Online Gathering Algorithms for Wireless Networks

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Korteweg, Peter

Online gathering algorithms for wireless networks / door Peter Korteweg. –
Eindhoven : Technische Universiteit Eindhoven, 2008.
Proefschrift. – ISBN 978-90-386-1237-9
NUR 919
Subject headings: algorithms / combinatorial optimisation / computational
complexity / computer networks / data communication
2000 Mathematics Subject Classification: 68Q25, 68Q17, 68W25, 68W15,
90B18, 90C27

Online Gathering Algorithms for Wireless Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 17 april 2008 om 16.00 uur

door

Peter Korteweg

geboren te Utrecht

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.K. Lenstra
en
prof. A. Marchetti-Spaccamela

Copromotor:
dr. L. Stougie

Acknowledgments

I want to thank Leen. Leen has stimulated me to get the most out of my Ph.D. position. He has given me the opportunity to work on a variety of mathematical problems, to meet researchers, and to attend research schools and conferences throughout Europe. Leen has always shown great interest in my research, and provided me with valuable feedback. Working with Leen on joint papers has been very fruitful and a pleasant experience. He also demonstrated to me the importance of having a pair of binoculars with you at all times. Leen has been the best supervisor I could have wished for.

Also, I want to thank Alberto. I have enjoyed doing joint research with Alberto and Leen which culminated in a series of papers, that form the basis of this thesis. I have benefited greatly from Alberto's knowledge of algorithm design and proof techniques. Alberto has been a great host during my visits to "La Sapienza" University in Rome, and an excellent cook. I am pleased to have Alberto as my promotor.

Jan Karel has provided me with valuable comments on my thesis; his advice improved the theoretical exposition, as well as my style of writing. I thank Jan Karel for his comments, and for acting as promotor.

I thank Gerhard, John, and the other members of the combinatorial optimization group for providing a nice research atmosphere, and for interesting lunch discussions. I greatly appreciated discussing optimization in practice with Cor while we both competed in the ROADEF Challenge.

I thank Vincenzo for his pleasant cooperation in Rome and Eindhoven, and I thank Marcel and Ank for their detailed comments on the introduction of this thesis.

I am grateful to the Algorithmic Discrete Optimization Network (ADONET) and to the European Union COST-action 293 GRAAL for funding visits to "La Sapienza" University in Rome, and to several mini-courses throughout Europe. My position at Eindhoven University of Technology was funded by the Netherlands Organisation for Scientific Research (NWO) through the NWO Open Programme.

There is more to life than mathematics. Friends from VIAE, Creditino and econometrics have enriched my life. Finally, I thank my family and Diana for their love and support.

Peter Korteweg

Contents

Chapter 1

Introduction

1.1 Optimization problems

Imagine that you are planning a weekend trip with friends. You have found a nice city to visit, and now you have to make the travel plans. So, you take a road map in order to decide which is the shortest route to the city of your choice.

This situation is a typical example of an *optimization problem*. Mathematically speaking an optimization problem is a general description of a situation which requires taking one or more decisions. These decisions together form a *solution* to the problem, and one is interested in finding an optimal solution among all solutions.

A particular example of the problem is called an *instance* of the problem, and contains relevant data specific to that particular instance. An *objective function* determines the value one attaches to each solution, and the *goal* of an optimization problem is to find a solution with either minimum or maximum objective value, called an *optimal solution*. Formally we have the following definition.

Definition 1.1 (Optimization Problem). *An optimization problem is a collection of the following:*
- a set of instances;
- a set of solutions for each instance;
- an objective value for each solution;
- a goal which is to minimize or to maximize the value of the solution.
For each instance the answer to an optimization problem is a solution which satisfies the goal of the problem, or the answer is that such a solution does not exist.

Typically, the set of instances of a problem is not given explicitly. Rather the problem consists of a description of an instance which depends on a small set of parameters. A particular instance is then defined as the representation of those parameters. There is a similar compact representation for solutions.

In this thesis we only consider minimization problems. This does not restrict our results, because any optimization problem can be formulated as a minimization problem. Also, we assume that objective values are non-negative.

The example we described above is an instance of the shortest path problem, and the goal of this problem is to find the shortest path between two points in a network. In this example shortest refers to the length of the route, and the network is a road map. Relevant problem data are streets, street crossings, street lengths, a start, and an end.

The shortest path problem is in fact a *combinatorial optimization problem*. A combinatorial optimization problem is an optimization problem in which for each decision there is a finite number of choices from which to choose. Even though the number of decisions and choices is finite, there are many combinations of choices, therefore the number of solutions is typically much larger than the number of choices and decisions. E.g., in the shortest path example there are many more routes leading to a city than there are streets on the map, although many of these routes are too long to be considered in practice.

Mathematicians study a combinatorial optimization problem to find a method that provides an answer to each instance of the problem. We call a formal description of such a method an *algorithm*.

Definition 1.2 (Algorithm). *An algorithm for a problem is a well-defined list of instructions which results in an answer to each instance of the problem.*

In theory, we can find the answer to a combinatorial optimization problem by considering all solutions, and choosing an optimal solution. However, because the number of solutions is typically large this so-called complete enumeration of solutions can quickly become too time-consuming, even if we use a computer. Therefore, we are interested in finding efficient algorithms.

1.2 Network problems

A *network problem* is a problem in which a network forms an important part of the problem description. Many real life problems, such as the shortest path problem, can be formulated as network problems. Another example is the traveling salesman problem (TSP) in which one is interested in finding a shortest tour visiting all points in a network exactly once. Throughout this chapter we use the shortest path problem and TSP to illustrate the concepts that we introduce.

Mathematicians represent a network as a graph, i.e., a set of nodes and a set of edges, which are connections between two nodes. The representation can also include a length function, which attaches a value to each edge. A graph formulation of a network problem typically describes the problem as finding a set of nodes and/or edges under specified constraints minimizing some objective function. E.g., the graph formulation of the shortest path problem is to find an ordered set of edges that represent a path of shortest length between two nodes.

Besides a graph formulation, network problems can also be modeled as linear programming problems (LP) of the form $\min\{cx|Ax \leq b, x \geq 0, x \in \mathbb{R}^n\}$; in case the variables x are constrained to the set of integers, we speak of an integer linear

programming problem (ILP). The shortest path problem can be formulated as an LP and TSP as an ILP.

Typical network problems consist of routing one or more entities over the network. In this thesis we consider routing of data packets over wireless networks. In particular, we are interested in problems where all data packets have to be sent to a central node, the sink. We call such a problem a *gathering problem*.

Due to problem restrictions, not all packets can be routed simultaneously over the network. Hence, time forms an important part of the problems that we consider. In this case the solution consists of a *schedule*, a list of decisions taken over time. In general such problems are called scheduling problems and a standard objective for these problems is to minimize the makespan, which is the time required to execute the schedule.

1.3 Complexity theory

Complexity theory classifies problems based on the existence of efficient algorithms for these problems. We present a brief overview of complexity theory and give a definition of efficiency. For a rigorous treatment of complexity theory see [64].

A formal definition of algorithm efficiency is based on the running time of an algorithm. We measure the time of an algorithm in basic computer operations, and we express the running time in the size of an instance; basic computer operations are addition, multiplication, and comparison, and the size $|I|$ of instance I is the space required to store the instance data in a computer.

For an optimization problem Π the running time of an algorithm on a particular instance $I \in \Pi$ is the number of basic computer operations the algorithm needs to provide an answer to this instance, expressed as a function of the size of the instance, and denoted $t(I)$. Let $I_n = \{I \in \Pi, |I| \leq n\}$, the set of instances of problem Π of size at most n. The *running time of an algorithm* is a function T that maps size n to a function of the running time of that algorithm on instances in I_n, i.e. $T(n) = G(\{t(I), I \in I_n\})$.

For complexity theory we consider the *worst-case* running time of an algorithm, which is a function that maps size n to the maximum running time of that algorithm on an instance in I_n, i.e. $T_{\max}(n) = \max_{I \in I_n} t(I)$. This yields the following definition of efficiency.

Definition 1.3 (Efficiency). *An algorithm is efficient if its worst-case running time is bounded by a polynomial function of the size of the instance.*

Informally, we also say that an efficient algorithm runs in polynomial time. The definition of efficiency includes algorithms which may have very different running times, e.g., $O(n^2)$ or $O(n^{10})$ for an instance of size n. In practice, one would choose from a set of efficient algorithms the algorithm whose running time is bounded by a polynomial function of smallest degree.

There are several reasons for this definition of efficiency. First, worst-case analysis provides a guarantee on the running time of an algorithm, regardless of the

instance. As such the definition of efficiency is as general as the definition of an algorithm, which should provide an answer to any instance of the problem. Second, polynomial functions have good scaling properties with respect to instance size. In real life instances, the instance size is often large, hence we are interested in running times for large instances. The amount of additional instances one can solve if one assigns extra computer resources is much more for polynomially bounded functions than for super-polynomial functions. E.g., consider two algorithms, one runs in n^2 time, and the other runs in 2^n time. If we increase the amount of computer resources to four times the original amount, then the size of instances that we can solve with the first algorithm doubles, but the size of instances that we can solve with the second algorithm increases by two. Hence, even in case the value of some super-polynomial function is less than that of a polynomial function on small instances, the latter will have a smaller value for instances large enough. Third, this definition of efficiency has proven to provide a very robust conceptual framework to classify optimization problems. This framework is known as complexity theory, and will be discussed below. And fourth, from a practical point of view efficient algorithms have good running times on practical problems.

The main drawback of worst-case analysis is that it may be based on instances which do not often occur in practice. Researchers have suggested other measures of running time, such as average-case analysis, which is based on a distribution function of problem instances. However, such an analysis only yields a valid answer in case the distribution function is known, or can be approximated to a certain degree. There are cases where the distribution function is not known, and obtaining a good approximation of the distribution function is not possible or too costly. E.g., as economist Keynes remarked on the future: "[T]here is no scientific basis on which to form any calculable probability whatever. We simply do not know!" (see [14]). In such a case we face true *uncertainty* of the instance we are likely to encounter, and worst-case analysis addresses the behavior of an algorithm in this case.

Probably the best known algorithm for which worst-case analysis provides a much weaker bound than average-case analysis is the simplex algorithm for LP. Many variants of the simplex algorithm have a worst-case running time which is exponential in the instance size. However, there exists a simplex algorithm which has a polynomially bounded average-case running time [2]. In practice simplex algorithms are competitive with interior point method algorithms which have a polynomially bounded worst-case running time [15, 16].

Consider again the shortest path problem. For this problem the size of an instance depends on the number of streets m, and the number of crossings n. There are many efficient algorithms for this problem, including Dijkstra's algorithm, which has a running time of $O(n^2)$ [74].

Complexity theory classifies problems based on the existence of efficient algorithms for these problems. Formally the classes are defined for decision problems. A decision problem is a problem for which the objective value is either 'yes' or 'no'. Optimization problems are closely related to decision problems. Given an instance I of an optimization problem, a solution set S, objective to minimize $f(x)$, and a parameter C, the naturally related decision problem is: is there a solution $x \in S$ such

that $f(x) \leq C$? We call this the recognition version of the optimization problem. Also, we call a solution which yields the answer 'yes' a certificate, and an instance for which there exists a certificate, a yes-instance. Under general assumptions an efficient algorithm for the recognition version of an optimization problem also yields an efficient algorithm for the optimization problem itself.

The class **P** is the set of decision problems for which there exists an efficient algorithm. E.g., the recognition versions of the shortest path problem and LP are in class **P**. The class **NP** is the set of decision problems for which there exists an algorithm which, given a yes-instance and a polynomial size certificate, gives in polynomial time the answer 'yes'. It is easy to see that $\mathbf{P} \subseteq \mathbf{NP}$, but the question $\mathbf{P} = \mathbf{NP}(?)$ remains one of the best known open problems in mathematics [64].

We can relate the complexity of decision problems through a reduction. We say that a decision problem Π polynomially reduces to decision problem Π', if each instance I of Π can be transformed in polynomial time to an instance I' of Π', the size of I' is bounded by a polynomial in the size of I, and I' is a 'yes'-instance if and only if I is a 'yes'-instance. If Π reduces to Π' this means that the complexity of Π is at most the complexity of Π' times a polynomial function of the size of the input. In particular, if $\Pi' \in \mathbf{P}$ then also $\Pi \in \mathbf{P}$.

A decision problem in the class **NP** is called **NP**-complete if for any problem in **NP** there is a polynomial time reduction to this problem. Since Cook established **NP**-completeness of the satisfiability problem in 1971, many problems have been identified as **NP**-complete [37, 65]. An optimization problem is **NP**-hard if its recognition version is **NP**-complete. E.g., TSP is **NP**-hard [37].

1.4 Approximation algorithms

For **NP**-hard problems there are no known efficient algorithms, and in case $\mathbf{P} \neq \mathbf{NP}$ such algorithms do not even exist for these problems. Hence, for such problems mathematicians have studied several alternatives to finding the optimal solution efficiently. One such approach is to find efficient algorithms for **NP**-hard optimization problems which yield solutions that are approximately optimal. Given an optimization problem Π let $C^*(I)$ be the value of an optimal solution to instance $I \in \Pi$, and let $\alpha \geq 1$.

Definition 1.4 (Approximation algorithm). *An α-approximation algorithm for problem Π is an efficient algorithm which yields a solution of value at most $\alpha C^*(I)$ for each instance $I \in \Pi$ for which the optimal solution exists.*

The *approximation ratio* of problem Π is the infimum of those values α for which there exist an α-approximation algorithm for problem Π. A 1-approximation algorithm is an algorithm which yields an optimal solution; such an algorithm can be called an exact algorithm to distinguish it from approximation algorithms. Usually the difference is clear from the context, and we refer to exact algorithms as algorithms.

Similar to complexity theory, we can classify optimization problems based on the existence of approximation algorithms. The class **APX** is the set of optimization problems for which there exists an efficient α-approximation algorithm for some constant α, i.e. α does not depend on the size of the instance. E.g., TSP does not belong to class **APX**, but METRIC TSP a practical version of TSP, which holds in case of two-way traffic, is in **APX** because there exists a 3/2-approximation algorithm for this problem [24].

1.5 Information models

An algorithm bases its solution on information of the instance. Until now we have assumed that the algorithm knows all information which describes the instance completely. There are problems in which it is more natural that information of the problem becomes known over time, or information is distributed over sources. In this case the algorithm should take into account the limitations on the availability of information when it makes decisions.

An *information model* is a model which defines in which way information of a problem is or becomes available. We briefly discuss several information models, with an emphasis on those that model limitations caused by time and location.

1.5.1 Online model

In an *online* model information on data becomes known over time. This model reflects the uncertainty we have in general of future events. A main feature of online problems is the *release time* of data. Each piece of data has a release time, and information on the data becomes known at this release time.

An algorithm for an online model makes decisions over time and each time it makes a decision, the decision is based on information which is available at that time; we call such an algorithm an online algorithm.

To distinguish the online model from the original model where all problem information is known a priori we may refer to this model as the offline model. Typically it depends on the nature of the problem whether an offline model or an online model is more appropriate. Even for problems which are online by nature an analysis of the offline model may help to gain an insight in the computational complexity of the problem.

E.g., in case of TSP both models can be appropriate, depending on the situation; offline TSP offers the possibility to find tours for recurring routing problems which occur in planning problems. In online TSP the problem consists of finding a tour of minimum makespan which visits a set of nodes whose locations become known at their respective release times. Note that in this version the tour is in fact a schedule and the objective depends on the time needed to execute the schedule; i.e., it includes both the time to visit all nodes as well as time spent waiting at nodes. Online TSP is more appropriate for unscheduled traffic, such as taxi-routing.

As in the case of approximation algorithms we can relate the value of a solution of an online algorithm to the value of an optimal solution to the offline problem. Again, let $C^*(I)$ be the value of an optimal (offline) solution to instance I, and let $\alpha \geq 1$.

Definition 1.5 (Competitive algorithm). *An α-competitive algorithm for problem Π is an online algorithm which yields a solution of value at most $\alpha C^*(I)$ for each instance $I \in \Pi$ for which the optimal offline solution exists.*

Competitive analysis of a problem is the study of finding α-competitive algorithms for some value α. The *competitive ratio* of problem Π is the infimum of those values α for which there exist an α-competitive algorithm for problem Π.

Note that an online algorithm need not be efficient. The reason for choosing this definition of competitive ratio is that in this way the ratio reflects only the difference in the solution value between online and offline algorithms. Hence, there can be a difference between the approximation ratio of a problem and the competitive ratio of a problem, and either of the two can be smaller than the other. E.g., Ausiello et al. [5] presented a 2-competitive algorithm for online Metric TSP, and demonstrated that the competitive ratio for this problem also equals 2. Their algorithm is not efficient, unless $\mathbf{P} = \mathbf{NP}$, nevertheless the competitive ratio is strictly larger than the approximation ratio of Metric TSP, which is at most 3/2. From a practical point of view, one is typically interested in efficient online algorithms; in this thesis we only consider efficient online algorithms.

Often, the competitive ratio is viewed as the outcome of a two-person game between an online player and an offline player, called the adversary. The online player chooses an online algorithm; next the adversary chooses an instance of the problem based on the algorithm the online player has chosen. The cost of the adversary is the value of an optimal offline solution, and the cost of the online player is the value of the online solution the algorithm produces. The adversary chooses the instance in such a way that the ratio of these costs is maximized.

Competitive analysis has the same disadvantage as complexity theory and approximation analysis, in the sense that it focuses on worst-case running times. However, in this case the effect of this disadvantage can be aggravated due to the fact that the competitive ratio relates an online solution to an offline solution. For some problems this yields an adversary which is so powerful that no algorithm can obtain a ratio which is better than some trivial upper bound on the competitive ratio. As such competitive analysis fails to differentiate between algorithms.

In the literature there exist several ways to compensate for the power of the adversary in such cases. In this thesis we consider *resource augmentation* as a way to compensate for the power of the adversary. In resource augmentation we give the online algorithm more resources than the adversary; typical resources are speed and size. In a σ-speed version of a problem we assume that the online player requires $1/\sigma$ units of time where the offline adversary requires 1 unit of time, for some $\sigma > 1$. E.g., in σ-speed TSP the online player is σ times faster than the offline adversary. In case of size augmentation, we assume the online player can use more resources than the adversary; in case of TSP this would mean more salesmen. A moderate

increase in the speed of algorithms may sometimes yield constant bounds on the competitive ratio of algorithms, whereas the competitive ratio is not a constant without resource augmentation [49, 67]. Basically, resource augmentation indicates the effect of a future investment in extra or faster resources in relation to an offline solution in the present. Note that resource augmentation may yield solutions with a value which is better than the value of an optimal offline solution.

A more restricted online model is the oblivious online model. In an oblivious model we assume that historical information cannot be used for future decisions. An *oblivious* or *memoryless* algorithm can base its decisions on information present at that time, not on information of previous decisions [20, 22].

There are numerous ways to restrict the information an algorithm can use in order to make its decisions. Typically, we analyze the competitive ratio of a restricted class of algorithms using the original definition of competitive ratio, hence the ratio depends on the cost of an unrestricted offline solution. Another approach is to compare classes of algorithms which correspond to different information models directly. This is called comparative analysis [54]. Finally, we can also compare the competitiveness of algorithms using the same information model. We call an α-competitive algorithm best possible within an information model if there is no α'-competitive algorithm with $\alpha' < \alpha$, in the same information model.

1.5.2 Distributed model

In a *distributed* model information of the instance is spread over several sources. The sources can be either processors, agents, or in case of a network nodes; we refer to the sources as nodes. Each node possesses some information of the instance, and should base its decisions on information it has available. Hence, in a distributed model each node contains an algorithm, which we call a distributed algorithm. Without further a priori knowledge of the problem instance, it is common to assume that each node uses the same algorithm.

Most distributed models are online models by nature, where nodes can exchange information with each other over time, and thus obtain more information. To distinguish the distributed model from the original model where all problem information is known to a single source, we may refer to the original model as the centralized model.

In distributed models a model of time becomes an important factor in the design of algorithms. We distinguish two main time models, the synchronous time model and the asynchronous time model. In a synchronous time model all nodes observe the same absolute time, whereas in an asynchronous time model each node only has a relative notion of time. A more precise description of time models specific to wireless networks is discussed in Chapter 2.

