our research enables robots to autonomously decide how to share information in a multi-robot system. The introduced system is able to maintain various constraints, for instance the one that limits the goodput used by the system. This allows the robots to work without overloading the network and facilitates scaling up of the system.

All the software developed as a part of the work described in this thesis was tested on robots, applied in practice and is available online on a permissive license for others to use.

8.2 Outlook

The main goals set for this thesis were successfully accomplished. We created an evaluation method and based on it proposed an optimiza-

tion middleware. However, there is always room for improvement. We envision multiple main tracks of continuation of our work.

First of all, a framework for assessing the performance of information distribution methods is needed. Currently, the state of the art approaches are usually evaluated for a specific use case. We initially addressed this problem by introducing mission characteristics (see section 5.4), but a more comprehensive solution would allow for direct comparison between methods. For instance, a dataset focused on information distribution could be proposed. Such a dataset would contain a collection of typical robotic applications and a way to evaluate the communication in them.

Next, perhaps the main disadvantage of the optimization method introduced in chapter 5 is related to the computational complexity of each decision. In fact, to get a good performance in a medium-sized MRS, the method introduces a noticeable latency. Of course, in general this problem cannot be solved efficiently, however its performance could be improved for specific information types or use cases. The reductions described in section 5.3 already present such an approach. However, it can still be extended for different information types. For instance, for an information type in which all messages come from a finite set, a policy could be defined that allows for constant-time decisions.

Further, the idea behind the adaptive goodput constraint introduced in chapter 6 is also worth extending. Different machine-learning methods could be used to estimate available goodput and anticipate the changes. For instance, a model that takes into account periodic transmissions taking place in the network could learn to reduce the constraint (i.e., lower the amount of information the system can exchange) during that transmission and raise it back to the previous level just after the transmission is finished.

But perhaps the way to make the most impact with the work of this thesis would be to implement the middleware as a part of existing communication solutions. For instance, incorporating it into the data distribution service in ROS 2 could enable the developers to take advantage of such a system with no changes to the source code.

List of Symbols

$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathcal{P}(S)$	power set of S
m	message
$\mathcal{M}$	set of all messages
M	subset of messages
g	agent
G	set of agents
M_{gen}	set of generated messages
M_{sent}	set of sent messages
M_{rcv}	set of received messages
$m.t_{\text{gen}}$	generation time of message m
$m.t_{\text{sent}}$	send time of message m
$m.t_{\text{rcv}}$	reception time of message m
$m.\text{size}$	size of message m
p	message type
$\mathcal{P}$	set of message types
ω	weight
$\mathcal{U}$	utility
$\hat{\mathcal{U}}$	agent's utility
h	mission history of an agent
$\mathcal{A}$	aggregation function
t	time
T	time period or time slot
f	frequency
t_{end}	mission end time
$\hat{t}$	relative time (e.g., relative to the moment when a message was generated)
s	state
S	set of all states or the amount of background traffic
A	area
ρ	battery depletion rate
o	objective
O	set of objectives
O_{pursued}	set of pursued objectives
B	base of a logarithm

A	available throughput
δ	latency in the network with empty buffers
$\hat{\delta}$	estimation of δ
ζ	data rate in the network with empty buffers
α	rate difference threshold for the adaptive goodput constraint
v	MCTS node
W	window of messages
$\mathcal{W}$	windowing function
$\|W\|_G$	goodput of window W
$\|W\|_\Delta$	rate difference of window W
$\|W\|_L$	average latency of window W
ξ	goodput limit
R	data rate model

List of Figures

Bibliography

[1] M. Barciś and H. Hellwagner, "An Evaluation Model for Information Distribution in Multi-Robot Systems," in *IEEE INFOCOM WKSHPS: MiSARN: Mission-Oriented Wireless Sensor, UAV and Robot Networking*, 2019. DOI: 10.1109/INFCOMW.2019.8845299.

[2] M. Barciś, A. Barciś, and H. Hellwagner, "Information Distribution in Multi-Robot Systems: Utility-Based Evaluation Model," *Sensors*, vol. 20, no. 3, p. 710, 2020. DOI: 10.3390/s20030710.

[3] M. Barciś, A. Barciś, Nikolaos Tsiogkas, and H. Hellwagner, "Information Distribution in Multi-Robot Systems: Generic, Utility-Aware Optimization Middleware," *Front. Robot. AI*, vol. 8, p. 207, 2021. DOI: 10.3389/frobt.2021.685105.