A relaxation of the distributed model is the *local* or *decentralized* information model. In this model we assume that a node can use information which is available to the node, and to nodes which are close to this node, e.g., adjacent nodes.

1.6 Outline of the thesis

We study optimization problems in wireless networks. We focus on deriving complexity results, and on finding efficient algorithms for each of the problems that we consider. We study the problems for various information models, and we assess the quality of our algorithms using approximation theory and competitive analysis.

Chapter 2 gives a background on wireless networks, introducing definitions, concepts and common problems in this field. These problems form the motivation for the optimization problems which we describe and analyze in the succeeding chapters.

Chapters 3 and 4 discuss a data aggregation problem in wireless networks, with constraints on packet latency.

In Chapter 3 we focus on the problem with arbitrary latency constraints which are modeled as hard constraints. The objective is to minimize the maximum communication costs of a node. We prove that the problem is **NP**-hard, and we provide offline and online approximation algorithms. For the synchronous time model the algorithm we present is best possible up to a multiplicative constant. For the asynchronous time model we present an algorithm with competitive ratio at most $O(\delta)$ times the competitive ratio of our synchronous algorithm, where δ is the maximum communication cost of some packet to reach the sink. We show that our algorithms are robust, in the sense that they have similar competitive ratios in case we choose as objective to minimize the sum of communication costs, in case we assume the cost function to be concave, or in case we limit the possibilities of aggregation.

In Chapter 4 we consider the same problem, but focus on constant latencies which are modeled as soft constraints. We use bicriteria optimization to find solutions with both low costs and small packet latencies. The main contributions of this chapter are that we present a constant competitive online distributed algorithm for this problem, in case the latency constraints are not too strict, and an analysis of the almost synchronous time model, a new time model.

The results in these chapters are partially based on joint work with Luca Becchetti, Alberto Marchetti-Spaccamela, Martin Skutella, Leen Stougie and Andrea Vitaletti [10, 53].

Chapters 5 and 6 discuss a gathering problem in a wireless network with interference, the wireless gathering problem.

In Chapter 5 we analyze the problem with objective to minimize completion times. We focus on minimizing the maximum completion time. We prove that this problem is **NP**-hard, in case communication radius equals interference radius, even in case each node contains exactly one packet, solving an open problem. We present a class of constant competitive online algorithms and we demonstrate that a particular algorithm of this class is best possible within the class of algorithms that send packets over shortest paths.

In Chapter 6 we consider the problem with objective to minimize flow times, the time packets are in the network. In case of minimizing maximum flow times we

present a proof of non-approximability, unless $\mathbf{P} = \mathbf{NP}$, and an online centralized algorithm which is constant competitive, in case the algorithm can send packets a constant factor faster than the adversary. In case of minimizing average flow times we present an unconditional proof of non-approximability for a general class of distributed algorithms, and an online distributed algorithm which is near-optimal in case the algorithm can send packets a factor a faster than the adversary, where a depends on the logarithm of the network diameter and the logarithm of the maximum node degree.

The results in these chapters are partially based on joint work with Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Leen Stougie [18, 17, 19].

1.7 Related literature

Literature on topics in combinatorial optimization is vast. We present an overview of handbooks and papers which cover the aspects of combinatorial optimization introduced in this chapter in more detail. This overview is by no means complete, but merely offers a starting point for further reading.

General information on combinatorial optimization problems can be found in Papadimitriou and Steiglitz [65]. Schrijver [74] provides extensive information on classical network optimization problems, including the shortest path problem and TSP. The book edited by Lawler et al. [57] provides a guided tour of TSP.

Complexity theory is treated rigorously in Papadimitriou [64]; [65] and Garey and Johnson [37] provide introductions, and the latter contains an extensive list of **NP**-complete problems. The books of Ausiello et al. [6], Hochbaum (ed.) [44], Hromkovič [45] and Vazirani [79] focus on approximation theory and approximation algorithms. The first book also provides an extensive list of known results on approximability of many optimization problems.

Borodin and El-Yaniv [20] provide an overview of online models and competitive analysis. The distributed time models that we described are based on work of Brito et al. [22]. Lynch [58] provides a more general overview of distributed time models. Resource augmentation in the context of scheduling problems is introduced by Kalyanasundaram and Pruhs [49] and Phillips et al. [67].

Chapter 2

Communication in wireless networks

2.1 Introduction

In the last two decades there has been an enormous increase in the use of mobile phones worldwide, from 11 million subscriptions in 1990 to over 2 billion in 2005. The penetration of mobile phones is currently around 30% worldwide, and even over 100% in some European countries including the Netherlands [23, 48]. Similarly, the use of notebooks, personal digital assistants and other wireless electronic devices has increased rapidly. Also, in recent years there has been extensive research in the area of sensor networks. Sensors are small electronic devices which operate automatically, sensing their environment and collecting data. Scientists believe sensor networks to become available for public use in the near future.

Main feature of these wireless devices is the availability of wireless communication between them. The communication devices, which we call *nodes*, can form a wireless network which enables the exchange of information such as speech, e-mails, and data files. A main challenge related to wireless networks is to establish *efficient* communication. Here, efficiency is related to the objective value and can be measured in different criteria, e.g. time or energy usage. Recall from Chapter 1 that we also use efficiency to characterize algorithms.

The study of efficient communication in wireless networks is a vast area which has generated much research in both hardware and software design. In this chapter we give a brief overview of wireless network architecture, and main problems related to finding efficient algorithms for efficient communication. We do not pretend to give a complete overview, instead our focus is to provide a sufficient background for understanding the communication problems discussed in this thesis. As such the overview focuses on the algorithmic aspects of wireless networks.

In Section 2.2 we describe architecture of wireless networks. In Section 2.3 we describe communication problems in wireless networks, introducing several object-

ive criteria. In Section 2.4 we introduce the communication problems in wireless networks which we study in this thesis. In Section 2.5 we summarize the models and problems discussed in this chapter. In Section 2.6 we give an overview of related literature on wireless networks, for further reference.

2.2 Wireless network architecture

In the literature there exist many types of wireless networks to address a variety of problems. We present a brief overview of a general network architecture common to many wireless networks. Also, we outline the scope of network models considered in this thesis.

The network architecture consists of hardware and software specifications. The hardware specifications address the specifications of the nodes, in particular the wireless communication device of these nodes. The software specifications address the algorithms used by the nodes. In practice the difference between software and hardware specifications is not always clear, as software is often integrated into hardware components. Nevertheless, we choose this model because the distinction between hardware and software specifications provides a clear model to describe wireless networks, at least from a theoretical point of view.

2.2.1 Hardware specifications

The hardware specifications of a wireless network architecture consist of the specification of the nodes, and of the network structure, i.e. the location of the nodes. The nodes of a wireless network may range from notebooks, mobile phones and PDAs to small sensor nodes. Each node is equipped with a data input device, a wireless communication device, a processor, memory, and an energy supply. A node can also be equipped with a clock.

The data input device enables a node to receive data from its environment. Data input is either through human interaction, e.g. through a keyboard, an electronic pen, or an USB-stick, or through automated sensing devices in sensor nodes. The processor and memory capabilities are used to store and process data. The communication device enables a node to send and receive data. An energy supply provides the energy to receive, process and communicate data. We consider batteries as the main energy supply. Typically, nodes have limited processor capabilities, memory, and battery power, except in notebooks. Especially sensor nodes are equipped with few resources, as they are being deployed in networks of huge size, and typically are not recharged [3]. A node can use a clock to attach a timestamp to the data, and to synchronize communication with other nodes.

In the next paragraphs we discuss the hardware specifications in more detail. First, we discuss wireless communication. Next, we discuss energy use in relation to communication. And, finally we discuss the influence of the network structure on network communication.

Wireless communication

In wireless networks nodes communicate with each other using a wireless medium. Typical wireless media are radio, infrared, and optical media. The last two media require that there is a line of sight between two nodes in order for them to communicate. I.e. these media do not function if there is an obstacle in between the two nodes. Although radio signals are also affected by obstacles between nodes, radio communication is possible in the presence of obstacles. In this thesis we focus on wireless radio networks.

Radio networks send data using a radio transmitter. The transmitter sends data using radio signals at a certain radio frequency. Data that is transmitted is *broadcasted*, which means that it is emitted to the region surrounding the transmitter. Radio signals are transmitted at a certain frequency or within a range of frequencies, and this range is called a broadcast channel.

There are two types of transmitters, based on the antenna being either unidirectional or omnidirectional. In the omnidirectional case the signal is broadcasted in any direction. In this case the broadcast region can be described as a ball centered at the sender node. In the unidirectional case, the antenna is pointed in a specific direction, hence the broadcast region can be described as a narrow cone centered at the sender node. In this thesis we focus on nodes with omnidirectional antennas.

There are two models for radio communication. There is a half-duplex model, in which at any time instant a node can either send or receive data, and there is a full-duplex model in which a node can both send and receive data simultaneously. In this thesis we focus on the half-duplex model. When two nodes communicate, we assume that there is a *sender node*, which has data to send, and a *receiver node* which wishes to receive the data. Data is communicated from the sender node to the receiver node, but the receiver node may use acknowledgement messages (ACKs) to confirm the data reception.

Radio signals have two important properties: fading and interference. *Fading* is the effect of radio signal loss due to physical circumstances. These circumstances are the composition of the space between the sender and receiver node, e.g. the above mentioned obstacles, and the distance between the sender and receiver node. The strength of a radio signal is a decreasing function of the distance d between the sender node and the receiver node, and the function is in the order of d^{-2} to d^{-6} [3, 21, 69, 73]. In the literature signal loss due to distance is also known as path loss. For a transmitter to receive data, the radio signal should be of a certain strength. As a consequence of this minimum signal strength and fading, the reachable broadcast region can be described as a closed ball, centered at the sender node. We call the radius of this ball the *communication radius*.

Interference, also called collision, is the effect of radio signal loss, due to the fact that multiple nodes communicate simultaneously on the same broadcast channel, within the same geographical region. As with data communication, interference occurs if the radio signal is strong enough. When a node broadcasts data, its radio signal is propagated to a region surrounding this node. The interference region is a closed ball, centered at the sender node. We call the radius of this ball the *interference radius*. Note that if a node sends data at certain power the interference

radius is at least as large as the communication radius, but may be larger [73]. A typical assumption is that if a node receives signals from multiple nodes, all data is lost.

The communication radius and interference radius depend on the power used by a node to communicate. Basically there are two power schemes. In a simple power scheme the signal strength is fixed, and in a power controlled scheme the signal strength can be adjusted for each transmission. In this thesis we consider simple power schemes with a fixed power, hence we assume fixed communication and interference radii.

The properties of fading and interference highly influence the design of wireless networks and communication algorithms. On the one hand fading makes communication over long distances costly, and interference limits the data throughput of the network. On the other hand fading allows multiple nodes to use the same broadcast channel simultaneously as long as the receiver nodes are sufficiently far apart. This is known as spatial frequency reuse [68, 69].

Energy use

Wireless devices are typically battery operated, and these batteries can not always be directly recharged. In case of sensor networks, batteries are not recharged at all either because sensor nodes are deployed in inaccessible areas, or because the scale of the network makes recollection too costly. As a consequence efficient energy use is currently a key challenge in wireless networks. Also, we foresee energy use to remain a key challenge in the future as technological advances in processing power are much faster than advances in battery storage [26]. We briefly outline the cost structure of a node's energy use.

Energy consumption of a node can be divided into the following domains: receiving, communicating, and processing data. Here, the process of receiving data exclusively refers to acquiring data through the input device; in case of sensor nodes this is also called sensing. Nodes use most of their energy in data communication, and for short-range communication the communication costs of sending and receiving are equivalent [3]. Also, smaller nodes such as sensors use a significant part of energy when they are activated but idle, i.e. even if the node is not active in any of the above mentioned domains [21]. A technique to reduce the energy use of idle nodes is to switch them to a sleeping mode. A node that sleeps is in a standby mode, and is activated occasionally for communication with other nodes. For an overview of sleeping mode techniques see [36]. In this thesis we focus on communication costs only.

The communication costs are divided into a fixed startup cost, and a variable transmission cost. The transmission costs typically depend linearly on the size of the data to be transmitted [43, 68]. Also, communication times have a similar dependence on the data size.

Network structure

The network structure consists of the network lay-out and the wireless communication properties of the nodes. The lay-out is based on the geographical position

of the nodes. A typical lay-out is the uniform lay-out where nodes are deployed randomly in a specified area. Other common lay-outs include grids and lines.

Based on the lay-out and node characteristics a *communication network* can be established; two nodes in the network are said to be connected if direct communication between the nodes is possible. Such a connection is also called a link. The connection depends on the network lay-out and the communication radius, as well as obstacles in the area which may block communication between certain pairs of nodes.

Over time the network structure may change due to several reasons. First, nodes may enter the network, when they require services from the network, and leave the network if they do not require network services anymore. Second, nodes may be put in sleeping mode to save energy. Third, nodes may stop to operate, either due to malfunction, or due to a depletion of their energy resources. And fourth, nodes may be mobile devices and move within the network region. A common assumption in the literature is to consider the graph structure fixed. If the amount of data that is communicated between two topology changes is significant such an assumption is reasonable. In this thesis we consider fixed wireless communication networks.

2.2.2 Software specifications

The software specifications of a wireless network architecture consist of specifications for the algorithms of the nodes. As such the specifications address a huge range of issues, and key issues include: error control, data flow control, channel allocation to nodes, data routing, energy use of nodes, location management, security, and information retrieval.

These examples of issues range from low level issues such as error control, which focuses on errors in the radio signal at bit level, to top level issues such as information retrieval, which focuses on presentation of the data to the end user.

Hence, for design purposes software architecture is organized in levels, also called layers. Each layer addresses certain networking issues, and offers an interface to communicate with adjacent layers. The layers are organized from communication at bit level, in the bottom layer to an interface with the end user in the top layer. We use the reference model of software layers given in Table 2.1. This model is also known as the five layer OSI reference model or the TCP/IP reference model [3, 78].

Table 2.1: Reference model of software layers.

5. Application layer
4. Transport layer
3. Network layer
2. Data link layer
1. Physical layer

We briefly describe each layer, from bottom layer to top layer. The *physical*

layer is concerned with communication of data at bit level. The *data link layer* is concerned with providing error-free node to node communication. At this level data is split into packets of similar size, and acknowledgement messages are used to ensure reception. The medium access (MAC) layer is a sub-layer of the data link layer which deals with access control to a shared broadcast channel. The *network layer* is concerned with routing of data packets, e.g. source to destination communication, and with addressing issues. The *transport layer* is an intermediate layer which is concerned with data splitting. The *application layer* serves as an interface with users.

Each layer contains a protocol. A *protocol* is an agreement to communicate in a certain layer, and it can be considered as a set of algorithms. Examples of internet protocols at the application level are the hypertext transfer protocol (HTTP) to display web pages, the file transfer protocol (FTP) to transfer files and the simple mail transfer protocol (SMTP) to send and receive emails. In the field of engineering, problems which address multiple layers are called cross-layer optimization problems, and protocols for such problems are called cross-layer protocols. An example of a cross-layer protocol is the TCP/IP model used for communication over the internet. This model consists of the transmission control protocol (TCP) for the transport layer and the Internet protocol (IP) for the network layer. In this thesis we study cross-layer optimization problems related to the data link layer and the network layer. More specifically, we devise algorithms to schedule and route data packets over a wireless network.

In the next paragraphs we consider in more detail the data link layer and the network layer. We also give an overview of different time models, as a time model greatly influences algorithm design. And we describe data aggregation, a technique used to reduce energy in data communication.

Time models

Time has a major impact on wireless communication. If multiple nodes send data at the same time to a single node, a collision occurs and data is lost. Also, data communication requires time, and minimizing the communication times or delay times is an important design issue in wireless networks. Thus, it is necessary to have a model of time.

In this paragraph we present several time models known in the literature. There are two main distinctions in time models, and these are the distinction between continuous and discrete time, and the distinction between synchronous and asynchronous time.

In a continuous time model nodes can start communication at any given time. Such a model is used in case nodes do not have a clock, hence they do not have a notion of time. But it can also be used in case nodes have a clock. In a discrete time model time is divided into rounds or time slots. Such a model can be used only in case nodes have a time clock, and nodes have a notion of the start time of each round.

Time synchronization is an important issue in wireless networks. We assume the existence of physical time, which is a reference time for the clocks of nodes in the

network. This physical time can either be an internal time, defined in the network, or an external time measure such as the Coordinated Universal Time (UTC), an international time standard. This physical time can be considered as an absolute measure; for which the length of a time unit is defined. Initially node clocks are synchronized to this physical time, and the time unit is set to match the time unit of physical time. However, due to inaccuracy the time unit cannot exactly match the time unit of physical time. Hence, over time the relative clock time of a node drifts from the physical time. Also, the length of a time unit, called clock frequency, may drift over time. Time synchronization algorithms synchronize both clock time and clock frequency to match physical time. We distinguish three such time models based on the precision of the time synchronization technique used. These models are the synchronous time mode, the almost synchronous time model, and the asynchronous time model.

In the synchronous time model the node clocks are considered to be synchronous to physical time, at any time. In the almost synchronous time model, the node clocks are considered to be synchronous to physical time up to a small constant, at any time. An upper bound on this constant is called the maximum drift. In the asynchronous time model we assume the node clocks are not synchronized at all. In this thesis we consider all three time models. However, we always assume that the clock frequency of all clocks is identical, at any time. Because of this frequency synchronization, all three time models are sometimes called partially synchronous time models in the literature [58].

Typically, in centralized models it is common to assume nodes have access to a single clock, which indicates physical time. In distributed models it is common to assume that each node has access to a local clock, which indicates relative clock time. Time synchronization enables nodes to coordinate their action in a distributed setting, i.e. it allows nodes to decide when to communicate such as to minimize the probability of collisions.

Data link layer

The data link layer addresses node to node communications. We consider a network with a single broadcast channel, thus an important part of this communication is to decide when a node has access to the channel. Protocols for this sub-problem are called MAC protocols. From an algorithmic point of view MAC protocols can be seen as scheduling algorithms.

We assume data is divided into packets of similar size. This is a common assumption in the literature. Even if data consists of packets which vary in size, typically the data link layer cuts data into smaller packets of more or less the same size. Also, we assume that in a discrete time model the size of the packets is such that a packet can be sent in a single round.

We distinguish two types of schemes for medium access and these are fixed assignment schemes, and contention schemes.

In fixed assignment schemes nodes are given a fixed allocation of resources, which are time, frequency or both. In time division nodes are allocated a time slot for radio communication, and in frequency division nodes are allocated a frequency range

for radio communication. Typically, there is a centralized node which allocates these resources. This can be a base station such as a server, a satellite, or a node which acts as a leader. Common fixed assignment schemes in the literature are the time division medium access (TDMA) model, the frequency division medium access (FDMA) model, and the code division medium access (CDMA) model which uses both time and frequency as a resource [21, 63].

There are two ways to allocate time slots, and these are static allocation, and dynamic allocation. In static allocation, an allocation scheme is based on the allocation of a single time slot to each node, and such an assignment can be considered a single communication round. A scheme consists of multiple communication rounds, i.e. a scheme consists of a cyclic assignment of the same time slots. Static allocation is an appropriate assignment model in case the data is uniform, i.e. each pair of sender and receiver nodes occurs with the same frequency in the data. In case data is not uniform, nodes may be assigned a slot, while they do not have data to communicate, and hence such an assignment wastes time and network capacity. In dynamic allocation the allocation of time slots is based on the demand of data communication of the nodes. This model is more complicated than the static allocation model, but is better suited to model non-uniform data communication flows. In this thesis we focus on dynamic allocation models.

In contention schemes nodes compete for a resource by sending data. Contention schemes are used in situations where there is no centralized control, i.e. in a distributed setting. A common contention scheme is the carrier sense multiple access (CSMA) model. In a distributed setting interference free communication can not be guaranteed. As a consequence nodes may have to send data multiple times, if interference occurs.

There exist several techniques to minimize interference, and to minimize the loss in time and energy associated with collisions. Two common techniques are backoff strategies, and the use of control messages.