[4] M. Barciś and H. Hellwagner, "Information Distribution in Multi-Robot Systems: Adapting to Varying Communication Conditions," in *Proc. Wireless Days Conference*, 2021. DOI: 10.1109/WD52248.2021.9508324.

[5] R. Mirollo and S. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM J. Appl. Math.*, vol. 50, no. 6, pp. 1645–1662, 1990. DOI: 10.1137/0150098.

[6] A. Barciś, M. Barciś, and C. Bettstetter, "Robots that sync and swarm: A proof of concept in ROS 2," in *Proc. IEEE Int'l Symp. on Multi-Robot and Multi-Agent Systems (MRS)*, 2019. DOI: 10.1109/MRS.2019.8901095.

[7] P. Mazdin, M. Barciś, H. Hellwagner, and B. Rinner, "Distributed Task Assignment in Multi-Robot Systems based on Information Utility," in *Proc. 16th Int'l Conf. on Automation Science and Engineering (CASE)*, 2020, pp. 734–740. DOI: 10.1109/CASE48305.2020.9216982.

[8] T. Andre, K. A. Hummel, A. P. Schoellig, E. Yanmaz, M. Asadpour, C. Bettstetter, P. Grippa, H. Hellwagner, S. Sand, and S. Zhang, "Application-driven design of aerial communication networks," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 129–137, 2014. DOI: 10.1109/MCOM.2014.6815903.

[9] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on Unmanned Aerial Vehicle Networks for Civil Applications: A Communications Viewpoint," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2624–2661, 2016. DOI: 10.1109/COMST.2016.2560343.

[10] M. Erdelj, M. Król, and E. Natalizio, "Wireless Sensor Networks and Multi-UAV systems for natural disaster management," *Computer Networks*, vol. 124, pp. 72–86, 2017. DOI: 10.1016/j.comnet.2017.05.021.

[11] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, "Drone networks: Communications, coordination, and sensing," *Ad Hoc Networks*, Advances in Wireless Communication and Networking for Cooperating Autonomous Systems, vol. 68, pp. 1–15, 2018. DOI: 10.1016/j.adhoc.2017.09.001.

[12] S. Hayat, E. Yanmaz, and C. Bettstetter, "Experimental analysis of multipoint-to-point UAV communications with IEEE 802.11n and 802.11ac," in *Proc. IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015, pp. 1991–1996. DOI: 10.1109/PIMRC.2015.7343625.

[13] E. Yanmaz, S. Hayat, J. Scherer, and C. Bettstetter, "Experimental performance analysis of two-hop aerial 802.11 networks," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2014, pp. 3118–3123. DOI: 10.1109/WCNC.2014.6953010.

[14] S. Hayat, C. Bettstetter, A. Fakhreddine, R. Muzaffar, and D. Emini, "An Experimental Evaluation of LTE-A Throughput for Drones," in *Proc. Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, ser. DroNet'19, Association for Computing Machinery, 2019, pp. 3–8. DOI: 10.1145/3325421.3329765.

[15] A. Fakhreddine, C. Bettstetter, S. Hayat, R. Muzaffar, and D. Emini, "Handover Challenges for Cellular-Connected Drones," in *Proc. Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, ser. DroNet'19, Association for Computing Machinery, 2019, pp. 9–14. DOI: 10.1145/3325421.3329770.

[16] L. Shi, N. J. H. Marcano, and R. H. Jacobsen, "A review on communication protocols for autonomous unmanned aerial vehicles for inspection application," *Microprocessors and Microsystems*, p. 104 340, 2021. DOI: 10.1016/j.micpro.2021.104340.

[17] J. Anderson and G. A. Hollinger, "Communication Planning for Cooperative Terrain-Based Underwater Localization," *Sensors*, vol. 21, no. 5, p. 1675, 5 2021. DOI: 10.3390/s21051675.

[18] B. Allotta, R. Costanzi, A. Ridolfi, C. Colombo, F. Bellavia, M. Fanfani, F. Pazzaglia, O. Salvetti, D. Moroni, M. A. Pascali, M. Reggiannini, M. Kruusmaa, T. Salumäe, G. Frost, N. Tsiogkas, D. M. Lane, M. Cocco, L. Gualdesi, D. Roig, H. T. Gündogdu, E. I. Tekdemir, M. I. C. Dede, S. Baines, F. Agneto, P. Selvaggio, S. Tusa, S. Zangara, U. Dresen, P. Lätti, T. Saar, and W. Daviddi, "The ARROWS project: Adapting and developing robotics technologies for underwater archaeology," *IFAC-PapersOnLine*, vol. 48, no. 2, pp. 194–199, 2015. DOI: 10.1016/j.ifacol.2015.06.032.

[19] F. Yazdani, S. Blumenthal, N. Huebel, A. K. Bozcuoğlu, M. Beetz, and H. Bruyninckx, "Query-based integration of heterogeneous knowledge bases for search and rescue tasks," *Robotics and Autonomous Systems*, vol. 117, pp. 80–91, 2019. DOI: 10.1016/j.robot.2019.03.013.

[20] S. Hoshino, H. Seki, and Y. Naka, "Development of a Flexible and Agile Multi-robot Manufacturing System," in *Proc. Int'l Federation of Automatic Control World Congress*, vol. 41, 2008, pp. 15 786–15 791. DOI: 10.3182/20080706-5-KR-1001.02669.

[21] T. Zhivkov, E. Schneider, and E. Sklar, "MRComm: Multi-Robot Communication Testbed," in *Towards Autonomous Robotic Systems*, K. Althoefer, J. Konstantinova, and K. Zhang, Eds., ser. Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 346–357. DOI: 10.1007/978-3-030-25332-5_30.

[22] F. Amigoni, J. Banfi, N. Basilico, I. Rekleitis, and A. Quattrini Li, "Online Update of Communication Maps for Exploring Multirobot Systems Under Connectivity Constraints," in *Distributed Autonomous Robotic Systems*, N. Correll, M. Schwager, and M. Otte, Eds., ser. Springer Proceedings in Advanced Robotics, vol. 9, Springer, 2019, pp. 513–526. DOI: 10.1007/978-3-030-05816-6_36.

[23] J. Anderson and G. Hollinger, "Communication Planning for Cooperative Terrain-Based Underwater Localization," in *Proc. Robots in the Wild: Challenges in Deploying Robust Autonomy for Robotic Exploration RSS Workshop*, 2020. DOI: 10.3390/s21051675.

[24] R. J. Marcotte, X. Wang, D. Mehta, and E. Olson, "Optimizing multi-robot communication under bandwidth constraints," *Autonomous Robots*, vol. 44, pp. 43–55, 2019. DOI: 10.1007/s10514-019-09849-0.

[25] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-MCTS: Decentralized planning for multi-robot active perception," *The Int'l J. of Robotics Research*, vol. 38, no. 2-3, pp. 316–337, 2019. DOI: 10.1177/0278364918755924.

[26] M. C. Fowler, T. C. Clancy, and R. K. Williams, "Intelligent Knowledge Distribution: Constrained-Action POMDPs for Resource-Aware Multiagent Communication," *IEEE Transactions on Cybernetics*, pp. 1–14, 2020. DOI: 10.1109/TCYB.2020.3009016.

[27] N. Tsiogkas and D. M. Lane, "Towards an Online approach for Knowledge Communication Planning: Extended Abstract," in *Proc. Int'l Symp. on Multi-Robot and Multi-Agent Systems (MRS)*, 2019. DOI: 10.1109/MRS.2019.8901086.

[28] V. V. Unhelkar and J. A. Shah, "ConTaCT: Deciding to Communicate during Time-Critical Collaborative Tasks in Unknown, Deterministic Domains," in *Proc. Conf. on Artificial Intelligence (AAAI)*, 2016. DOI: 10.5555/3016100.3016256.

[29] M. Roth, R. Simmons, and M. Veloso, "What to Communicate? execution-Time Decision in Multi-agent POMDPs," in *Proc. Distributed Autonomous Robotic Systems (DARS)*, 2006. DOI: 10.1007/4-431-35881-1_18.

[30] S. A. Williamson, E. H. Gerding, and N. R. Jennings, "Reward Shaping for Valuing Communications During Multi-agent Coordination," in *Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009. DOI: 10.1145/1558013.1558102.

[31] A. Kassir, R. Fitch, and S. Sukkarieh, "Communication-aware Information Gathering with Dynamic Information Flow," *Int'l Journal of Robotic Research*, vol. 34, no. 2, pp. 173–200, 2015. DOI: 10.1177/0278364914556911.