A backoff strategy determines the delay between sending data after a collision has occurred. A node may notice a collision if it does not receive an acknowledgement message from the receiver node. Backoff strategies have a random component to avoid multiple nodes to choose the same delay.

In practice a backoff strategy works as follows. Each node with data to send chooses the start time of communication uniformly from a time interval, called the contention window. This model applies to both a continuous and a discrete time model. In theory usually a slightly different model is studied. In the theoretical model time is slotted, and each node with data to send, chooses to send in a round with a certain probability. It is not proven that both models are equivalent, but it is believed they yield similar results [41].

There are several classes of backoff strategies based on, respectively the length of the contention window in the practical model, and the backoff probability in the theoretical model. In a *constant backoff strategy* each node uses a constant probability. In an *exponential backoff strategy* each node uses a probability, which decreases exponentially if a collision occurs, and is reset to a constant if no collision occurs; typically the probability is halved until it reaches a lower bound. One of

the first backoff strategies is a constant backoff strategy used in the ALOHA model [1]. The Ethernet, a network for local communication between computers, uses an exponential backoff strategy [76].

A performance measure of backoff strategies is stability, which is defined as a bounded average message delay. In [51] it is shown that most backoff strategies are unstable, i.e. they lead to infinitely large delays, on problem instances with an unbounded number of nodes. The authors of [41] demonstrated that a super linear polynomial backoff strategy is stable if the number of nodes is bounded. Also, they demonstrated that linear and exponential backoff strategies are not stable.

A disadvantage of the above backoff strategies is that the model is not fair, in the sense that once a node sends data successfully, the probability increases that this node sends data successfully. The authors of [7] describe a backoff strategy which can be considered fair. This model is called a decay strategy and is as follows. Time is divided into rounds, and rounds into phases. At the start of each phase, all nodes that want to communicate, start communication. For each next round of the phase the node chooses to communicate with probability a half, or it chooses not to communicate for the rest of the phase with probability a half. In this model all nodes have the same probability of communication with some node. However, this implementation requires a known upper bound on the maximum number of nodes that can communicate with a single node, in order to determine the length of the phase. We use this backoff strategy in Chapter 6.

Control messages are data packets of small size. A node may use control messages to reduce the communication costs. A typical model of communication which uses control messages is the two-way handshake protocol. In this protocol the sender node sends a request-to-send (RTS) control message, and the receiver node responds with a confirm-to-send (CTS) control message, after which data communication can be established. In case the sender node does not receive a CTS message, it can assume that the receiver node has not received the RTS message due to interference. Control messages can also be used to confirm or acknowledge the reception of data.

Network layer

The network layer addresses end-to-end communication between nodes, i.e. from the sender node to the receiver node. The main issues of the network layer are to establish a routing network, and to route data over this network. From an algorithmic point of view protocols which address the second issue can be seen as routing algorithms.

The *routing network* is a subgraph of the communication network, i.e. it contains a subset of all possible links. As mentioned before, we consider networks where nodes have a fixed communication radius. Typically, the communication radius is chosen such that each node can only communicate with nodes within its direct vicinity, in order to decrease energy use, which increases more than quadratically with distance, and to allow spatial frequency reuse.

As a consequence nodes can not always directly communicate data to the receiver node, but instead the sender node has to send data over a path of nodes which ends at the receiver node, i.e. each node on the path forwards the data to the next node

on the path. This type of routing is called *multi-hop* routing, and in this case the choice of the path becomes an important decision.

Note that in case of multi-hop routing we may refer to the sender and receiver node as the pair of nodes which are the origin and destination of the data, as well as two nodes on the multi-hop path which transfer the data. It will be clear from the context to which nodes we refer.

Finding a routing network in a wireless network is an interesting problem on its own, which we do not address here. In this thesis we are interested in routing over a routing network, and we assume the routing network is known.

A sender node that wishes to send data to a receiver node using multi-hop routing, can use two modes of communicating data over the network: broadcasting, and point-to-point communication. In *broadcasting* the sender node sends data to any node within its communication radius, and these nodes in turn also broadcast the data, until it reaches the destination; this is also known as flooding. The broadcasting mode of communication is typical in problems without a routing network, or in case the same data has to be sent to many nodes, see Section 2.3 for classes of such problems. In *point-to-point* communication the sender node sends the data to a specified node within its communication radius, and this node also sends the data to a next node on a multi-hop routing path towards the receiver node. Point-to-point communication is common in problems with personalized data. In this thesis we only consider point-to-point communication.

We study communication in arbitrary networks, tree networks and line networks. A tree network is a typical routing network, where each node sends data to a single node only, called its parent. A tree is often used in data aggregation problems [3, 22]. The advantages of a tree include the fact that nodes do not have to choose to which node to send, and the number of links depends linearly on the number of nodes, thus reducing the possibility of collisions. Another reason to use a routing tree is that a tree is a minimal size network which enables node to node communications, without having to resort to broadcasting or flooding. Typically, a wireless network is equipped with an algorithm to derive a routing network [66]. Trees can be used to collect data at a central node, the root of the tree. This root node may process the data, act as a gateway to other information networks, or it can be used to further communicate the data in the network using a reverse routing tree; a reverse tree is a tree where each node receives data from a single node. Lines are trees where each node can communicate with at most two other nodes.

There are also other graph models which take into account the geometry of the network, such as planar graph and unit disk graph models. In some cases, the algorithmic performance significantly improves for these restricted graphs classes. However, such graph model are in fact quite a restricted graph class, because such graphs exclude the possibility that nodes which are close may not be able to communicate, due to obstacles. Hence, we choose to consider only the graph models outlined above.

We observe that the choice of the routing network as outlined above is often based on a priori assumptions regarding the objectives. A routing network is used to simplify the problem, without making too much concessions regarding the ob-

jective of the problem.

Data aggregation
Because most energy is used in data communication, and efficient energy use is an important objective in wireless networks, there has been extensive research in methods which decrease overall data communication. These methods include sleeping of nodes, and data aggregation. In this thesis we consider data aggregation as a technique to reduce energy.

Data aggregation consists of aggregating redundant or correlated data at a node thus reducing the number of data packets sent, as well as the size of the data. Especially in sensor networks, data which is sensed in the same region is often correlated or even redundant [3, 36, 70]. E.g. multiple sensors may sense the same activity, leading to highly correlated data, or even exact copies of the data. Data aggregation is also possible if data can be described by a single value, e.g. when the required data is an extreme value such as maximum temperature.

Data aggregation reduces the network traffic and hence directly reduces energy use. Also, it may reduce energy use indirectly, because sending less data packets reduces the possibilities of collisions, which leads to a reduction in energy use for retransmissions [33].

Most literature on data aggregation assumes *total aggregation*. In total aggregation we assume packets have the same size, and aggregation of two or more packets at a node results in a *single packet* of the same size being sent from this node. Even if the assumption of total aggregation may be considered rather simplistic, it allows us to derive an upper bound on the gains of data aggregation in terms of energy use.

Under total aggregation any two messages can be aggregated into a single packet regardless of their release times and release nodes. In practice total aggregation is not always possible; whether messages can be aggregated depends on the data.

Two generalizations which model these limitations are partial aggregation and geographically bounded aggregation. In partial aggregation the aggregation of packets reduces the data size to a certain extent. The function which models to what extent packets can be aggregated is typically a concave non-decreasing function [40]. In the geographically bounded total aggregation model we take into account the position of the nodes from where the data originates. In practice, the aggregation of packets can be subject to geographical constraints. In particular, it may be infeasible to aggregate two messages originating from sensors that are too far apart.

Data aggregation is usually performed in routing networks which are trees, because trees are natural structures to collect data at a single point.

2.3 Problems and objectives

The main purpose of wireless networks is to establish efficient communication between nodes. In the literature there exist many classes of wireless communication problems, and many different measures of efficiency. In this paragraph we describe

several problem classes, also called information dissemination processes, and we describe several objectives or quality of service (QoS) measures. The problems that we consider focus on routing and scheduling of messages, hence the problems address cross-layer issues from the data link layer and the network layer.

The information dissemination processes that we consider are broadcasting, gossiping, gathering and packet routing. A problem instance using one of these processes typically uses the process name to describe the problem, e.g. we speak of a gathering problem.

We give definitions of the information dissemination processes:
Broadcasting is the process of dissemination of data from a source node to all other nodes in the network.
Gossiping is the process of dissemination of data from each node in the network to every other node in the network.
Gathering is the process of dissemination of data from nodes in the network to a sink node.
Packet routing is the process of dissemination of data between nodes.

Dissemination processes can be characterized in several ways depending on the data characteristics. Common characterizations are data distribution and data destination. The data distribution can be either uniform, if the amount of data which is sent and/or received by each node is in the same order of magnitude, or non-uniform otherwise. The data destination can be either a single node, or multiple nodes. In case of personalized data there is a single receiver node, and in case of non-personalized data there are multiple receiver nodes. Typically, broadcasting and gossiping problems are related to uniform non-personalized data and employ the broadcasting mode of communication. Gathering and packet routing problems are related to non-uniform personalized data, and employ point-to-point communication. Note that gathering of non-personalized uniform data is equivalent to broadcasting, except for the reverse role of sender and receiver nodes.

The main objective in wireless communication is to establish efficient communication, where efficiency can be measured in several QoS measures. We consider the QoS measures time and energy use. There are also other QoS measures such as throughout, connectivity, security, and network reliability but we do not discuss these in this thesis.

In *time efficient* communication we are interested in minimizing the time between sending and receiving data. The motivation of this objective lies in the fact that the value of information degrades over time; especially in time-critical communication such as video-streaming or phone calls we prefer no delay at all between sending and receiving data. Therefore, in applications typically the delay of a message is bounded. E.g., in the TCP model, there is a bound on the maximum allowed message delay, called the time-to-live (TTL) [76].

We distinguish between the *latency* of a data packet, and the *delay* of a data packet. The latency of a data packet is the difference between the time the data becomes available at the sender node, and the time the data arrives at the receiver node, hence it can be viewed as an absolute measure. The latency of a packet is also called flow time; another related absolute time measure is the completion time

of a packet, the time the packet arrives at the sink. The delay of a data packet is a relative measure: we assume that each packet requires a certain amount of time to be sent in ideal conditions. The delay is the extra time a data packet requires, due to either the choice of algorithm or the information regime.

In *energy efficient* communication we are interested in minimizing the energy used to establish communication. Especially in networks where the nodes are battery operated, such as in sensor networks, energy efficiency is the main objective. This is because in this case an inefficient use significantly reduces the network lifetime, which is the time that all nodes in the network are active. We distinguish two energy objectives: minimizing average energy costs per node, and minimizing maximum energy costs per node. The first objective is more relevant in a setting where nodes can be recharged, and we have an interest in minimizing overall energy use. The second objective is more relevant in a setting where nodes can not be recharged, such as in sensor networks. In this case minimizing the maximum energy costs, is equivalent to maximizing the network lifetime.

The different QoS measures are in conflict with one another. Typically, faster communication requires the use of extra data packets, or simply communication at a higher speed which results in an increased energy use. It depends on the application of the problem which of the measures is chosen as objective, and how the other measures are modeled in the problem. Typically, the other measures are modeled as constraints, although it is also possible to optimize several objectives simultaneously using multicriteria optimization.

2.4 Problems covered in this thesis

In this thesis we focus on gathering problems with data aggregation, and with interference. In Chapters 3 and 4 we study a gathering problem using data aggregation. We study to what extent data aggregation can be used to establish energy efficient communication. In Chapter 3 the objective is to minimize energy use of the nodes, and we consider both average energy use and maximum energy use. We impose hard constraints on the message latencies. In Chapter 4 we study the same problem in a bicriteria setting, with objectives to minimize energy use and latency costs. In Chapters 5 and 6 we study a gathering problem under interference. We study how we can obtain efficient communication within an interference model. In Chapter 5 we analyze the problem minimizing completion times. In Chapter 6 we analyze the problem minimizing flow times.

2.5 Summary

Wireless networks have become an important means of communication, and the use of wireless networks is likely to increase in future years. Main challenge in wireless networks is to establish efficient communication between users. There are many classes of wireless networks depending on network architecture, and the purpose of

the network. In this thesis we study several combinatorial optimization problems in classes of wireless networks.

We study problems in networks which use an omnidirectional radio antenna to communicate data over a broadcast channel. We consider a class of problems, called gathering problems. In gathering problems data has to be communicated from nodes in the network to a central node, the sink. Data gathering is a fundamental problem in communication networks, and also used as a sub-problem in other communication problems.

The algorithms for data gathering address scheduling and routing of messages. We assume that necessary information such as a communication network is provided by other algorithms. The algorithms that we propose can be implemented in protocols for the data link layer, and the network layer, two design layers of the OSI reference model, a standard model to organize software.

We consider gathering problems in a wireless network, under the assumption of interference, and we use data aggregation as a technique to reduce data communication. We consider several quality of service measures to assess the efficiency of out algorithms. These measures are message communication times and node energy use.

2.6 Related literature

General information on wireless networks can be found in the handbooks of Boukerche [21], Ilyas and Mahgoub [47], and Pahlavan and Levesque [63], as well as in the surveys on wireless sensor networks of Akyildiz et al. [3], and Ganesan et al. [36]. Schmid and Wattenhofer [73] give an overview of different models for wireless sensor networks. The information dissemination processes that we described have previously been studied in wired networks. Overviews of these processes and algorithms for these processes are given in Fraigniaud and Lazard [35], in Hedetniemi et al. [42], and in Hromkovič et al. [46]. An early account of the issue of time synchronization has been given in [56]. [61] describes the Network Time Protocol, a synchronization model for distributed networks, and a standard model for time synchronization in the Internet. General time synchronization methods are discussed in [30, 31, 58]. Time synchronization in ad hoc and sensor networks is addressed in [32] and [72, 77] respectively. Data aggregation in sensor networks is discussed in [3, 36, 70].

Chapter 3

Data aggregation with hard latency constraints

3.1 Introduction

In the following two chapters we study a gathering problem in a wireless network where we use data aggregation to reduce the network energy use. We call this problem the data aggregation problem (DAP).

In DAP we consider a wireless network where data is released over time at the nodes, and the nodes should communicate all data to the sink using multi-hop communication. Nodes may delay messages in order to aggregate multiple messages into a single packet, and forward this packet to the sink. This aggregation reduces the communication costs at the expense of an increased message latency. The objective of DAP is to minimize both the message latencies, and the network communication costs, i.e. the energy use of the nodes.

In this chapter we study the latency constrained data aggregation problem in which the objective is to minimize the energy use, under constraints on the maximum message latency. In the next chapter we study the bicriteria data aggregation problem in which we use bicriteria optimization to simultaneously minimize the two objectives mentioned above. A motivation for the problems can be found in Chapter 2.

Mathematically, the latency constrained data aggregation problem (LDAP) is the following. We are given a routing tree $D = (V, A)$ of a wireless network rooted at a *sink node* $s \in V$. Nodes represent stations and arcs represent the possibility of communication between two stations. The arcs are oriented towards the sink.

Over time m messages arrive at nodes and have to be sent to the sink. Let $M = \{1, \ldots, m\}$ be the set of these messages. Message j arrives at its *release node* v_j at its *release date* r_j and must arrive at the sink via the unique $v_j - s$-path. Message j must arrive at the sink before a specified time, its *due date* d_j.

We assume messages to have the same size. This assumption is often made in

models of wireless communication, as typically a wireless network protocol divides all data into messages of similar size. To summarize, for LDAP a message j can be characterized by the triple (v_j, r_j, d_j). We use both the terms time and date to refer to points in time.

A *packet* is a set of messages, that are sent simultaneously along an arc. Two messages j and j' can be aggregated at a node v. The resulting packet has due date $d = \min\{d_j, d_{j'}\}$. This definition naturally extends to the case when more messages are aggregated into a single packet.

Communication of a message along an arc takes time and requires energy (cost). For most part of this chapter we assume that the communication time $\tau : A \to \mathbb{R}_{\geqslant 0}$ and communication cost $c : A \to \mathbb{R}_{\geqslant 0}$ are independent of packet size. In this case, we often refer to the *communication cost* $c(v)$ of a node v as the communication cost of its unique outgoing arc. This models the situation in which all messages have more or less the same size and where *total aggregation* is possible, as discussed in Subsection 2.2.2. We assume that the communication time to send a packet over an arc is identical for all arcs, and choose $\tau(a) = 1$ for each arc a. This models the situation, where the distance between any pair of nodes in the routing tree is more or less equivalent. For $v \in V$, let τ_v be the total communication time on the path from v to s, i.e. the sum of communication times of all arcs on this path.

The *latency* of a message is the difference between its arrival time at the sink and its release time. Let C_j be the completion time of message j in a solution. We define $L_j := d_j - r_j$ as the maximum allowed latency of message j, and $l_j := C_j - r_j$ as the realized latency of message j. A message specific latency allows the modeling of messages with different priorities. The *delay* of a message is the difference between the latency of a message and the minimal required communication time to send the message to the sink. I.e. the delay of message j is $l_j - \tau_{v_j}$. Note that in case communication times are zero, the latency and the delay of a message are equal.

We consider as objective to minimize the maximum communication costs per node. This is a natural objective in sensor networks because of limited and unreplenishable energy at nodes. The due dates provide message specific bounds on the maximum allowed latency. In the next chapter we assume that these latency bounds are not strict.

We introduce some extra notation to facilitate the exposition of this chapter. The release and due date of message j define a time interval for this message to be in the network, $[r_j, d_j]$. They also define transit intervals and an arrival interval. For message j and node u on the path from v_j to s, we define *transit interval* $I_j(u)$ as the time interval during which message j can reside at node u: $I_j(u) := [r_j + \tau_{v_j} - \tau_u, d_j - \tau_u]$. In particular, $I_j(s) = [r'_j, d_j]$, where $r'_j := r_j + \tau_{v_j}$ the *earliest possible arrival time* of j at s. We abbreviate $I_j(s)$ to I_j and call it the *arrival interval* of message j. In order to satisfy the latency constraints message j should arrive at the sink at some time in interval I_j. We write $|I|$ for the length of interval I; note that for each message j we have $|I_j(u)| = |I_j|$ for all nodes u on the path from v_j to s. The length of this interval is the maximum allowed delay message j can incur in the network. Finally, we define $\delta := \max_v \tau_v$ as the depth of the network in terms

of the communication time. And let $U := \frac{\max_j |I_j|}{\max\{1,\min_j |I_j|\}} = \frac{\max_j (L_j - \tau_{v_j})}{\max\{1,\min_j (L_j - \tau_{v_j})\}}$, the ratio between the minimum and maximum of the maximum allowed delay.

We use complexity theory to assess LDAP, and we use concepts from approximation theory and competitive analysis to analyze the performance of algorithms that we propose for LDAP. See Chapter 1 for an introduction to these concepts.

In Section 3.2 we study the complexity of LDAP. We show that the offline version of LDAP is **NP**-hard. In Section 3.3 we present a 2-approximation for the offline version of LDAP. In Section 3.4 we study online algorithms under several information models. For the synchronous model we present an $O(\log U)$-competitive algorithm. We also show that no synchronous algorithm can be better than $\Omega(\log U)$-competitive, hence our algorithm is best possible up to a multiplicative constant. For the asynchronous model we present an $O(\delta \log U)$-competitive memoryless algorithm, and we show that no memoryless algorithm can be better than $\Omega(\delta)$-competitive. We end this section with an analysis of special instances of LDAP. We show that for a constant latency of at least two times the maximum communication time to the sink, our synchronous distributed algorithm is in fact constant competitive. We introduce a centralized algorithm for LDAP, and demonstrate that it is 2-competitive, in case of constant latency, and the network is a half-line, a line with the sink at the end. In Section 3.5 we study variations and generalizations of LDAP. We consider LDAP with objective minimizing the sum of energy. We indicate which results for LDAP minimizing maximum energy also hold for this objective. And we present a polynomial time algorithm which solves the offline version of LDAP minimizing the sum of energy, on a half-line. We show that our results also hold in case the energy use cost function is concave. And we generalize our model to accommodate geographically bounded aggregation of messages, i.e. of messages which are released within the same region only.

Related work

In the literature several problems have been studied which are related to DAP. The TCP acknowledgment problem is closely related to DAP. In this problem messages have to be gathered at a sink using a routing network which is a tree. Communication costs can be decreased using total aggregation, at the expense of message delays. The main difference with DAP is that in the TCP acknowledgment problem the single objective is to minimize both energy costs and message latencies. We believe such a single objective does not properly reflect the problem objective, because time and energy are measured in different metrics. A second difference is that the problem focuses on minimization of total energy costs and minimization of total message delays whereas the focus of DAP is on minimizing maximum energy costs and maximum delays.

We give a brief overview of results for the TCP acknowledgment problem. Dooly et al. [28] presented a distributed 2-competitive algorithm, in case the network is a single edge, and communication times are zero. Karlin et al. [50] presented a randomized distributed $\frac{e}{e-1}$ -algorithm for the same model. The general case on a routing tree with communication times zero and communication costs $c(v) \geq 1$

has been studied by several authors. Let for this version δ be the diameter of a graph. Khanna et al. [52] presented an $O(\delta \sum_v c(v))$-competitive algorithm for the synchronous distributed model. Brito et al. [22] presented an $O(\chi^*(D))$-competitive algorithm for the asynchronous distributed model on graph D. Here $\chi^*(D) \leq \max_{p \in D} \sum_{v \in p} c(v)$, where p is a path in D.

Albers et al. [4] considered the same model as Dooly et al., but instead of minimizing the sum of delays they minimized the maximum delay, as in DAP. They presented a $\frac{\pi^2}{6}$-competitive deterministic online algorithm in case the graph is a single edge. In their algorithm the i-th packet sent incurs a delay which is iz for some constant z, i.e. the delay of a packet increases linearly with the number of packets. In our opinion it seems rather unrealistic that solutions, in which messages incur delays which increase over time, are of interest from a practical point of view.

3.2 Complexity

In this section we analyze the complexity of LDAP. We prove that this problem is **NP**-hard. But first we prove some properties of optimal offline solutions.

Lemma 3.1. *There exists an optimal solution to* LDAP *such that:*

(i) whenever two messages are present together at the same node, they stay together until they reach the sink;

(ii) a message never waits at an intermediate node, i.e., a node different from its release node and the sink;

(iii) the time when a packet of messages arrives at the sink is the earliest due date of any message in that packet.

Proof. (i): Repeatedly apply the argument that whenever two messages are together at the same node but split up afterwards, keeping the one arriving later at the sink with the other message does not increase cost.

(ii): Use (i) and repeatedly apply the following argument. Whenever a packet of messages arrives at an intermediate node and waits there, changing the solution by shifting this waiting time to the tail node of the incoming arc does not increase cost.

(iii): Follows similarly as (ii) by interpreting the time between the arrival of a packet at the sink and earliest due date as waiting time. □

The lemma provides properties for *some* optimal offline solution. In fact, all algorithms we present for LDAP in this chapter construct solutions which satisfy property (i), and it seems reasonable to do so. It may also seem that if we construct algorithms which produce solutions that satisfy all properties of Lemma 3.1, we obtain algorithms with a good approximation ratio. However, this idea is false. In Subsection 3.4.2 we present an algorithm for the asynchronous time model which generates solutions that do not satisfy property (ii), and we demonstrate that the

competitive ratio of this algorithm is strictly better than distributed asynchronous algorithms which produce solutions that do satisfy property (ii).

Now, we turn to the complexity analysis of LDAP.

Theorem 3.2. LDAP *is* **NP***-hard.*

Proof. We prove the theorem using a reduction from the satisfiability problem (SAT) [37]. Given an instance of SAT with m Boolean variables $X_1, \ldots, X_m$ and k clauses $Y_1, \ldots, Y_k$, we construct the intree on $m + 2$ nodes depicted in Figure 3.1.

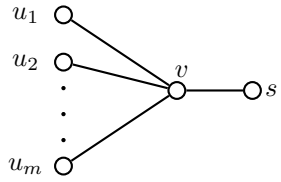

Figure 3.1: An example of a wireless network.

The nodes $u_1, \ldots, u_m$ on the left correspond to variables $X_1, \ldots, X_m$. There is one intermediate node v and the sink s on the right. The communication costs of the arcs are determined later. To facilitate the exposition we assume that the communication times of all arcs are zero, whence the earliest arrival times of the messages coincide with their release dates. The proof can also be made to hold for arbitrary constant communication times.

For clause Y_i we define a time interval $T(Y_i) = [3i(m+1), 3(i+1)(m+1) - 1]$ and a message $y_i = (v, r_i, d_i) := (v, (3i+1)(m+1), (3i+2)(m+1))$, $i = 1, \ldots, k$. Notice that the arrival interval $I_{y_i} = [r_i, d_i] \subset T(Y_i)$. We also define two dummy messages $y_0 := (v, 0, 0)$ and $y_{k+1} := (v, 3(k+1)(m+1), 3(k+1)(m+1))$. Notice the crucial fact $|I_{y_0}| = |I_{y_{k+1}}| = 0$, leaving no choice in sending y_0 and y_{k+1}.

If variable X_j occurs unnegated in clause Y_i, we create a message $x_i^j = (u_j, r_i^j, d_i^j)$ $:= (u_j, (3i+1)(m+1)+j, (3i+2)(m+1)+j)$. If X_j occurs negated in clause Y_i, we create message $x_i^j := (u_j, 3i(m+1)+j, (3i+1)(m+1)+j)$. If X_j does not occur in Y_i no message x_i^j is created. Notice that in both cases the arrival time interval $I_{x_i^j} \subset T(Y_i)$. If X_j does not occur in Y_i no message x_i^j is created. An illustration is given in Figure 3.2 on the next page.

The idea behind the reduction is the following: in an optimal solution, message x_i^j is either sent at its release or at its due date (the reason for this will become clear later). Moreover, sending x_i^j at its release (due) date means setting X_j to true (false). Thus, message y_i can join message x_i^j at node v if and only if the value of variable X_j makes clause Y_i true.

We continue with the description of the instance. Let $i_1^j < \cdots < i_{k_j}^j$ denote the indices of the clauses in which variable X_j occurs. We create $k_j + 1$ additional messages released at node u_j. The release and due dates of these messages are chosen such that the $2k_j + 1$ arrival time intervals formed by the release and due dates of

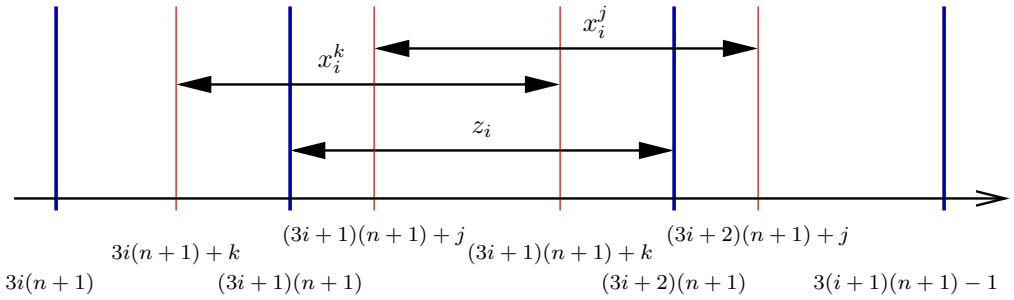

Figure 3.2: Arrival intervals corresponding to clause Y_i. In the depicted example, the clause has the form $Y_i = (X_j \vee \neg X_h)$.

all messages released at node u_j form a partition of the interval $[0, 3(k+1)(m+1)]$; see Figure 3.3.

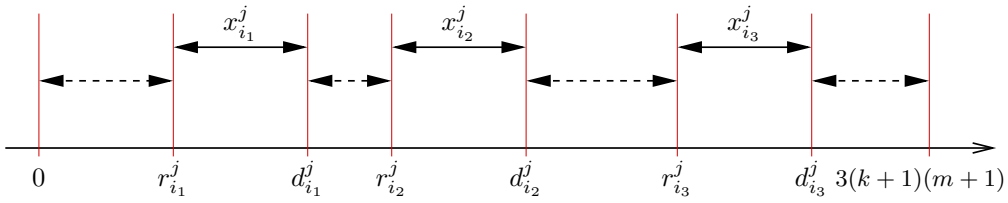

Figure 3.3: Arrival intervals of messages with release node u_j. In the depicted example, variable X_j occurs in three clauses Y_{i_1}, Y_{i_2}, and Y_{i_3}. Arrival intervals of the four auxiliary messages are represented by dashed arrows.

We will demonstrate that for appropriately chosen cost functions any instance of SAT reduces to an instance of LDAP. We define the cost function by $c(u_j, v) = (\max_l k_l + 1)/(k_j + 1)$ for $j = 1, \ldots, m$, and $c(v, s) = (\max_l k_l + 1)/(\sum_{l=1}^{m} k_l + 2)$. The proof is based on two claims.

Claim 1. Every optimal solution to the sub-instance obtained by ignoring messages $y_1, \ldots, y_k$ has the following properties:

(a) The cost of each node is $\max_l k_l + 1$;

(b) A message with release node u_j is either sent from u_j at its release date or at its due date, $j = 1, \ldots, m$;

(c) For each fixed $j = 1, \ldots, m$, either all messages x_i^j $(i = i_1^j, \ldots, i_{k_j}^j)$ are sent at their release dates or all of them are sent at their due dates.

Proof of Claim 1. Let us consider a solution which minimizes the cost of nodes u_j. Since the $2k_j + 1$ arrival time intervals of messages with release node u_j form a partition of $[0, 3(k+1)(m+1)]$, at most one x_i^j-message and one of the auxiliary

messages can be aggregated into a packet, which then has to be sent at the single intersection point of the two arrival time intervals. Thus the minimal number of packets that have to be sent from node u_j is k_j+1, i.e. k_j pair-packets and one packet containing a single message. Hence the minimal cost of node u_j is $c(u_j, v)(k_j + 1) = \max_l k_l + 1$ for all nodes u_j.

Each pair-packet is sent at the common release and due date of its two messages and by construction of these dates no two pair-packets emanating from different nodes can be aggregated into a single packet at node v.

Also, note that as a consequence pair-packets are not delayed at node v. Thus, there are $\sum_{l=1}^{m} k_l$ pair-packets passing v. And we have the two dummy messages y_0 and y_{k+1}, which are sent from node v at times 0 and $3(k + 1)(m + 1)$. Pair-packets cannot be aggregated with these dummy messages but each single-message-packet can be sent at time 0 or $3(k+1)(m+1)$ and hence it may join dummy message y_0 or y_{k+1} at node v. This gives a total of $\sum_{l=1}^{m} k_l + 2$ packets passing v. Thus, the cost of node v is $c(v, s)(\sum_{l=1}^{m} k_l + 2) = \max_l k_l + 1$. Notice that a single-message-packet contains either the first or the last auxiliary message released at node u_j. If the single-message-packet is the first auxiliary message then all pair-packets are sent on the due date of the x_i^j-message in the packet. Otherwise, all pair-packets are sent on the release date of the x_i^j-message in the packet.

Thus, we have constructed a solution which satisfies properties (a),(b) and (c). As the cost of node u_j is at least $\max_l k_l + 1$ the solution is an optimal solution. From the construction of this solution it can easily be verified that any solution which violates property (a),(b) or (c) has a node with a cost which exceeds $\max_l k_l + 1$. □

This claim suffices to prove the following claim.

Claim 2. LDAP has a solution with maximum cost at most $\max_l k_l + 1$ if and only if the underlying instance of SAT is satisfiable.

Proof of Claim 2. Given a satisfying assignment for the SAT instance, we can obtain a feasible solution to LDAP as follows. Notice that in the construction of an optimal solution in the proof of Claim 1, for each j, there is a choice for the set of messages corresponding to X_j, to send either dummy message y_0 separately at time 0 or the dummy message y_{k+1} separately at time $3(k + 1)(m + 1)$. In both cases the cost of sending all messages corresponding to the variables and y_0 and y_{k+1} is $\max_l k_l + 1$ for each node. We make the choice now by sending y_{k+1} separately if X_j is true in the satisfying assignment and y_0 separately if X_j is false.

We claim that message y_i corresponding to clause Y_i, $i = 1, \ldots, k$, can be aggregated at v with one of the pair-packets corresponding to a variable in the clause. Suppose that clause Y_i is satisfied due to variable X_j. If X_j appears unnegated in Y_i (thus X_j is true), then the pair-packet containing message x_i^j is sent at time $r_i^j := (3i + 1)(m + 1) + j \in I_{y_i}$, the arrival interval of message y_i. Hence, message y_i can join this packet at no additional cost. Similarly, if X_j appears negated at Y_i (thus X_j is false) then x_i^j is sent at $d_i^j := (3i + 1)(m + 1) + j \in I_{y_i}$. This concludes the proof of the "if" part.

It follows from Claim 1 that any feasible solution with maximum cost $\max_l k_l + 1$ yields an assignment of values to the Boolean variables $X_1, \ldots, X_m$: variable X_j is set to true (false), if all messages x_i^j, $i = i_1^j, \ldots, i_{k_j}^j$, are sent at their release (due) dates. It also follows from Claim 1 that in an optimal solution message y_i, $i = 1, \ldots, k$ should not cause additional cost, therefore it must join one of the packets starting at a node u_j. Due to the construction of the instance, this is only possible if the value of variable X_j causes clause Y_i to be satisfied. This concludes the proof of the "only if" part. □

The theorem follows directly from Claim 2. □

3.3 A constant approximation algorithm

We formulate LDAP as an integer linear programming problem. We design a novel rounding technique for the LP-relaxation of this problem, and show that a solution to this problem yields a 2-approximation for offline LDAP.

First, we derive the integer programming formulation. Assume message set M is ordered by increasing due date, i.e. $d_1 \leq \ldots \leq d_m$. For each message-node-pair $\{i, v\}$ we introduce a binary decision variable x_{iv}, which is set to 1 if and only if node v sends some message j which arrives at s in a packet with message i at time d_i. We use the notation $j_{\min}$ for the smallest index i such that $d_i \geq r'_j$; i.e., $j_{\min} := \min\{i : d_i \geq r'_j\}$. Similarly we use the notation $j_{\max}$ for the largest index i such that $d_i \leq d_j$; i.e., $j_{\max} := \max\{i : d_i \leq d_j\}$. The integer programming problem formulation of LDAP is

$$\begin{array}{ll} \min & z \\ \text{s.t.} & z \geq c(v) \sum_{i=1}^{m} x_{iv} \quad \forall v \in V, \\ & \sum_{i=j_{\min}}^{j_{\max}} x_{iv_j} \geq 1 \quad \forall\, 1 \leq j \leq m, \\ & x_{iv} \geq x_{iu} \quad \forall\, 1 \leq i \leq m \;\forall\, (u,v) \in A\ , \\ & x_{iv} \in \{0,1\} \quad \forall\, 1 \leq i \leq m \;\forall\, v \in V. \end{array} \tag{3.1}$$

The set of constraints $\sum_{i=j_{\min}}^{j_{\max}} x_{iv_j} \geq 1$ forces each message to leave its release node in time to reach the sink before its due date. The set of constraints $x_{iv} \geq x_{iu}$ is sufficient to ensure that a message does not have to wait at intermediate nodes. In fact, the formulation also holds for arbitrary communication times $\tau(a)$.

The following lemma demonstrates the equivalence of the integer program and LDAP. The lemma is based on the observation that a solution x to (3.1) can be transformed to a feasible solution to LDAP.

Lemma 3.3. *An optimal solution to* (3.1) *yields an optimal solution to* LDAP.

Proof. Given is an instance of LDAP. Let S and S^* be feasible solutions to LDAP which satisfy the properties of Lemma 3.1. We prove the lemma by demonstrating that each solution S of maximum energy cost Z, is equivalent to a feasible solution (x, z) of (3.1) of value $z = Z$, and that each optimal solution (x^*, z^*) of (3.1) of

value $z^* = Z$ is equivalent to a feasible solution S^* of maximum energy cost Z. The proof then follows from the fact that there exists an optimal solution to LDAP which satisfies the properties of Lemma 3.1.

Suppose we are given a solution S of cost Z. Then, it follows from property (iii) of Lemma 3.1 that each packet arrives at the due date of some message. Hence, solution S directly corresponds to a binary solution vector x, where x is defined as above. Because S has cost Z we must have $\max_v c(v) \sum_{i=1}^{m} x_{iv} = Z$ thus $z \geq Z$. As S satisfies property (ii) of Lemma 3.1 we must have $x_{iv} \geq x_{iu}$, and as each message in LDAP is sent to the sink before its due date in S we have $\sum_{i=j_{\min}}^{j_{\max}} x_{iv_j} \geq 1$, hence (x, Z) is a feasible solution to (3.1). This proves the first part of the claim.

Suppose we are given an optimal solution (x^*, z^*) of the integer program (3.1) of cost $z^* = Z$. Then, it follows from equations $x^*_{iv} \geq x^*_{iu}$ that each message which is sent from its release node, can be forwarded to the sink, without being delayed at other nodes. Also, from equations $\sum_{i=j_{\min}}^{j_{\max}} x^*_{iv_j} \geq 1$ it follows that the release node of each message j sends a packet such that j can be aggregated and sent with this packet, and arrives at the sink before its due date. We assume that a packet sent contains all messages present at this node. Then, it follows from the observations above that solution x^* is equivalent to a feasible solution S^* of cost Z. This proves the second part of the claim. □

We obtain the LP-relaxation of the integer program of (3.1) by replacing the last set of constraints with $0 \leq x_{iv} \leq 1 \;\; \forall\, 1 \leq i \leq m \;\forall\, v \in V$. We use the following technique to round fractional solutions to integral solutions.

Algorithm 1 GreedyRounding

Let $\alpha_1, \ldots, \alpha_m \in \mathbb{R}_{\geqslant 0}$ and $\beta_1, \ldots, \beta_m \in \{0, 1\}$ with

$$\sum_{i=j}^{k} \alpha_i \geq 1 \quad \Longrightarrow \quad \sum_{i=j}^{k} \beta_i \geq 1 \quad \forall 1 \leq k \leq m \;\forall 1 \leq j \leq k. \tag{3.2}$$

Consider the β_i's in order of increasing index. If $\beta_i = 1$, then round it down to 0, unless this yields a violation of (3.2). Let the resulting vector be $\bar{\beta}_1, \ldots, \bar{\beta}_m$.

It is straightforward to see that the rounding procedure GreedyRounding requires time linear in the size of vectors, m.

Lemma 3.4. *Given α and β* GreedyRounding *yields a resulting vector $\bar{\beta}$ which satisfies property* (3.2) *and the inequality*

$$\sum_{i=1}^{m} \bar{\beta}_i \leq 2 \sum_{i=1}^{m} \alpha_i \tag{3.3}$$

Proof. We are given vectors α, β and a vector $\bar{\beta}$ which results from GreedyRounding. By definition of the rounding procedure $\bar{\beta}$ satisfies property (3.2). It remains to prove that inequality (3.3) holds for the resulting numbers $\bar{\beta}_1, \ldots, \bar{\beta}_m$.

For $h \in \{1, \ldots, m\}$, let $\bar{h} := \min\{i > h \mid \bar{\beta}_i = 1\}$; if $\bar{\beta}_i = 0$ for all $i > h$ or $h = m$, then $\bar{h} := m + 1$. Similarly, let $\underline{h} := \max\{i < h \mid \bar{\beta}_i = 1\}$; if $\bar{\beta}_i = 0$ for all $i < h$ or

$h = 1$, then $\underline{h} := 0$. We prove the following generalization of (3.3):

$$\sum_{i=1}^{h} \bar{\beta}_i \leq \; 2 \sum_{i=1}^{\bar{h}-1} \alpha_i \quad \forall\, 1 \leq h \leq m. \tag{3.4}$$

By contradiction, consider the smallest index h violating (3.4). Since h is chosen minimally, it must hold that $\bar{\beta}_h = 1$; rounding $\bar{\beta}_h$ down to 0 would yield a violation of (3.2). In particular this would yield

$$\sum_{i=\underline{h}+1}^{\bar{h}-1} \alpha_i \geq \; 1 \tag{3.5}$$

while $\sum_{i=\underline{h}+1}^{\bar{h}-1} \bar{\beta}_i = 0$. Notice that $\underline{h} \geq 1$, i.e. there is an index $i < h$ for which $\bar{\beta}_i = 1$, since, by choice of h,

$$\sum_{i=1}^{h} \bar{\beta}_i \;>\; 2 \sum_{i=1}^{\bar{h}-1} \alpha_i \overset{(3.5)}{\geq} \; 2.$$

Thus, $\bar{\beta}_{\underline{h}} = \bar{\beta}_h = 1$. We get a contradiction to the choice of h:

$$\sum_{i=1}^{h} \bar{\beta}_i \;= \sum_{i=1}^{\underline{h}-1} \bar{\beta}_i + 2 \overset{(3.4)}{\leq} \; 2 \sum_{i=1}^{\underline{h}-1} \alpha_i + 2 \overset{(3.5)}{\leq} \; 2 \sum_{i=1}^{\underline{h}-1} \alpha_i + 2 \sum_{i=\underline{h}+1}^{\bar{h}-1} \alpha_i \leq \; 2 \sum_{i=1}^{\bar{h}-1} \alpha_i \;.$$

The first inequality follows from (3.4) since $\overline{(\underline{h}-1)} = \underline{h}$. □

Now, we are in position to present an algorithm for LDAP, which uses the LP-relaxation derived from (3.1) and the GreedyRounding technique.