[32] R. Becker, A. Carlin, V. Lesser, and S. Zilberstein, "Analyzing Myopic Approaches for Multi-Agent Communication," *Computational Intelligence*, vol. 25, no. 1, pp. 31–50, 2009. DOI: 10.1111/j.1467-8640.2008.01329.x.

[33] D. Szer and F. Charpillet, "Improving coordination with communication in multi-agent reinforcement learning," in *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, 2004, pp. 436–440. DOI: 10.1109/ICTAI.2004.74.

[34] M. Gadd and P. Newman, "Checkout my map: Version control for fleetwide visual localisation," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2016. DOI: 10.1109/IROS.2016.7759843.

[35] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-Efficient Decentralized Visual SLAM," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, 2018. DOI: 10.1109/ICRA.2018.8461155.

[36] N. Mahdoui, V. Fremont, and E. Natalizio, "Cooperative exploration strategy for micro-aerial vehicles fleet," in *Proc. IEEE Int'l Conf. on Multisensor Fusion and Integration for Intelligent Systems*, 2017. DOI: 10.1109/MFI.2017.8170426.

[37] G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-Aware Communication for Decentralised Multi-Robot Coordination," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, 2018. DOI: 10.1109/ICRA.2018.8460617.

[38] K. Krinkin, A. Filatov, A. Filatov, O. Kurishev, and A. Lyanguzov, "Data Distribution Services Performance Evaluation Framework," in *Proc. Conference of Open Innovations Association FRUCT*, 2018.

[39] D. Tardioli, R. Parasuraman, and P. Ögren, "Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks," *Robotics and Autonomous Systems*, vol. 111, pp. 73–87, 2019. DOI: 10.1016/j.robot.2018.10.009.

[40] F. Martín, E. Soriano, and J. M. Cañas, "Quantitative analysis of security in distributed robotic frameworks," *Robotics and Autonomous Systems*, vol. 100, pp. 95–107, 2018. DOI: 10.1016/j.robot.2017.11.002.

[41] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the Performance of ROS2," in *Proc. Int'l Conf. on Embedded Software*, 2016. DOI: 10.1145/2968478.2968502.

[42] S. Oh, J.-H. Kim, and G. Fox, "Real-time Performance Analysis for Publish/Subscribe Systems," *Future Generation Computer Systems*, vol. 26, no. 3, pp. 318–323, 2010. DOI: 10.1016/j.future.2009.09.001.

[43] S. Tornell, E. Hernandez-Orallo, C. Calafate, J.-C. Cano, P. Manzoni, and E. Hernández, "An Analytical Evaluation of a Map-based Sensor-data Delivery Protocol for VANETs," in *Proc. IEEE Int'l Symp. on a World of Wireless, Mobile and Multimedia Networks*, 2013. DOI: 10.1109/WoWMoM.2013.6583405.

[44] OPTICOM GmbH, "PEVQ Advanced Perceptual Evaluation of Video Quality," OPTICOM GmbH, 2005.

[45] S. D'Aronco and P. Frossard, "Online Resource Inference in Network Utility Maximization Problems," *IEEE Transactions on Network Science and Engineering*, 2018. DOI: 10.1109/TNSE.2018.2832247.

[46] T. Kimura, T. Matsuura, M. Sasabe, T. Matsuda, and T. Takine, "Location-aware utility-based routing for store-carry-forward message delivery," in *Proc. Int'l Conf. on Information Networking*, 2015. DOI: 10.1109/ICOIN.2015.7057881.

[47] S. Raut, J. N. D. Gupta, and S. Swami, "Single Machine Scheduling with Time Deteriorating Job Values," *The Journal of the Operational Research Society*, vol. 59, no. 1, pp. 105–118, 2008. DOI: 10.1057/palgrave.jors.2602303.

[48] P. Li, H. Wu, B. Ravindran, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 454–469, 2006. DOI: 10.1109/TC.2006.47.

[49] X. Gan, Y. Wang, Y. Yu, and B. Niu, "An Emergency Vehicle Scheduling Problem with Time Utility Based on Particle Swarm Optimization," in *Intelligent Computing Theories and Technology. Lecture Notes in Computer Science*, D.-S. Huang, K.-H. Jo, Y.-Q. Zhou, and K. Han, Eds., Springer, Berlin, Heidelberg, 2013, pp. 614–623.

[50] M. Mu, A. Mauthe, and F. Garcia, "A Utility-Based QoS Model for Emerging Multimedia Applications," in *Proc. Int'l Conf. on Next Generation Mobile Applications, Services, and Technologies*, 2008. DOI: 10.1109/NGMAST.2008.24.

[51] Y. Xia, H.-S. W. So, R. H.-J. La, V. Anantharam, S. McCanne, D. Tse, J. Walrand, and P. Varaiya, "The Framework of User-Centric Optimization in Web-Based Applications," Department of Electrical Engineering and Computer Science University of California, Berkeley, 2000.

[52] R. W. Yeung, *Information Theory and Network Coding*. Springer, 2008, ISBN: 978-0-387-79233-0.

[53] S. A. Williamson, E. Gerding, and N. R. Jennings, "A principled information valuation for communications during multi-agent coordination," in *Proc. AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2008.

[54] A. Alshehri, T. Miller, and L. Sonenberg, "Modeling communication of collaborative multiagent system under epistemic planning," *International Journal of Intelligent Systems*, vol. 36, no. 10, pp. 5959–5980, 2021. DOI: 10.1002/int.22536.

[55] J. Hintikka, "Knowledge and Belief: An Introduction to the Logic of the Two Notions," *Studia Logica*, vol. 16, pp. 119–122, 1962.

[56] H. van Ditmarsch, W. van der Hoek, and B. Kooi, *Dynamic Epistemic Logic*, ser. Synthese Library. Springer Netherlands, 2008, ISBN: 978-1-4020-5838-7. DOI: 10.1007/978-1-4020-5839-4.

[57] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," *IEEE Comm. Surv. Tutor.*, vol. 21, no. 1, pp. 562–585, 2019. DOI: 10.1109/COMST.2018.2862938.

[58] J. Kua, G. Armitage, and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1842–1866, 2017. DOI: 10.1109/COMST.2017.2685630.

[59] Y. Sani, A. Mauthe, and C. Edwards, "Adaptive Bitrate Selection: A Survey," *IEEE Comm. Surv. Tutor.*, vol. 19, no. 4, pp. 2985–3014, 2017. DOI: 10.1109/COMST.2017.2725241.

[60] S. Kacianka and H. Hellwagner, "Adaptive Video Streaming for UAV Networks," in *Proc. ACM International Workshop on Mobile Video (MoVid)*, 2015. DOI: 10.1145/2727040.2727043.

[61] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale," *IEEE J. Sel. Areas Comm.*, vol. 32, no. 4, pp. 719–733, 2014. DOI: 10.1109/JSAC.2014.140405.

[62] M. Böhmer, A. Schmidt, P. Gil Pereira, and T. Herfet, "Latency-aware and -predictable Communication with Open Protocol Stacks for Remote Drone Control," in *Proc. Int'l Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2020, pp. 304–311. DOI: 10.1109/DCOSS49796.2020.00055.

[63] Y. Liu, H. Wang, y. Lin, S. Cheng, and G. Simon, "Friendly P2P: Application-Level Congestion Control for Peer-to-Peer Applications," in *Proc. of the Global Communications Conference (GLOBECOM)*, 2008. DOI: 10.1109/GLOCOM.2008.ECP.334.

[64] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," in *Proc. SIGCOMM*, 2006. DOI: 10.1145/1151659.1159918.

[65] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Comm.*, vol. 13, no. 8, pp. 1465–1480, 1995. DOI: 10.1109/49.464716.

[66] S. H. Low, L. Peterson, and L. Wang, "Understanding TCP vegas: A duality model," in *Proc. ACM Int'l Conf. on Measurement and Modeling of Comp. Sys. (SIGMETRICS)*, 2001. DOI: 10.1145/378420.378787.

[67] S. D'Aronco, L. Toni, S. Mena, X. Zhu, and P. Frossard, "Improved Utility-Based Congestion Control for Delay-Constrained Communication," *IEEE ACM Transactions on Networking*, vol. 25, no. 1, pp. 349–362, 2017. DOI: 10.1109/TNET.2016.2587579.

[68] N. Cardwell, Y. Cheng, G. C. Stephen, Y. Hassas, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, no. 5, 2016.

[69] "Video: Firefly Synchronization of a Robot Swarm." (2018), [Online]. Available: https://youtu.be/BfPEWH4ASXE.

[70] A. Barciś and C. Bettstetter, "Beyond sync: Distributed temporal coordination and its implementation in a multi-robot system," in *Proc. IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems (SASO)*, 2019. DOI: 10.1109/SASO.2019.00020.