Algorithm 2 LPRounding

Solve the LP-relaxation of (3.1) to obtain fractional solution (x, z).
Consider the arcs in order of non-decreasing distance from s.
for each node u **do**
  if $(u, s) \in A$ **then**
    set $\hat{x}_{iu} := 1 \; \forall \; i = 1, \ldots, m$
  else
    set $\hat{x}_{iu} := \bar{x}_{iv} \; \forall i = 1, \ldots, m$, and unique node $\{v | (u, v) \in A\}$
  end if
  Use GreedyRounding on values $x_{1u}, \ldots, x_{mu}$ and $\hat{x}_{1u}, \ldots, \hat{x}_{mu}$ to obtain $\bar{x}_{1u}, \ldots, \bar{x}_{mu}$.
end for
Let $\bar{z} := 2z$. This results in integral solution $(\bar{x}, \bar{z})$.

Here, the distance function of a node v is defined as the cardinality of the unique $v - s$ path.

Theorem 3.5. LPRounding *is a polynomial time* 2*-approximation algorithm for* LDAP*.*

Proof. LPRounding first solves the LP-relaxation of (3.1) to obtain optimal fractional solution (x, z). This can be done in polynomial time, see Chapter 1. Then, LPRounding uses GreedyRounding to round down arcs to an integral solution $(\bar{x}, \bar{z})$, as follows. For each node u the α's are $x_{1u}, \ldots, x_{mu}$, and the β's are $\hat{x}_{1u}, \ldots, \hat{x}_{mu}$. It is easy to see that premise (3.2) of Lemma 3.4 is satisfied for node u adjacent to s. We use induction to show that premise (3.2) of Lemma 3.4 holds for each node u. Suppose it holds for each node at distance at most d from the sink. Consider a node u at distance $d+1$ from the sink. We have $(u, v) \in A$ for some node v at distance d from the sink. Using induction and Lemma 3.4 we have that (3.2) holds for $x_{1v}, \ldots, x_{mv}$ and $\bar{x}_{1v}, \ldots, \bar{x}_{mv}$. Then, since $x_{iu} \leq x_{iv}$, the premise (3.2) of Lemma 3.4 is also satisfied for $x_{1u}, \ldots, x_{mu}$ and $\hat{x}_{1u}, \ldots, \hat{x}_{mu}$. Thus it follows from Lemma 3.4 that $\sum_{i=1}^{m} \bar{x}_{iu} \leq 2 \sum_{i=1}^{m} x_{iu}$ for each node u, hence the final solution $(\bar{x}, \bar{z})$ is feasible if $\bar{z} \geq 2z$. □

3.4 Online algorithms

In this section we analyze LDAP in an online distributed model using competitive analysis. We consider both the synchronous time model, and the asynchronous time model as defined in Subsection 2.2.2.

In both time models we assume that each node v knows its total communication time τ_v to the sink. This assumption is not unreasonable in a latency constrained model, because without this knowledge nodes can not even identify when messages exceed their due dates.

3.4.1 A synchronous distributed algorithm

We present an algorithm for the synchronous distributed model, and we prove that this algorithm is best possible, up to a multiplicative constant, among all deterministic algorithms for LDAP, using this time model.

Lemma 3.6. *Given any interval* $[a, b]$*,* $a, b \in \mathbb{N}$*, . Let* $i^* = \max\{i \in \mathbb{N} \mid \exists k \in \mathbb{N} : k2^i \in [a, b]\}$*, then* k^* *for which* $k^*2^{i^*} \in [a, b]$ *is odd and unique.*

Proof. Assume that $k_1 2^{i^*} \in [a, b]$ and $k_2 2^{i^*} \in [a, b]$, with $k_1 < k_2$. We may assume that $k_2 = k_1 + 1$, for if $k_2 > k_1 + 1$, then obviously also $(k_1 + 1)2^{i^*} \in [a, b]$. This means that either k_1 or k_2 is even. Suppose this is k_1 (if k_2 is even the arguments are analogous). Then, $k_1/2 \in \mathbb{N}$ and $(k_1/2)2^{i^*+1} \in [a, b]$, contradicting the definition of i^*. □

We use notation $t(I)$ to represent the unique point in the interval $I = [a, b]$ which equals $k^*2^{i^*}$ with i^* and k^* as defined in Lemma 3.6. The algorithm we present sends messages j to the sink at time $t(I)$ where interval I depends on message j and its message characteristics. The following algorithm uses as interval for message j its arrival interval I_j.

Algorithm 3 CommonClock

Message j is sent from v_j at time $t(I_j)-\tau_{v_j}$ to arrive at s at time $t(I_j)$ unless some other packet passes v_j in the interval $[r_j, t(I_j)-\tau_{v_j}]$, in which case j is aggregated and the packet is forwarded directly.

For a competitive analysis of CommonClock we first derive a bound on the competitive ratio of this algorithm for instances in which the arrival intervals I_j differ by at most a factor 2 in length. Let $M_i = \{j \in M, 2^{i-1} < |I_j| \leq 2^i\}$ for some $i \in \mathbb{N}$. Let $\mathcal{I} = \{\lfloor \log(\max\{1, \min_j |I_j|\}) \rfloor, \ldots, \lceil \log(\max_j |I_j|) \rceil\}$. Note that $\bigcup_{i\in\mathcal{I}} M_i = M$, and $|\mathcal{I}| = O(\log U)$, where $U = \frac{\max_j |I_j|}{\max\{1, \min_j |I_j|\}}$.

Lemma 3.7. CommonClock *is* 3*-competitive if* $M_i = M$ *for some* $i \in \mathcal{I}$.

Proof. We will prove that the communication cost of each node in the CommonClock-solution is at most 3 times the communication cost of this node in the optimal solution.

Assume that in an optimal solution packets arrive at s at times $t_1 < \cdots < t_\ell$. Let P_h^* be the packet arriving at time t_h at s. Since $t_h \in I_j \ \forall j \in P_h^*$ and $|I_j| \leq 2^i$ $\forall j$, we have $I_j \subset [t_h - 2^i, t_h + 2^i] =: I \ \forall j \in P_h^*$, and $|I| = 2 \cdot 2^i$. If $t_h = k2^i$ then in the CommonClock-solution all messages in P_h^* may arrive at s at times $t_h - 2^i$ or t_h. If $t_h \neq k2^i$ then I contains two different multiples of 2^i, say $k2^i$ and $(k+1)2^i$, such that $k2^i < t_h < (k+1)2^i$. In this case, since $|I_j| > 2^{i-1}$ $\forall j$, we have $\forall j \in P_h^*$ that $I_j \cap \{k2^i, k2^i + 2^{i-1}, (k+1)2^i\} \neq \emptyset$. Lemma 3.6 implies that in a CommonClock-solution every message $j \in P_h^*$ arrives at s at one of $\{k2^i, k2^i + 2^{i-1}, (k+1)2^i\}$.

Hence, $\forall h = 1, \ldots, \ell$, all messages in P_h^* arrive at s at at most 3 distinct time instants in the CommonClock-solution. CommonClock does not delay messages at intermediate nodes. This implies that the nodes used by messages in P_h^* are traversed by these messages at most 3 times in the CommonClock-solution, proving the lemma. □

Theorem 3.8. CommonClock *is* $\Theta(\log U)$*-competitive.*

Proof. CommonClock sends the messages in M_i, $i \in \mathcal{I}$, at a cost of no more than 3 times the optimum, by Lemma 3.7. Let $M_0 = \{j \in M, |I_j| = 0\}$ CommonClock sends the messages $j \in M_0$ at a cost equal to the optimal solution, because $|I_j| = 0$ for each $j \in M_0$, hence there is no choice in sending these messages. As $|\mathcal{I}| = O(\log \delta)$, and $\bigcup_{i\in 0\cup\mathcal{I}} M_i = M$ this proves $O(\log U)$-competitiveness.

To prove $\Omega(\log U)$-competitiveness, consider a half-line of 2^{m+1} nodes $u_1, \ldots, u_{2^{m+1}} = s$ for some $m \in \mathbb{N}$. Take $c(a) = 1$ $\forall a$. An adversary releases messages j, $j = 1, \ldots, m$, with $v_j = u_{2^j}$, $r_j = 0$, and $d_j = 2^{m+1} - 1$. Hence $r'_j = 2^{m+1} - 2^j = k2^j$ for some odd $k \in \mathbb{N}$ and $|I_j| = 2^j - 1$. Therefore, CommonClock makes each message j arrive at s at time r'_j, no two messages are aggregated, whereas in an optimal solution all messages are aggregated into a single packet arriving at s at time $2^{m+1} - 1$. Thus, the CommonClock solution has value m against an optimal value of 1. Notice that $U = 2^m - 1$ here. □

In fact, CommonClock also works in case of arbitrary communication times $\tau(a)$, and with the same bound on the competitive ratio [9]. The following theorem shows that CommonClock is best possible, up to a multiplicative constant.

Theorem 3.9. *Any deterministic synchronous algorithm is $\Omega(\log U)$-competitive.*

Proof. Consider an intree of depth $\delta = 2^{m+1}$ with m the number of messages, and where each node, except the leaves, has indegree m. For any online algorithm we will construct an adversarial sequence of m messages all with latency $L = \delta$, such that there exists a node at which the adversary can aggregate all messages in a single packet, but at which none of them is aggregated by the online algorithm. Using a similar argument as in the proof of Lemma 3.1 (i) the fact that all messages can be aggregated in a single packet implies that there exists a solution such that every node sends at most one packet, hence the cost of the adversarial solution is 1, whereas the cost of the online algorithm is m.

Fix any online algorithm. Given an instance of the problem, let $W_j(u)$ be the time interval message j spends at node u by application of the algorithm. We call this the waiting time interval of message j at node u, and we denote its length by $|W_j(u)|$. Note that $\sum_u |W_j(u)| \leq |I_j|$ for each message j. We notice that the waiting time of a message in a node can be influenced by the other messages that are present at that node or have passed that node before. Because the algorithm is distributed the waiting time of a message in a node is not influenced by any message that will pass the node in the future.

The adversary chooses the source node v_j with total communication time $\tau_{v_j} := \delta - 2^j$ from s, for $j = 1, \ldots, m$, so that $|I_j| = 2^j$. Thus, $U = 2^{m-1} = \delta/4$. The choice of the exact position of v_j and the release time r_j is made sequentially and, to facilitate the exposition, described in a backward way starting with message m. The proof follows rather directly from the following claim.

Claim. For any set of messages $\{k, \ldots, m\}$ the adversary can maintain the properties:
(i) all messages in $\{k, \ldots, m\}$ pass a path p_k with 2^k nodes;
(ii) $I_k(u) = \bigcap_{j \geq k} I_j(u)$ $\forall u \in p_k$;
(iii) if $k < m$, then $W_{k+1}(u) \bigcap I_k(u) = \emptyset$ $\forall u \in p_k$;
(iv) if $k < m$, then $W_i(u) \bigcap W_j(u) = \emptyset$ $\forall u \in p_k$, $i = k, \ldots, m$, $j > i$.

We notice that for any message j and any node u on the path from v_j to s, $W_j(u)$ may have length 0 but is never empty; it contains at least the departure time of message j from node u.

Note that properties (i) and (ii) for $k = 1$ imply that all messages can indeed be aggregated into one packet, hence as argued above, the adversarial solution has a cost of 1. Properties (iv) and (i) for $k = 1$ imply that the algorithm sends all messages separately over a common path with 2 nodes, yielding a cost of m. This proves the theorem. See Figure 3.4 on the next page for a visualization of the transit intervals and waiting time intervals for some node $u \in p_k$ as stated in the claim.

We prove the claim by induction. The basis of the induction, $k = m$, is trivially verified. Suppose the claim holds for message set $\{k, \ldots, m\}$ and p_k is the path

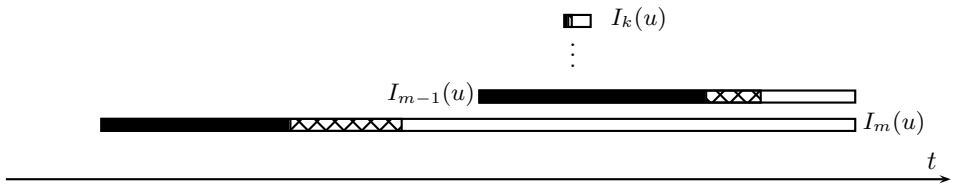

Figure 3.4: Transit intervals (I) and waiting time intervals (W) on some node $u \in p_k$ of the claim. The waiting time interval of a message before reaching u is solid, the waiting time interval of a message at u is dashed. In the depicted example messages k and m incur less than half of its total waiting time before reaching u, and message $m-1$ incurs more than half of its total waiting time before reaching u. The waiting time intervals of messages at node u (dashed) do not intersect, i.e. the algorithm cannot aggregate messages at u.

between nodes $\overline{v}$ and $\hat{v}$. We partition p_k into two sub-paths $\overline{p}$ and $\hat{p}$ consisting of 2^{k-1} nodes each, such that $\overline{v} \in \overline{p}$ and $\hat{v} \in \hat{p}$. We denote the node of $\overline{p}$ adjacent to $\hat{p}$ by $\overline{u}$ and the node of $\hat{p}$ adjacent to $\overline{p}$ by $\hat{u}$. See Figure 3.5 for a visualization of path p_k.

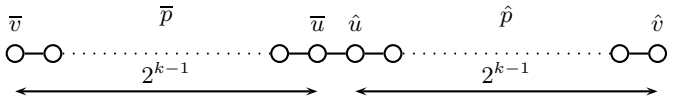

Figure 3.5: An example path p_k of length 2^k, as used in proof of the claim.

We distinguish two cases with respect to the waiting times the algorithm has selected for message k in the nodes on p_k.

Case a: $\sum_{u \in \overline{p}} |W_k(u)| \geq (1/2)|I_k|$. The adversary chooses v_{k-1} with total communication time $\tau_{v_{k-1}} = \delta - 2^{k-1}$ such that its path to s traverses $\hat{p}$ but not $\overline{p}$. More precisely, we ensure that the first node message $k-1$ has in common with any other message is $\hat{u}$. This is always possible, since the node degree is m. This choice immediately makes that setting $p_{k-1} = \hat{p}$ satisfies property (i). The release time of $k-1$ is chosen so that $I_{k-1}(\hat{u})$ and $I_k(\hat{u})$ start at the same time, implying that $I_{k-1}(u)$ and $I_k(u)$ start at the same time for every $u \in \hat{p}$. Since $|I_{k-1}(u)| = |I_k(u)|/2$ we have $I_{k-1}(u) \subset I_k(u)$ for all $u \in \hat{p}$, whence property (ii) follows by induction.

Note that, as we consider distributed algorithms, message $k-1$ does not influence the waiting time of $j, j > k-1$, on $\overline{p}$ as $\hat{u}$ is the first node which both j and $k-1$ traverse. In particular, $W_k(u), \forall u \in \overline{p}$ is not influenced by $k-1$

Now, the equal starting times of $I_{k-1}(\hat{u})$ and $I_k(\hat{u})$ together with $\sum_{u \in \overline{p}} |W_k(u)| \geq (1/2)|I_k|$ and $|I_{k-1}(\hat{u})| = |I_k(\hat{u})|/2$ imply that k will not reach $\hat{u}$ before interval $I_{k-1}(\hat{u})$ ends. This, together with the consideration above, implies property (iii).

To prove (iv), note that by induction it is sufficient to prove that $W_{k-1}(u) \cap W_j(u) = \emptyset \, \forall j > k-1 \; \forall u \in \hat{p}$. Since, as just proved, $W_k(u) \cap I_{k-1}(u) = \emptyset \; \forall u \in \hat{p}$ we have $W_{k-1}(u) \cap W_k(u) = \emptyset \; \forall u \in \hat{p}$. We have by induction that, for $j > k$, $W_j(u) \cap I_{j-1}(u) = \emptyset \; \forall u \in \hat{p}$ and we just proved that $I_{k-1}(u) \subset I_{j-1}(u) \subset I_j(u) \; \forall u \in \hat{p}$, which together imply $W_{k-1}(u) \cap W_j(u) = \emptyset \; \forall j > k \; \forall u \in \hat{p}$.

Case b: $\sum_{u \in \overline{p}} |W_k(u)| < (1/2)|I_k|$. As in the previous case, the adversary chooses v_{k-1} with total communication time $\tau_{v_{k-1}} = \delta - 2^{k-1}$ such that its path to s traverses $\overline{p}$ (therefore also $\hat{p}$) but does not intersect any of the paths used by messages $\{k, \ldots, m\}$ before it reaches $\overline{p}$ in $\overline{v}$. Again, this is always possible since the indegree of each node is m. Hence, choosing $p_{k-1} = \overline{p}$ satisfies property (i). The release time of $k-1$ is chosen so that $I_{k-1}(\overline{v})$ and $I_k(\overline{v})$ end at the same time, implying that $I_{k-1}(u)$ and $I_k(u)$ end at the same time for every $u \in \overline{p}$. Since $|I_{k-1}(u)| = |I_k(u)|/2$ we have $I_{k-1}(u) \subset I_k(u)$ for all $u \in \overline{p}$, whence property (ii) follows by induction.

The equal ending times of $I_{k-1}(\overline{u})$ and $I_k(\overline{u})$ together with $\sum_{u \in \overline{p}} |W_k(u)| < 1/2|I_k|$ and $|I_{k-1}(\overline{u})| = |I_k(\overline{u})|/2$ implies that k has left $\overline{u}$ before $I_{k-1}(\overline{u})$ begins, implying property (iii). Indeed, this gives $W_{k-1}(u) \cap W_k(u) = \emptyset, \forall u \in \overline{p}$. It also implies that $k-1$ could not influence the waiting time of k on $\overline{p}$.

The proof of (iv) follows the very same lines as in Case a, with the difference that we now refer to nodes in $\overline{p}$ instead of $\hat{p}$. □

Since in the proof $U = \delta/4$ we also have the following lower bound on the competitive ratio of any deterministic synchronous algorithm.

Corollary 3.10. *Any deterministic synchronous algorithm is* $\Omega(\log \delta)$*-competitive.*

3.4.2 An asynchronous distributed algorithm

We present an algorithm for the asynchronous distributed model. We analyze its competitive ratio, and we provide a lower bound on the competitive ratio for a broad class of algorithms which includes this algorithm.

The introduction of the asynchronous time model requires a more careful look at the concept of due dates and latency. Until now, we have assumed that each message is characterized by a due date, which implies the maximum allowed latency a message can incur before reaching the sink. The due date is an absolute time measure, whereas the latency is a relative time measure, which is related to the release time. In an asynchronous time model, nodes only have a notion of relative time, hence we assume in this case messages are characterized by their latency, rather than their due date. Because of the one-to-one relation between latency L_j and due date d_j, we can still refer to the due dates of messages in proofs, without loss of generality.

We propose algorithm SpreadLatency, which divides the maximum allowed latency minus communication time of each message j equally over the nodes on the $v_j - s$-path, except for the sink.

Algorithm 4 SpreadLatency

Let the waiting time of message j be $w_j = (L_j - \tau_{v_j})/\tau_{v_j}$ per node visited. Each node sends a packet as soon as the waiting time of some message at that node has elapsed. This packet contains all messages present at that node at that time.

By definition of the algorithm no message j incurs a total waiting time which exceeds $L_j - \tau_{v_j}$, message j has latency at most L_j, and the solution is feasible.

Theorem 3.11. SpreadLatency *is* $O(\delta \log U)$*-competitive.*

Proof. We prove that for all $v \in V$ the number of packets SpreadLatency sends from v is at most $O(\delta \log U)$ times that number in an optimal solution.

Let $\mu := \max\{1, \min_j(L_j - \tau_{v_j})\}$. Consider a packet P of messages sent by an optimal solution from node v at time t. Without loss of generality we do not consider messages for which $\min_j(L_j - \tau_{v_j}) = 0$ as these messages have to be sent upon release by both SpreadLatency and the optimal algorithm. To bound the number of packets sent by SpreadLatency that contain at least one message from P, define $P_i := \{j \in P \mid 2^{i-1}\mu \leq L_j - \tau_{v_j} < 2^i\mu\}$, for $i = 1, \ldots, \lceil \log U \rceil$. We *charge* any sent packet to the message that caused the packet to be sent due to its waiting time being over. It suffices to prove that the number of packets charged to messages in P_i is $O(\delta)$.

Since the waiting time of messages $j \in P_i$ at node v is at least $2^{i-1}\mu/\delta$, the delay between any two packets that are charged to messages in P_i is at least $2^{i-1}\mu/\delta$. Since the optimal solution sends packet P at time t from v, we get $t \in I_j(v)\ \forall j \in P$ and thus $I_j(v) \subseteq [t - 2^i\mu, t + 2^i\mu]\ \forall j \in P_i$. Thus, the number of packets charged to messages in P_i is at most $2 \cdot 2^i\mu/(2^{i-1}\mu/\delta) = 4\delta$. □

In fact, the following theorem demonstrates that the competitive ratio of SpreadLatency cannot be improved.

Theorem 3.12. SpreadLatency *is* $\Omega(\delta \log U)$*-competitive.*

Proof. Consider a half-line with end nodes v and s, unit communication costs, and $\delta \geq 6$. An adversary releases messages $j_{i,k}$ at node v for $i = 1, \ldots, \delta/4 - 1$ and $k = 0, \ldots, \log \delta - 1$. The release time of message $j_{i,k}$ is $r(i,k) = \delta + \frac{2^{k+1}}{\delta} i - 2^k$ and the latency is $L(i,k) = 2^k + \delta$. It follows from the observations $r(i,k) < r(i+1,k)$ and $r(\delta/4, k) < r(1, k-1)$ that messages are released ordered by decreasing value of k and then by increasing value of i. This induces the total order $\prec$ on pairs (i,k) and (i',k'). Formally, $(i,k) \prec (i',k')$ if either $k > k'$ or $k = k'$ and $i < i'$.

We have $I_{j_{i,k}} = [r(i,k) + \delta, r(i,k) + L(i,k)]$. As $r(i,k) + \delta \leq 2\delta$ and $r(i,k) + L(i,k) \geq 2\delta$ for each message $j_{i,k}$, we have $\bigcap_{j \in j_{i,k}} I_j \neq \emptyset$, hence the adversary may aggregate all messages at their common release node v. Let $W_j(u)$ be, as defined before, the waiting time interval of message j on u determined by SpreadLatency. We have $W_{j_{i,k}}(v) = [r(i,k), r(i,k) + \frac{L(i,k) - \delta}{\delta}]$.

It follows from simple arithmetics that $r(i,k) + \frac{L(i,k)-\delta}{\delta} < r(i',k')$ for all pairs $(i,k),(i',k')$ such that $(i,k) \prec (i',k')$. This can be seen as follows. If $i < i'$ and $k = k'$ then the claim follows trivially. Suppose $k > k'$. Then we should prove $\delta + \frac{2^{k+1}}{\delta} i - 2^k + \frac{2^k}{\delta} < \delta + \frac{2^{k'+1}}{\delta} i' - 2^{k'}$. This leads to the following:

$$\begin{aligned} 2^k(\frac{2i}{\delta} - 1 + \frac{1}{\delta}) &< 2^{k'}(\frac{2i'}{\delta} - 1) \Rightarrow \\ 2^k(2i - \delta + 1) &< 2^{k'}(2i' - \delta) \Rightarrow \\ \frac{\delta - 2i'}{\delta - 2i - 1} &< \frac{2^k}{2^{k'}}. \end{aligned}$$

The last inequality holds because $\frac{\delta-2i'}{\delta-2i-1} < 2$ and $\frac{2^k}{2^{k'}} \geq 2$.

Hence, $W_j(v) \cap W_{j'}(v) = \emptyset$ for any two messages j and j' and SpreadLatency sends all messages separately from v. As all messages can be aggregated at v, and $\delta = \Theta(U)$, this proves the theorem. □

SpreadLatency is a memoryless algorithm, i.e. it bases decisions only on information which is available at the time of decision making, and not on information of previous decisions, and packets which no longer reside at that node. The following lower bound shows that the competitive ratio of SpreadLatency cannot be beaten by more than a factor $O(\log U)$ by any other memoryless algorithm. In the derivation of the lower bound we restrict to memoryless algorithms that employ the same algorithm in nodes which have the same communication time to s, and the same indegree. This is not a severe restriction, given that this is typically the only information about the network that a node has.

Theorem 3.13. *Any deterministic asynchronous memoryless algorithm is $\Omega(\delta)$-competitive.*

Proof. Consider an intree with root s, where each non-leaf node has indegree δ, and all leaves have communication time δ to s. An adversary releases message 1 with latency L at time r_1 in a leaf v_1. There must be a node u where message 1 waits at most $(L - \tau_{v_1})/\delta$. The adversary releases message $j, j = 2, \ldots, \delta$ at time $r_1 + j(L - \tau_{v_1})/\delta$. The adversary chooses the release nodes of these messages such that all messages j are sent over node u, and no two messages traverse the same node before reaching u. Because $\tau_{v_j} = \delta \ \forall j$ and we assumed that any memoryless algorithm applies the same algorithm in nodes which have the same communication time to the sink, and the same degree, all messages are sent non-aggregated to and from u, whereas they are aggregated as early as possible in an optimal solution, in particular at u. □

The lower bound does not hold for arbitrary algorithms as a node may adjust the waiting time of subsequent messages that traverse that node. However, we note that only if a node delays subsequent messages longer the competitive ratio may improve. The following theorem formalizes this idea.

Theorem 3.14. *Any online algorithm for which the waiting time of message j at its release node is at most $\frac{L-\tau_{v_j}}{K}$ is $\Omega(K)$-competitive.*

Proof. Consider a half-line which consists of two nodes v and s. We assume a constant latency of L for each message. The adversary releases $K-1$ messages with an interval of $(L-\tau_{v_j})/(K-1)$ at v. Since the waiting time of message j at v is at most $(L-\tau_{v_j})/K$, none of these messages are aggregated in the online solution, whereas they are all aggregated in one packet in an optimal solution. □

For arbitrary asynchronous algorithms we do not have any better lower bound than the bound of Theorem 3.9 for the synchronous case. Furthermore, notice that for any memoryless algorithm to have a competitive ratio better than some constant times the number of messages, it should delay messages at their release node.

We conclude this paragraph with a proof that asynchronous distributed algorithms which produce solutions that satisfy property (ii) of Lemma 3.1 can not have a competitive ratio which is better than proportional to the number of messages.

Theorem 3.15. *Any distributed deterministic asynchronous algorithm which only delays messages at their release node is $\Omega(m)$-competitive.*

Proof. Consider the graph of Figure 3.1, and assume a constant message latency $L \geq m+\delta = m+2$. For each j, $j = 1, \ldots, m$, an adversary releases one message j at node u_j. Because the algorithm of each node is deterministic, the adversary knows in advance the waiting time of each message j at node u_j. Moreover, because we assume the algorithm to be asynchronous we may assume without loss of generality that the waiting time of the first message released at a node does not depend on the release time of this message. Hence, the algorithm knows in advance the waiting time of each message j.

The adversary releases message 1; let t_1 be the time message 1 is sent from u_1. Now, the adversary chooses the release times of message j such that it is sent from its release node at time $t_j := t_{j-1}+1$. As no message is delayed after being sent from its respective release node, messages j are sent from node v in packets containing a single message, in a time interval of length m. As each message can be delayed for time at least m the adversary can aggregate all messages at v, and send them in a single packet. □

The theorem demonstrates that enforcing structural properties of classes of optimal offline solutions are no guarantee for obtaining online solutions with a good competitive ratio.

3.4.3 Special instances

So far, in this section we considered online distributed algorithms. Theorem 3.9 leaves open the possibility that there exist online centralized algorithms with a competitive ratio strictly better than $\Omega(\log U)$. It is interesting to investigate whether such centralized algorithms exist in order to determine to what extent the lower

bound of Theorem 3.9 can be attributed to the distributed nature of the model. First, we present a lower bound on the competitive ratio for *any* algorithm.

Theorem 3.16. *No online algorithm can have a competitive ratio less than* 2*, not even on a half-line.*

Proof. Consider a half-line consisting of nodes u_2, u_1, s with unit cost. Fix any algorithm ALG. At some time t an adversary releases a message at u_2. If ALG sends the message at time t then at time $t+1$ another message is released at u_2. This gives ALG a maximum node cost of at least 2 against a node cost of 1 for the adversary. If ALG delays the message then the adversary releases a message at u_1 with due date $t+2$. Again, this gives ALG a maximum node cost of at least 2 against a cost per node of 1 for the adversary. □

However, we will outline in this paragraph that the gap in approximability between centralized and distributed algorithms is not so much related to the information model, but more to the fact that the maximum allowed delay can differ over messages. If we take a closer look at the lower bound proof of Theorem 3.9, then we may observe that it is based on instances where the maximum ratio of the length of two arrival intervals is $O(\log U)$.

We analyze CommonClock for instances where the latency is a constant L, and $L \geq 2\delta$. In this case each message can be delayed for at least time δ. Hence, also U is a constant and in this case CommonClock is in fact constant competitive.

Corollary 3.17. *Under constant latency* $L \geq 2\delta$ CommonClock *is* 6*-competitive.*

Proof. Suppose we assume a constant latency L, $L = k\delta$, $k \geq 2$. Then for each message j we have $(k-1)\delta \leq |I_j| \leq k\delta$. Hence, $U = 2$ and the proof follows from the proof of Theorem 3.8. □

Note that if CommonClock knows the latency L, then using an adjusted time function $t^*(I) := \max\{kL | kL \in I, k \in \mathbb{N}\}$ instead of $t(I)$, CommonClock is even 3-competitive, in case of a constant latency L, and $L \geq 2\delta$. In this case the algorithm CommonClock is in fact equivalent to a *heartbeat* algorithm which sends packets at specific time intervals. Heartbeat algorithms are typical algorithms for the TCP model; see Stevens [76].

We have not been able to devise centralized algorithms for general LDAP, with competitive ratio strictly better than that of CommonClock. However, we present a centralized algorithm which yields a constant competitive ratio in case the graph is a half-line, and messages have a constant maximum allowed latency L, also called the *constant latency*. The centralized algorithm is called EarliestDueDate, and is described on the next page.

Note that a constant message latency assumes $L_j \geq \delta$ for each message j, as otherwise not all messages can be sent to the sink within the maximum allowed latency. As such, this constraint is less strict than the $L_j \geq 2\delta$ necessary to prove the constant competitiveness of the distributed CommonClock-algorithm. Recall, that without this condition CommonClock is $\Omega(\log U)$-competitive.

Algorithm 5 EARLIESTDUEDATE

for each time t **do**
- Select any message with earliest due date, t^*, from the set of messages that have not arrived at the sink, as the *responsible message*;
- Each node v which contains messages sends a packet containing all messages present at this node if $t + \tau_v = t^*$.

end for

Also note that in case of constant latency once a message is chosen as responsible message, we may assume without loss of generality that it remains the responsible message in the network until it reaches the sink. In case of arbitrary latencies, a newly released message may become the responsible message if its due date precedes the due date of the previous responsible message. It can easily be observed that any EARLIESTDUEDATE-solution satisfies properties (i-iii) of Lemma 3.1, i.e. properties of some optimal offline solution.

Theorem 3.18. EARLIESTDUEDATE *is* 2*-competitive for* LDAP *on a half-line, with a constant latency.*

Proof. We will prove that the number of packets sent over a node v in the EARLIESTDUEDATE-solution is at most 2 times the number of packets sent over v in an optimal solution, which satisfies property (i-iii) of Lemma 3.1.

Assume that in this optimal solution packets arrive at s at times $t_1 < \cdots < t_\ell$. Let P_h^* be the packet arriving at time t_h at s. Observe that $t_h \in I_j \ \forall j \in P_h^*$. Suppose to the contrary that there is a node v such that the messages of P_h^*, which pass v, are sent from v in at least 3 packets, P_1, P_2, P_3. Let the arrival times of these packets be respectively t_1, t_2, and t_3 and we assume $t_1 < t_2 < t_3$.

As EARLIESTDUEDATE sends each packet of messages such that it arrives at the due date of some message we must have for responsible message j of P_2 that $j \notin P_h^*$ (note that it is not necessary that $j \in P_2$). This can be seen as follows. All messages in P_h^* are sent simultaneously in the optimal solution to the sink. Hence each message in P_h^* can be sent such that it arrives at the sink at the due date of some message in P_h^*, if it has not been sent before. Now, if $j \in P_h^*$, and P_2 is sent such that it arrives at the due date of j, then all messages in $P_h^* \cap P_3$ would have been aggregated with this packet because $t_2 < t_3$. But this contradicts our assumption that messages of P_h^* are sent from v in at least 3 packets.

Thus, $j \notin P_h^*$. Next, we must have $t_2 \in I_j$, and $t_1 \notin I_j$ as otherwise $j \in P_1$ by definition of EARLIESTDUEDATE. Also, because $P_h^* \cap P_3 \neq \emptyset$ we have $t_2 < t_h$ as otherwise EARLIESTDUEDATE would have sent all messages in $P_h^* \cap P_3$ at time t_2.

Any message $j_1 \in P_h^* \cap P_1$ can be sent both at times t_1 and t_h, hence $|I_{j_1}| \geq t_2 - t_1$, and $|I_j| < |I_{j_1}|$. Thus, because we assume a constant latency we must have $\tau_{v_j} > \tau_{v_{j_1}}$, i.e. message j is sent further from the sink than $j_1 \in P_h^* \cap P_1$.

Now, consider the optimal solution. As message j has due date $t_2 < t_h$ it must arrive at the sink before message j_1; but as the network is a half-line message j

would pass message j_1, and it follows from property (i) in Lemma 3.1 that message j_1 is aggregated with j. But then $j \in P_h^*$ which contradicts our assumption. □

The competitive ratio of EARLIESTDUEDATE can be arbitrarily close to 2. Consider the half-line u_2, u_1, s with costs $c(u_2, u_1) = l, c(u_1, s) = 1$, for some l, $l \geq 1$. Assume a latency of $L \geq 3$ for each message. Message 1 is released at node u_1 at time $r_1 = 0$. Message 2 is released at node u_2 at time $r_2 = L - 2$. Message 3 is released at node u_2 at time $r_2 = L - 1$. EARLIESTDUEDATE aggregates messages 1 and 2 in a packet and sends message 3 separately, yielding a maximum cost of $2l$. The optimal solution aggregates messages 2 and 3 and has a maximum cost of l. Hence, EARLIESTDUEDATE is at best 2-competitive. As this example shows, EARLIESTDUEDATE cannot be better than 2-competitive because it aggregates all messages that can be aggregated with the responsible message in a single packet. It does not consider the extra communication costs nodes have to make in order to forward packets to the node where they can be aggregated with the responsible message.

Using this insight we give an example which demonstrates that EARLIESTDUEDATE is at least $\Omega(m)$-competitive on a tree. Consider the tree of Figure 3.1; assume that all leaf nodes $u_1, \ldots, u_{m-1}$ are connected to v with a path of length $L - 1$ and let u_m be at distance 1 from v. The adversary releases $2(m-1)$ messages with some constant latency $L \geq 2m$. He releases message j_i at node u_m at time $r_i = L + i$ for $i = 1, \ldots, m - 1$. Further, he releases messages j_i' at node u_i at time $r_i = i$ for $i = 1, \ldots, m - 1$. Thus $I_{j_i} = [L + i + 1, 2L + i]$ and $I_{j_i'} = [L + i + 1, L + i + 1]$. EARLIESTDUEDATE aggregates each message j_i with message j_i' into a single packet, regardless of the arc costs. As a result, no two packets can be aggregated at v. This gives a cost of $mc(u_m, v)$ for leaf u_m, a cost of $c(u_i, v)$ for leaf u_i, $i = 1, \ldots, m - 1$, and a cost of $mc(v, s)$ for node v. If we choose as cost function $c(u_m, v) \geq (m + 1)$ and $c(a) = 1$ for all $a \in A \backslash (u_m, v)$, then, for both objectives, the optimal solution aggregates all messages at u_m into a single packet, thus (u_m, v) has to be traversed only once. This gives a cost of $c(u_m, v)$ for leaf u_m, a cost of $c(u_i, v)$ for leaf u_i, $i = 1, \ldots, m - 1$, and a cost of $(m + 1)c(v, s)$ for node v. This proves that EARLIESTDUEDATE is $\Omega(m)$-competitive on a tree. As a result the exposition of algorithm EARLIESTDUEDATE is only useful from a theoretical point of view to gain a better understanding of problem LDAP.

We conclude this section with a general lower bound on the competitive ratio of any online algorithm on instances with constant latency. The theorem is a generalization of the proof that EARLIESTDUEDATE is at best 2-competitive, as described above.

Theorem 3.19. *No online algorithm can be better than $\sqrt{2}$-competitive for* LDAP *with a constant latency L, $L \geq 2\delta$, not even on a half-line.*

Proof. Consider the half-line u_2, u_1, s and costs $c(u_2, u_1) = l, c(u_1, s) = 1$, for some l,$1 \leq l \leq 2$. Assume a latency of $L \geq 4$ for each message. Message 1 is released at node u_1 at time $r_1 = 0$. Message 2 is released at node u_2 at time $r_2 = L - 2$. If node u_2 does not send message 2 immediately to be aggregated with message 1, then

the algorithm incurs a maximum cost of $\max\{2, l\} = 2$ whereas the optimal cost is $\max\{1, l\} = l$. If messages 1 and 2 are aggregated, then message 3 is released at node u_2 at time L. Hence, a maximum cost of $2\max\{l, 1\} = 2l$ is incurred, whereas the maximum cost of a node in the optimal solution is $\max\{2, l\} = 2$. The theorem follows if we choose $l := \sqrt{2}$. □

3.5 Variations and generalizations of the model

In this section we discuss a variant and generalizations to the LDAP model. We consider a variant of LDAP where the objective is to minimize the total energy costs, and we discuss two generalizations of the aggregation model. The first generalization is to use an energy function which is non-decreasing in the number of messages in a packet, instead of constant. The second generalization limits the possibility of total aggregation of two messages.

3.5.1 Minimize total energy use

We consider LDAP with the objective to minimize total energy consumption. The problem can be formulated as an integer program similar to formulation of the original LDAP (3.1) but with objective to minimize $\sum_{a \in A} c(a) \sum_{i=1}^{m} x_{ia}$, instead of z. The objective to minimize total energy use is common in networks where nodes have access to a replenishable energy source.

For the offline problem, the rounding algorithm LPRounding also provides a 2-approximate solution, as can be observed from the proof of Theorem 3.5. Besides, for this objective there exists a dynamic programming formulation, which yields an optimal solution in polynomial time on a half-line. We give the dynamic programming formulation.

Assume message set M is ordered by increasing due date, i.e. $d_1 \leq \ldots \leq d_m$. For $0 \leq i \leq k \leq m+1$, we denote by $M(i,k)$ the set of messages in $\{i+1, \ldots, k-1\}$ whose earliest arrival time is strictly later than i's due date. More formally,

$$M(i,k) \ := \{j \in M \mid i < j_{\min} \leq j < k\} \ .$$

$\mathcal{O}PT(i,k)$ denotes the cost of an optimal solution to the partial instance defined by the subset of messages in $M(i,k)$. The dynamic programming formulation is based on the following lemma.