[71] "Video: Robots that sync and swarm: A proof of concept in ROS2." (2018), [Online]. Available: https://youtu.be/atLhROLzsFo.

[72] "Video: Demonstrator of Information Distribution in Multi-Robot Systems Optimization." (2021), [Online]. Available: https://youtu.be/BfPEWH4ASXE.

[73] Y. Sung, A. K. Budhiraja, R. K. Williams, and P. Tokekar, "Distributed Assignment with Limited Communication for Multi-Robot Multi-Target Tracking," *Autonomous Robots*, 2019. DOI: 10.1007/s10514-019-09856-1.

[74] M. Corah, C. O'Meadhra, K. Goel, and N. Michael, "Communication-Efficient Planning and Mapping for Multi-Robot Exploration in Large Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1715–1721, 2019. DOI: 10.1109/LRA.2019.2897368.

[75] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, "Drone Networks: Communications, Coordination, and Sensing," *Ad Hoc Networks*, vol. 0, pp. 1–15, 2017. DOI: 10.1016/j.adhoc.2017.09.001.

[76] E. A. Thompson, C. McIntosh, J. Isaacs, E. Harmison, and R. Sneary, "Robot communication link using 802.11n or 900MHz OFDM," *Journal of Network and Computer Applications*, vol. 52, pp. 37–51, 2015. DOI: 10.1016/j.jnca.2015.02.005.

[77] A. Leelasantitham and P. Chaiprapa, "A study of performances on an automatic IEEE 802.11g wireless-standard robot using infrared sensors," in *Proc. IEEE Int'l Conf. on Robotics and Biomimetics*, 2009, pp. 1556–1560. DOI: 10.1109/ROBIO.2009.4913232.

[78] B. B. Luu, B. J. O'Brien, D. G. Baran, and R. L. Hardy, "A Soldier-Robot Ad Hoc Network," in *Proc. IEEE Int'l Conf. on Pervasive Computing and Communications Workshops (PerComW)*, 2007, pp. 558–563. DOI: 10.1109/PERCOMW.2007.13.

[79] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: An open-source Robot Operating System," in *Proc. ICRA Workshop on Open Source Software*, vol. 3, 2009.

[80] Object Management Group. "Data Distribution Service (DDS)." (2015), [Online]. Available: https://www.omg.org/spec/DDS/.

[81] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. "O'Reilly Media, Inc.", 2017, ISBN: 978-1-4919-7977-8.

[82] "Source code repository with Ansible playbooks." (2018), [Online]. Available: https://gitlab.aau.at/aau-nav/development/ansible-playbooks.

[83] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A Large Nano-Quadcopter Swarm," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3299–3304. DOI: 10.1109/ICRA.2017.7989376.

[84] G. T. Fechner, *Elements of Psychophysics*. Holt, Rinehart and Winston, 1966.

[85] D. Pynadath and M. Tambe, "The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models," *Journal of Artificial Intelligence Research*, vol. 16, 2002. DOI: 10.1613/jair.1024.

[86] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. Computers and Games*, Springer-Verlag, 2006. DOI: 10.1007/978-3-540-75538-8_7.

[87] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., ser. Lecture Notes in Computer Science, Springer, 2006, pp. 282–293. DOI: 10.1007/11871842_29.

[88] T. Cazenave, A. Saffidine, and N. Sturtevant, Eds., *Computer Games*. Springer, 2019, ISBN: 978-3-030-24336-4.

[89] G. Corera, "Google achieves AI 'breakthrough' at Go," *BBC News*, 2016.

[90] T. Pepels, M. H. M. Winands, and M. Lanctot, "Real-Time Monte Carlo Tree Search in Ms Pac-Man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 245–257, 2014. DOI: 10.1109/TCIAIG.2013.2291577.

[91] C. Mansley, A. Weinstein, and M. L. Littman, "Sample-based planning for continuous action Markov Decision Processes," in *Proc. Int'l Conf. on Automated Planning and Scheduling (ICAPS)*, ser. ICAPS'11, AAAI Press, 2011, pp. 335–338.

[92] M. Barciś, *Information Distribution Optimization Middleware*, 2020. [Online]. Available: https://github.com/zeroos/infdist.

[93] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *Int'l Conf. on Comm. (ICC)*, vol. 3, 1979, pp. 43.1.1–43.1.10.

[94] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[95] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989. DOI: 10.1016/0169-7552(89)90019-6.