Lemma 3.20. *Let ℓ be a message in $M(i,k)$ whose release node v_ℓ has maximum communication time τ_{v_ℓ} among all messages in $M(i,k)$. Then,*

$$\mathcal{O}PT(i,k) \ = \ c_{v_\ell} + \min_{\ell_{\min} \leq j \leq \ell} \big(\mathcal{O}PT(i,j) + \mathcal{O}PT(j,k)\big) \ . \tag{3.6}$$

Proof. In an optimal solution to $M(i,k)$, message ℓ arrives at the sink at time d_j, for some $j \in \{\ell_{\min}, \ldots, \ell\}$. Moreover, there exists an optimal solution to $M(i,k)$ in which all messages in $M_{i,k}(j) := \{h \in M(i,k) \mid h_{\min} \leq j \leq h\}$ arrive at the sink

together with message ℓ at time d_j because they can join ℓ on its way to the sink at no additional cost. Notice that $M(i,k)$ is the union of the disjoint sets $M(i,j)$, $M_{i,k}(j)$, and $M(j,k)$. Moreover, since no two messages in $M(i,j)$ and $M(j,k)$ can ever reach the sink together in a feasible solution, the remaining problem can be decomposed into two subproblems for messages in $M(i,j)$ and $M(j,k)$. This yields the desired result. □

Theorem 3.21. *Off line* LDAP *minimizing total energy use can be solved in* $O(m^3)$ *time, on a half-line.*

Proof. Consider a dynamic program which computes values $\mathcal{O}PT(i,k)$, $0 \leq i \leq k \leq m+1$, in order of non-decreasing $k-i$. It follows from Lemma 3.20, that the algorithm can compute these $O(m^2)$ values and since it takes $O(m)$ time to evaluate (3.6) the algorithm needs a total time of $O(m^3)$. The value of an optimal solution can be derived from $\mathcal{O}PT(0,m+1)$. Moreover, keeping track of the messages j that minimize the right hand sides of (3.6), we may derive for each message in M the time at which it is sent to s. □

Next, we consider the objective minimizing total energy use for distributed information models. Because the proofs on the competitive ratio for the distributed algorithms CommonClock and SpreadLatency (Theorems 3.8 and 3.11 and Corollary 3.17) are based on bounding the energy use of each node these proofs also hold for the objective to minimize the total energy use. However, the proofs of lower bounds on the competitive ratio for the distributed models do not hold.

For the offline version of the problem, a minor adaptation makes the **NP**-hardness proof hold also for LDAP with objective to minimize total energy use, see [9].

3.5.2 Generalizations of aggregation models

We consider generalizations of the assumption of total aggregation. Under total aggregation any two messages can be aggregated into a single packet regardless of their release times and release nodes. In practice total aggregation is not always possible; whether messages can be aggregated depends on the data. Generally speaking, aggregation is more likely to be possible if messages are released within the same region around the same time [36].

In this paragraph we consider two generalizations which model these limitations. We analyze the consequences of partial aggregation by modeling the cost function as a concave nondecreasing function, and we analyze the total aggregation model of messages in a geographically bounded region.

Concave cost functions

In the *concave cost function model* the communication cost of a packet is a non-decreasing concave function of data size. The reason to choose a concave cost function is twofold. First, short-range communication costs are typically determined by a (significant) start-up cost, and a communication cost which is linear in the data

size [43, 68]. This can be modeled through a cost function which is affinely linear for a positive number of packets and zero in case of no packets, i.e. a concave and non-decreasing function. Second, aggregation typically results in a packet with data size less than the sum of the original packets. This can also be modeled through a concave non-decreasing function [40]. Thus a concave cost function reflects both the economies of scale of sending an aggregate packet in case of start-up cost, and the gain obtained by data compression.

Most results of the previous sections generalize to concave non-decreasing cost functions. But first we consider a negative result. We observe that an optimal solution does not always satisfy Lemma 3.1 (i), i.e. there are instances where a better solution can be obtained if a node does not aggregate all messages into a packet, but is allowed to delay one or more packets, and send these at a later time. We illustrate this by an example. Consider a node v adjacent to s and messages $j, j = 1, \ldots, 4$, with release times $r_1 = r_2 = 0, r_3 = r_4 = 1$. Message 1 has latency 1, the other messages have latency 2. Let the communication cost on arc (v, s) be $c(x) = \sqrt{x}$. Message 1 and 2 are present together at node v at time 0. A solution which consists of packets $\{1, 2\}$ and $\{3, 4\}$ has a cost of $2\sqrt{2}$. However, the solution which consists of packets $\{1\}$ and $\{2, 3, 4\}$ has a cost of $\sqrt{3} + 1 < 2\sqrt{2}$.

Next, we consider the positive results. Since the constant cost function of the total aggregation model is both concave and non-decreasing, all lower bounds derived in this chapter hold for any such cost function. Thus, we focus on the upper bound results. Consider the algorithms we presented for LDAP with constant energy use function, CommonClock for the synchronous model and SpreadLatency for the asynchronous model. The proofs of Theorem 3.8 and Theorem 3.11 are both based on bounding the number of packets the algorithm sends for each packet that the optimal solution sends. It follows from a straightforward analysis that these proofs generalize to cost functions which are both concave and non-decreasing.

Geographically bounded aggregation

In practice, the aggregation of packets can be subject to geographical constraints. In particular, it may be infeasible to aggregate two messages originating from nodes that are too far apart. In order to model this kind of constraints we introduce the *geographically bounded total aggregation model*, which is defined as follows. Messages j and j' can be aggregated into a single packet if both j and j' can reach a common ancestor node in time at most ρ; i.e., there is a node v on the intersection of the path from v_j to s and the path from $v_{j'}$ to s, such that $\tau_{v_j} - \tau_v \leq \rho$ and $\tau_{v_{j'}} - \tau_v \leq \rho$. Otherwise, the messages can not be aggregated. Note that in this case multiple packets may be sent from a node at a single time, in case the messages in these packets can not be aggregated into a single packet. If two messages can be aggregated they can be totally aggregated, i.e., the cost of a packet is independent of the number of messages it contains. Note that it is possible that messages i and j can be aggregated, and messages j and k can be aggregated, but messages i and k cannot be aggregated. The total aggregation model studied before is a special case, with $\rho = \delta$.

For the synchronous model we propose the CommonClock algorithm again.

We briefly discuss the proof of the competitive ratio of CommonClock and we discuss how the lower bound proof can be adapted.

CommonClock is $O(\log U)$-competitive for any choice of ρ, because Lemma 3.7 remains valid. Also, Theorem 3.9, which gives a lower bound of $\Omega(\log U)$ on any deterministic synchronous algorithm, remains valid.

For the asynchronous model we present a generalization of the algorithm SpreadLatency; the algorithms are identical if $\rho = \delta$.

Algorithm 6 GeographicSpreadLatency

Let the waiting time of message j be $w_j = (L_j - \tau_{v_j})/\rho$ per node visited.
Each node sends a packet as soon as the waiting time of some message, called the responsible message, at that node has elapsed. This packet contains all messages present at that node at that time, which can be aggregated with the responsible message.

Next, we prove upper and lower bounds similar to the proofs given in Theorems 3.11 and 3.13.

Theorem 3.22. GeographicSpreadLatency *is* $O(\rho \log U)$*-competitive.*

Proof. We prove that for every node $v \in V$ the number of packets GeographicSpreadLatency sends from v is at most $O(\rho \log U)$ times that number in an optimal solution.

Let $\mu := \max\{1, \min_j(L_j - \tau_{v_j})\}$. Consider a packet P of messages sent by an optimal solution from node v at time t. Without loss of generality we do not consider messages for which $\min_j(L_j - \tau_{v_j}) = 0$ as these messages have to be sent upon release by both GeographicSpreadLatency and the optimal algorithm. To bound the number of packets sent by GeographicSpreadLatency that contain at least one message from P, define $P_i := \{j \in P | 2^{i-1}\mu \leq L_j - \tau_{v_j} < 2^i\mu\}$, for $i = 1, \ldots, \lceil \log U \rceil$. We *charge* any packet sent from node v to a message that caused the packet to be sent due to its waiting time being over; ties are broken arbitrarily, but if possible to some message which was charged a packet arriving at v. It suffices to prove that the number of packets charged to messages in P_i is $O(\rho)$.

Suppose, v is the first node where all messages in P can be aggregated. Then we must have $\tau_{v_j} - \tau_v \leq \rho$ for all $j \in P$, hence each message $j \in P_i$ has a waiting time of at least $2^{i-1}\mu/\rho$ at node v. Thus, the delay between any two packets that are charged to messages in P_i is at least $2^{i-1}\mu/\rho$. Since the optimal solution sends packet P at time t from node v, we get $t \in I_j(v)\ \forall j \in P$ and thus $I_j(v) \subseteq [t - 2^i\mu, t + 2^i\mu]\ \forall j \in P_i$. Thus, the number of packets charged to messages in P_i is at most $2 \cdot 2^i\mu/(2^{i-1}\mu/\rho) = 4\rho$. This proves the claim in this case.

Otherwise, let v' be the first node where all messages in P can be aggregated. It follows from the argument above, that at most 4ρ packets which contain a message of P are charged to a message in P_i at node v'. GeographicSpreadLatency only de-aggregates these packets if some message in this packet can be sent with another responsible packet. Hence, the number of packets which contain a message

of P and are charged to a message in P_i is at most 4ρ for any node on the path $v' - s$, in particular for node v. □

Theorem 3.23. *Any deterministic asynchronous memoryless algorithm is* $\Omega(\rho)$*-competitive.*

Proof. Consider an intree with root s, where each non-leaf node has indegree ρ, and all leaves have communication time ρ to s. An adversary releases message 1 with latency L at time r_1 in a leaf v_1. There must be a node u where message 1 waits at most $(L - \tau_{v_1})/\rho$. The adversary releases message $j, j = 2, \ldots, \rho$ at time $r_1 + j(L - \tau_{v_1})/\rho$. The adversary chooses the release nodes of these messages such that all messages j are sent over node u, and no two messages traverse the same node before reaching u. Because $\tau_{v_j} = \rho \; \forall j$ and we assumed that any memoryless algorithm applies the same algorithm in nodes which have the same communication time to the sink, and the same degree, all messages are sent non-aggregated to and from u, whereas they are aggregated as early as possible in an optimal solution, in particular at u. □

3.6 Conclusion and open problems

In this chapter we studied the latency constrained data aggregation problem, LDAP. The problem is to find a schedule to gather a set of packets at a central sink node. Aggregation can be used to decrease energy use. The objective of the problem is to find schedules which balance the energy costs of the nodes and the latency costs of the messages.

Our main focus was to analyze the complexity of the problem, and to develop an online distributed algorithm with good competitive ratio. We proved that LDAP is **NP**-hard, and we presented an offline algorithm which yields a 2-approximation. We also proved that no online algorithm can be better than 2-competitive.

Then we considered online distributed algorithms under two time models. For the synchronous time model we considered the algorithm CommonClock, which uses synchronous clocks to have nodes synchronize their communication, in a distributed setting. The algorithm is $O(\log U)$-competitive, where U is the ratio between minimum and maximum of the maximum allowed message delay. We demonstrated that no deterministic algorithm can be better than $\Omega(\log U)$-competitive. Thus, CommonClock is best possible up to a multiplicative constant. For the asynchronous time model we considered the algorithm SpreadLatency, and proved that it is $O(\delta \log U)$-competitive. The algorithm is memoryless, and for this class of algorithms no algorithm can be better than $\Omega(\delta)$-competitive.

We also analyzed some variations of the problem. In case the latency is constant, i.e. messages have more or less the same priority, and the latency is at least twice the maximum network communication time, then CommonClock is in fact a constant competitive algorithm. The algorithms we presented have the same approximation ratio when minimizing the sum of energy costs; and for this problem we presented

an exact polynomial time offline algorithm, in case the graph is a half-line. Our main results are summarized in Table 3.1.

Model	**Graph**	$\max c(v)$		$\sum c(v)$	
		ratio	*lower bound*	*ratio*	*lower bound*
$Offline$	$tree$	2	*open*	2	*open*
	$half-line$	*open*	*open*	1	1
$Online$					
$-centralized$	$tree$	$O(\log U)$	2	$O(\log U)$	2
$-synchronous$	$tree$	$O(\log U)$	$\Omega(\log U)$	$O(\log U)$	2
$-asynchronous$	$tree$	$O(\delta \log U)$	$\Omega(\log U)$	$O(\delta \log U)$	2

Table 3.1: Approximation and competitive ratio results on LDAP.

Finally, we studied two generalizations of the aggregation model, and demonstrated that for these models our distributed algorithms give similar results in terms of competitive ratio. In this sense our algorithms can be considered robust.

Our research yields some open problems related to the data aggregation problem. For the offline problem the complexity on a half-line is an open problem when minimizing maximum energy use; this contrasts with the problem with objective minimizing sum of energy use, for which we presented a polynomial time algorithm. For the online problem, we do not know whether there exists a constant competitive online algorithm for LDAP with arbitrary message latencies. For the distributed asynchronous time model there is a gap of $O(\log \delta)$ between the competitive ratio of the memoryless algorithm we proposed, and the lower bound on the competitiveness of any memoryless asynchronous algorithm. Also, it is an open problem whether there exists an algorithm for the asynchronous time model with competitive ratio which is strictly better than that of any memoryless algorithm.

Another open problem is related to interference. The algorithms we proposed use aggregation to reduce the number of times that a node has to communicate. This aggregation also decreases the probability of interference between communicating nodes. However, the LDAP problem does not explicitly consider interference. It would be interesting to analyze the performance of our algorithms in a model which takes interference into account as well.

Chapter 4

Data aggregation with soft latency constraints

4.1 Introduction

In this chapter we study the bicriteria data aggregation problem (BDAP), the bicriteria formulation of DAP. Recall, that DAP is to gather a set of messages, released in a wireless network, at some sink node. Nodes can delay messages in order to aggregate multiple messages into a single packet, thus reducing communication costs at the expense of an increased message latency. The objective is to minimize the maximum communication costs and the maximum latency costs.

We formulate the data aggregation problem as a bicriteria problem. Bicriteria problems are optimization problems with two objectives. Often, real life problems have two or more objectives. In the literature such problems are typically modeled as optimization problems with a single objective, which is a weighted sum of the objectives.

This approach seems reasonable if we assume that the objectives can be measured in the same unit, i.e. money or time. However, often the objectives have different units. In this case the weights used to obtain a single objective strongly influence the value of a solution, and there is no impartial way to choose these weights. In the data aggregation problem the objectives are communication costs, which we measure in energy usage, and latency costs, which we measure in time. Because the objectives are measured in different units, we choose an explicit bicriteria problem formulation.

In bicriteria optimization an optimal solution is called a Pareto optimal solution, which is a solution where none of the objectives can be decreased without an increase in the value of the other objective. We have proven in the previous chapter, that DAP is **NP**-hard when minimizing the maximum communication costs under constraints on the maximum latency. Hence, BDAP is also **NP**-hard and we focus on deriving approximately good solutions. For this purpose we need a definition of

approximately good solutions for bicriteria problems.

We use a bicriteria formulation of approximation solutions as introduced in [60, 71]. In this formulation we minimize one of the objectives under a budget restriction on the other objective. The difference with LDAP, is that in this case the budget restriction is a soft constraint. Using soft constraints we analyze to what extent a budget excess on the latency constraints affects the communication costs.

We consider a bicriteria problem with objective functions C_A and C_B. We call a bicriteria problem a (B, A)-bicriteria problem if we minimize objective C_A under a budget $\mathcal{B}$ on objective C_B. For a fixed budget $\mathcal{B}$ we define $S_A^*(\mathcal{B})$ as a solution with minimal value with respect to function C_A, amongst all solutions S for which we have $C_B(S) \leq \mathcal{B}$.

We say a solution S is (β, α)-approximate if $C_A(S) \leq \alpha C_A(S_A^*(\mathcal{B}))$ and $C_B(S) \leq \beta\mathcal{B}$, for any value $\mathcal{B}$ for which there exists a solution. This formulation gives a relative guarantee, i.e. the value of the approximate solution is bounded with respect to the best solution to one of the objectives, under a budget constraint on the other objective. In the literature there also exists a definition of an approximate bicriteria solution, which provides an absolute guarantee, i.e. the cost of the solution is bounded with respect to the best offline solutions to two single objective problems; e.g., in [75] the authors minimize makespan and average completion time of a schedule. Such an approach is not likely to yield good results for the data aggregation problem. The reason is that a bound on the maximum latency has a strong influence on the communication costs; the latter are likely to decrease for each additive increase in the bound on the maximum latency. In fact, it is intuitively clear that the communication costs tend to 1 in case the maximum latency tends to infinity. We have the following formal definition of a bicriteria approximation algorithm.

Definition 4.1 (Bicriteria approximation algorithm). *A (β, α)-approximation algorithm for a (B, A)-bicriteria problem Π is an efficient algorithm which yields a (β, α)-approximate solution S for each instance $I \in \Pi$.*

In case there are multiple Pareto optimal solutions, this formulation is general in the sense that we may obtain similar results regardless which of the two objectives is minimized, and which is budgeted [60].

As in approximation theory we want to associate an approximation ratio to bicriteria problems. We call a pair (β, α) an approximation ratio pair if it is Pareto optimal, i.e. if there is no (β', α')-approximation algorithm for $\beta' < \beta$ and $\alpha' < \alpha$. There may exist multiple approximation ratio pairs, and we could provide a set of all approximation ratio pairs. An alternative to this approach is to determine the maximum value γ such that there does not exist a (γ, γ)-approximation algorithm [60]. We call a (γ, γ)-approximation algorithm a balanced algorithm. Balanced algorithms can be viewed as algorithms which consider both criteria equally important, hence a (γ, γ)-approximation ratio provides a lower bound on the approximation ratio of balanced approximation algorithms.

More information on bicriteria optimization can be found in Ehrgott and Gandibleaux [29]; this paper provides a survey on multicriteria optimization, with an emphasis on research focused on identifying the set of all Pareto optimal solutions.

Bicriteria approximation models for network problems are described in Marathe et al. [60] and Ravi et al. [71].

The algorithms we present are modifications of the algorithms we presented in the previous chapter for LDAP. There we observed that in case message latencies are constant, and not too small, the competitiveness of the algorithm can improve significantly with respect to the general case. In this chapter we observe the same improvement. Because the proofs of different variants of the algorithm are identical, up to the necessary changes to accommodate for the latencies, we have chosen in this chapter to give extensive proofs for BDAP with constant latencies, and more briefly discuss BDAP with arbitrary latencies.

We use the same definitions and notation as in the previous chapter. We introduce some extra notation which is specific to BDAP. In case of a constant latency for each message j, let L denote this constant latency; because of feasibility we have $L \geq \delta$. In a bicriteria setting a solution may exceed the allowed latency L. We call a solution L-bounded if the realized latency l_j of each message j does not exceed L.

We study the (B, A)-data aggregation problem in which objective A is to minimize the maximum communication costs and objective B is to minimize the maximum latency costs. Given a budget L on the latency and value β, $\beta \geq 1$, we study the communication cost of algorithms that provide a βL-bounded feasible solution: a βL-bounded feasible solution is (β, α)-approximate if its communication cost is at most α times the communication cost of the optimal L-bounded solution. In case α or β depend on a parameter the notation is short for $(O(\beta), O(\alpha))$-approximate.

In Section 4.2 we study online algorithms for BDAP under several time models. For the synchronous model we present an algorithm which balances communication and latency costs. The algorithm is $(2, 2)$-competitive. For the asynchronous model we present an algorithm which balances communication and latency costs, up to a multiplicative factor $\log U$. The algorithm is $(2\delta^{\lambda}, 2\delta^{1-\lambda} \log U)$-competitive, for any λ, $0 < \lambda \leq 1$. The algorithm is member of a class of memoryless algorithms for which we show that no better competitiveness than $(\delta^{\lambda}, \delta^{1-\lambda})$ exists.

We introduce the almost synchronous model for this problem. For this model we present an algorithm which on instances with a clock drift of at most Δ between any two nodes and latency budget L is $(1 + 3\Delta\delta/L, \log^2 \delta)$-competitive. For constant Δ, i.e. a drift which does not depend on the network diameter δ, the algorithm has better competitive ratio than the algorithm for the asynchronous model, demonstrating the fact that we may use approximately synchronized clocks to obtain better performance. In Section 4.4 we present conclusions and open problems for BDAP.

In Chapter 3 we discussed related work of DAP. To the best of our knowledge all existing models for DAP and related problems consider optimization of a single criterion, often combining objectives related to communication and latency costs.

4.2 Online algorithms

In this section we analyze BDAP in an online distributed model. We consider the synchronous time model, the asynchronous time model, and the almost synchronous

time model as defined in Subsection 2.2.2.

Similar to the LDAP model we assume that each node v knows its total communication time τ_v to the sink. The assumption is not necessary for the algorithm to function as messages can be delayed longer than the allowed latency. However, it is hard to imagine how to balance communication and latency costs without this information.

4.2.1 A synchronous distributed algorithm

We present a variant of CommonClock which balances communication and latency costs. For any time interval I,$|I| \geq L$, we define $t^*(I) := \max\{kL | kL \in I, k \in \mathbb{N}\}$. Let $I_j^* := [r_j + \tau_{v_j}, r_j + \tau_{v_j} + L]$. Recall, that the arrival time for message j to stay within the allowed budget on the latency is $I_j = [r_j + \tau_{v_j}, r_j + L]$.

Algorithm 7 BalancedCommonClock

Message j is sent from v_j at time $t^*(I_j^*) - \tau_{v_j}$ to arrive at the sink at time $t^*(I_j^*)$ unless some other packet passes v_j in the interval $[r_j, t^*(I_j^*) - \tau_{v_j}]$, in which case j is aggregated and the packet is forwarded directly.

The algorithm is basically equivalent to a *heartbeat* algorithm which sends data packets at regular time intervals [28]. The difference is that in a heartbeat algorithm messages are sent "at the first beat", i.e. at time $\min\{kL | kL \in I, k \in \mathbb{N}\}$.

Theorem 4.2. BalancedCommonClock *is* $(2, 2)$*-competitive.*

Proof. Each message j arrives at the sink not later than $r_j + \tau_{v_j} + L \leq r_j + 2L$, hence its latency cost is at most $2L$, i.e. at most 2 times the budget. It suffices to prove that the communication cost of each node in the BalancedCommonClock-solution is at most 2 times the communication cost of this node in an optimal L-bounded solution.

Assume that in an optimal L-bounded solution packets arrive at s at times $t_1 < \cdots < t_\ell$. Let P_h^* be the packet arriving at time t_h at s. Since $t_h \in I_j \ \forall j \in P_h^*$, $I_j \subset I_j^*$, and $|I_j^*| = L \ \forall j$, we have $I_j^* \subset [t_h - L, t_h + L] =: I \ \forall j \in P_h^*$. If $t_h = kL$ for some $k \in \mathbb{N}$ then in the BalancedCommonClock-solution all messages in P_h^* arrive at s at time t_h or $t_h + L$. If $t_h \neq kL$ then I contains two different multiples of L, say kL and $(k+1)L$, such that $kL < t_h < (k+1)L$. In this case, since $|I_j^*| = L$ $\forall j$, we have $I_j^* \cap \{kL, (k+1)L\} \neq \emptyset \ \forall j \in P_h^*$. By definition of the algorithm in a BalancedCommonClock-solution every message $j \in P_h^*$ arrives at s at one of the times in $\{kL, (k+1)L\}$. Hence, $\forall h = 1, \ldots, \ell$, all messages in P_h^* arrive at s at at most 2 distinct time instants in the BalancedCommonClock-solution. BalancedCommonClock does not delay messages at intermediate nodes. This implies that the nodes used by messages in P_h^* are traversed by these messages at most 2 times in the BalancedCommonClock-solution. □

BalancedCommonClock balances communication and latency costs in the sense that if we allow for each message latency costs of at most twice the budgeted

costs, we obtain a solution with communication costs at most twice the costs of an optimal offline L-bounded solution.

Next, we demonstrate that the algorithm is best possible in the sense that there is no online distributed algorithm with latency costs at most twice the budget which has a better competitive ratio on the communication costs. We start with a simple preliminary lemma.

Lemma 4.3. *Any $(2,\alpha)$-competitive distributed algorithm, $\alpha < 2$, must delay the first message j at leaf node v_j for at least $L-\tau_{v_j}$ as long as no other message arrives at this node.*

Proof. Any algorithm that sends the first message j released at leaf node v_j at time $t < r_j + L - \tau_{v_j}$ will be confronted with another message released at time $r_j + L - \tau_{v_j}$ which the adversary can combine with j. □

Theorem 4.4. *There is no $(2,\alpha)$-competitive distributed algorithm, for $\alpha < 2$.*

Proof. Consider a tree with three leaf nodes u_1, u_2, and u_3 all at distance δ from s. They are chosen such that u', the first node in which the $u_1 - s$-path and the $u_2 - s$-path meet is nearer to s than u'', the first node in which the $u_1 - s$-path and the $u_3 - s$-path meet. Choose $L > 2\delta$. See Figure 4.1 for a visualization.

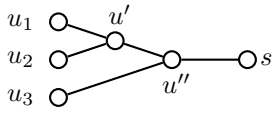

Figure 4.1: The instance of Theorem 4.4.

The adversary releases message 1 at node u_1 at time r_1. And, the adversary either releases message 2 at node u_2 at time $r_2 := r_1 + L - \delta$, or it has released message 3 at node u_3 at time $r_3 := r_1 - L + \delta$. The choice depends on the description of the algorithm, as we describe below.

Consider any distributed algorithm. Suppose it is $(2,\alpha)$-competitive for $\alpha < 2$. Then it follows from Lemma 4.3 that the transit intervals of messages 1,2 and 3 at any non-leaf node u that they pass are respectively:

$$\begin{aligned} I_1^*(u) &= [r_1 + L - \tau_u, r_1 + 2L - \tau_u], \\ I_2^*(u) &= [r_1 + 2L - \delta - \tau_u, r_1 + 3L - \delta - \tau_u], \\ I_3^*(u) &= [r_1 + \delta - \tau_u, r_1 + L + \delta - \tau_u]. \end{aligned}$$

where we use notation $I_j^*(u)$ as before to refer to the transit interval of a $2L$-bounded solution.

Notice that $I_1^*(u') \cap I_2^*(u') \neq \emptyset$ and $I_1^*(u'') \cap I_3^*(u'') \neq \emptyset$ but for any u $I_2^*(u) \cap I_3^*(u) = \emptyset$, given $L > 2\delta$.

If the algorithm allows for the possibility of combining messages 1 and 2 in u' then the adversary gives message 3 and not message 2 and message 3 will have passed u'' before message 1 arrives there. Otherwise, if the algorithm allows for the possibility of combining messages 1 and 3 in u'' then the adversary gives message 2 and not message 3 and message 1 will have passed u' before message 2 arrives there. Thus, in both cases the algorithm incurs communication cost 2.

In both cases the adversary can combine the two messages since the transit intervals of the messages for the adversary in a node u on their respective paths to the sink are

$$\begin{aligned} I_1(u) &= [r_1 + \delta - \tau_u, r_1 + L - \tau_u], \\ I_2(u) &= [r_1 + L - \tau_u, r_1 + 2L - \delta - \tau_u], \\ I_3(u) &= [r_1 - L + 2\delta - \tau_u, r_1 + \delta - \tau_u]. \end{aligned}$$

whence $I_1(u') \cap I_2(u') \neq \emptyset$ and $I_1(u'') \cap I_3(u'') \neq \emptyset$. □

Note that the theorem does not exclude the existence of a $(\beta, 2)$-competitive algorithm for $\beta < 2$. Observe that CommonClock as proposed for LDAP is a $(1, \log \delta)$-competitive algorithm for BDAP.

4.2.2 An asynchronous distributed algorithm

For the asynchronous model we present a modification of the algorithm SpreadLatency.

Algorithm 8 BiSpreadLatency

Let the waiting time of message j be $w_j = 2(L - \tau_{v_j})/(\tau_{v_j})^{1-\lambda}$ per node visited. Each node sends a packet as soon as the waiting time of some message at that node has elapsed. This packet contains all messages present at that node at that time.

Theorem 4.5. *Algorithm* BiSpreadLatency *is* $(2\delta^\lambda, 2\delta^{1-\lambda} \log U)$*-competitive for* λ*,* $0 \leq \lambda \leq 1$*.*

Proof. Consider algorithm BiSpreadLatency for fixed λ, $0 \leq \lambda \leq 1$. First note that, because no message is delayed due to aggregation, the latency of each message j is at most

$$\tau_{v_j} 2(L - \tau_{v_j})/\tau_{v_j}^{1-\lambda} + \tau_{v_j} \leq 2\delta^\lambda L.$$

We prove that for each node u the number of packets BiSpreadLatency sends from u is at most $2\delta^{1-\lambda} \log U$ times that number in an optimal L-bounded solution. This proves the theorem.

Let $\mu := \max\{1, \min_j(L - \tau_{v_j})\}$. Consider a packet P of messages sent by an optimal L-bounded solution from u at t. To bound the number of packets sent by

BiSpreadLatency that contain at least one message from P, define $P_i := \{j \in P \mid 2^{i-1}\mu \leq L - \tau_{v_j} < 2^i\mu\}$, for $i = 1, \ldots, \lceil \log U \rceil$. We charge any sent packet to the message that caused the packet to be sent due to its waiting time being over. It suffices to prove that the number of packets charged to messages in P_i is at most $2\delta^{1-\lambda}$. Since the waiting time of messages $j \in P_i$ at node u is at least $2 \cdot 2^{i-1}\mu/\delta^{1-\lambda}$, the delay between any two packets that are charged to messages in P_i is at least $2^i\mu/\delta^{1-\lambda}$. Since the optimal solution sends packet P at t from u, we get $t \in I_j(u)\ \forall j \in P$ and hence $I_j(u) \subseteq [t - 2^i\mu, t + 2^i\mu]\ \forall j \in P_i$. Thus, the number of packets charged to messages in P_i is at most $2 \cdot 2^i\mu/(2^i\mu/\delta^{1-\lambda}) = 2\delta^{1-\lambda}$. □

BiSpreadLatency is a memoryless algorithm. The following lower bound shows that the competitive ratio of BiSpreadLatency cannot be beaten by more than a factor $O(\log U)$ by any other memoryless algorithm. To be precise, for every choice on the excess of budget B on the latency, no other memoryless algorithm can yield a solution where the competitive ratio on the communication costs is more than a factor $O(\log U)$ less than that of BiSpreadLatency. Similar to the proof of Theorem 3.9 we restrict to memoryless distributed algorithms that employ the same algorithm in all nodes with the same communication time to s, in the derivation of the lower bound.

Theorem 4.6. *No deterministic asynchronous memoryless algorithm is better than* $(\delta^\lambda, \delta^{1-\lambda})$*-competitive, for fixed* λ*,* $0 \leq \lambda \leq 1$*.*

Proof. Consider any deterministic asynchronous memoryless algorithm with latency costs at most δ^λ times the budget on the latency costs for fixed λ, $0 \leq \lambda \leq 1$. An adversary chooses a tree with root s and all leaves at distance δ from s. The adversary releases message 1 with latency L, $L = 2\delta$, at time r_1 in a leaf v_1. There must be a node u where message 1 waits at most $(\delta^\lambda L - \tau_{v_1})/\delta = \delta^{\lambda-1}L - 1 < 2\delta^\lambda$. The adversary releases message j, at time $r_1 + j(2\delta^\lambda)$, $j = 1, \ldots, \delta^{1-\lambda}/2$. The release nodes of these messages are chosen such that they all pass node u, and no two messages can be aggregated before reaching u. Note that the difference between the release times of any two messages is at most $L - \delta$, hence all messages can be aggregated at u. Because $\tau_{v_j} = \delta\ \forall j$ and we assumed that any memoryless distributed algorithm applies the same algorithm in nodes at equal distance from s, all messages are sent non-aggregated to and from u, whereas they are aggregated as early as possible in an optimal solution, in particular at u. □

If we assume that $L \geq 2\delta$, which in practice is not a severe restriction, essentially the same analysis as in the proof of Theorem 4.5 gives $(2\delta^\lambda, 2\delta^{1-\lambda})$-competitiveness. Thus, in this case BiSpreadLatency is a best possible on-line algorithm up to a constant multiplicative factor.

Theorems 4.5 and 4.6 immediately imply the following corollary, choosing $\lambda = \frac{1}{2}$.

Corollary 4.7. *There exists a deterministic asynchronous algorithm that is* $(\sqrt{\delta}, \sqrt{\delta}\log U)$*-competitive and no deterministic asynchronous memoryless algorithm is better than* $(\sqrt{\delta}, \sqrt{\delta})$*-competitive.*

4.2.3 An almost synchronous distributed algorithm

As indicated in Chapter 2 it is typical in wireless networks that clocks have a small drift. I.e. there is a difference between the time indicated at different node clocks. In this section we consider the almost synchronous time mode, where we assume that the difference in times between any two internal node clocks is bounded by Δ.

The BalancedCommonClock-algorithm we presented for the synchronous time model is not robust against clock drift in the sense that its competitive ratio may be much worse if we assume existence of clock drifts. However, the idea underlying the BalancedCommonClock-algorithm gives rise to algorithms which have good competitive ratio even in the almost synchronous model.

Our algorithm for the almost synchronous time model delays messages at their release nodes as in the CommonClock-algorithm, proposed in the previous chapter. To compensate for the clock drift the algorithm delays messages some extra time, linear in Δ.

For a description of the algorithm we divide nodes into classes; a node v is of *class* p if p is the maximal integer such that $\tau_v = h2^p + 1$ for some integer h, and v is of class 0 if $\tau_v = 1$. Note that $p \in \{0, \ldots, \lceil \log \delta \rceil\}$. The algorithm is the following.

Algorithm 9 AlmostSynchronousClock

Message j incurs 3 kinds of delay:

1. at its release node v_j a delay of $t(I_j) - \tau_{v_j} - r_j$;
2. at each node it traverses a delay of Δ ;
3. at the first node of class $p, p > 0$, it traverses a delay which together with the delays of this kind incurred before sums to $2^{p+1}\Delta$.

The waiting time of message j at node v is the sum of the delays at that node. Message j is sent from node v once its waiting time is over, unless some other message (packet) is sent from v earlier in which case j is aggregated with it.

Note that if $\Delta = 0$ the algorithm is identical to the CommonClock-algorithm. To illustrate delay of the third kind we give an example: if a message traverses nodes of classes 1-4 in order 1,2,3,4 then its delay of the third kind of these nodes is respectively $4\Delta, 4\Delta, 8\Delta, 16\Delta$. If the order is 4,1,2,3 then its delay of the third kind is 32Δ at the node of class 4 and 0 elsewhere.

Now we analyze the competitive ratio of AlmostSynchronousClock. Let $V_k := \{v | 2^{k-1} < \tau_v \leq 2^k\}$, for $k = 1, \ldots, \lceil \log \delta \rceil$. First, we analyze the behavior of the algorithm for instances in which the release nodes of all messages are in V_k for some $k \in \mathbb{N}$. We compare the number of packets that an AlmostSynchronousClock-solution sends in the almost synchronous time model to the number of packets CommonClock would send if all clocks would be synchronized. Notice that the competitive ratio is measured against an adversary that sees all the clocks and hence can be assumed to work with synchronized clocks.

Lemma 4.8. *Suppose $v_j \in V_k \; \forall j$ for some k. For each packet that* CommonClock *would send from v if all clocks would be synchronized, in the* AlmostSynchronous

CLOCK-*solution for the almost synchronous time model at most* $(k+1)$ *packets are sent from* v *which contain a message of the* COMMONCLOCK-*packet.*

Proof. Each packet, of COMMONCLOCK and ALMOSTSYNCHRONOUSCLOCK, contains at least one message whose waiting time is completely over when the packet is forwarded. Hence without loss of generality we only consider messages whose waiting time is completely over when counting packets.

Consider a packet P_{CC} sent by the COMMONCLOCK-solution from some node v at time t if all clocks are synchronous. In the remainder of the proof we only consider the messages in this packet. We analyze the number of ALMOSTSYNCHRONOUSCLOCK-packets which contain a message of P_{CC}. By definition of COMMONCLOCK the delays of messages in P_{CC} are chosen such that all messages in this packet which traverse v, i.e. for which v is not the release node, arrive at this node at time t.

The delay of the first kind that a message incurs in the ALMOSTSYNCHRONOUSCLOCK-algorithm is identical to the delay incurred by the COMMONCLOCK-algorithm, if its release node does not have a clock drift. We focus on the deviation from this time to analyze the number of packets ALMOSTSYNCHRONOUSCLOCK sends. This deviation is caused by the clock drift, and the delays of kind 2 and 3.

If $k = 0$ the lemma trivially holds, because all messages which are sent over some node $v \in V_0$ have this node v as release node. Hence, if they are sent in a single packet by the COMMONCLOCK-solution they are also sent in a single packet in the ALMOSTSYNCHRONOUSCLOCK-solution.

For $k \geq 1$ we introduce the following notation: $V_{p,k} = \{v \in V_k | v \text{ is of class } p, \forall v' \in V_k \text{ of class } p, \tau_v \leq \tau_{v'}\}$ for $p \in \{0, \ldots, k-1\}$: $V_{p,k}$ is the set of nodes in V_k of class p with minimal communication time to the sink. Define $\tau(V_{p,k}) := \tau_v$ for some $v \in V_{p,k}$. The nodes of V_k are partitioned into layers $U_{p,k}$ for $p \in \{0, \ldots, k-1\}$ as follows:

$$\begin{aligned} U_{p,k} &:= \{v \in V_k | \tau(V_{p,k}) \leq \tau_v < \tau(V_{p+1,k})\} \text{ for } p \in \{0, \ldots, k-3\}, \\ U_{k-2,k} &:= \{v \in V_k | \tau(V_{k-2,k}) \leq \tau_v\}, \\ U_{k-1,k} &:= V_{k-1,k}. \end{aligned}$$

Note that $V_{p,k} \subseteq U_{p,k}$ for all p. Further, each message j with $v_j \in U_{p,k}$ traverses some node in $V_{p,k}$. See Figure 4.2 on the next page for a sketch of the layer structure.

We characterize a set of nodes S by its depth, which is $\max_{v \in S} \tau_v - \min_{v \in S} \tau_v$ and its class string. The *class string* is an ordered string representing the class of nodes in S by increasing communication time to the sink. I.e. V_3 has depth 4 and class string $\{2010\}$. In general, set V_k has depth 2^{k-1}. Node sets S and S' are equivalent if they have the same depth and class string.

We observe that all messages j with $v_j \in V_k$ are sent to a node in V_{k-1} from some node in $V_{k-1,k}$, i.e. a node of class $k-1$. Also, there are no nodes of higher class in V_k and this is the only node of class $k-1$ a node traverses in V_k. From these observations we may derive that all messages j with $v_j \in V_k$ which are sent over

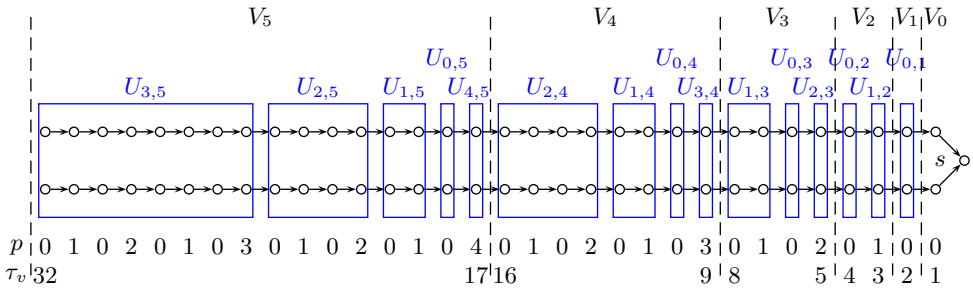

Figure 4.2: Node set V_k and layers $U_{0,k}, \ldots, U_{k-1,k}$ for $k = 1, \ldots, 5$.

the same node $v \in V_{k-1,k}$ are sent from this node in a single packet. This can be seen as follows. The total accumulated delay of kind 2 and 3 that any message has incurred when sent from v is at least $2^k\Delta + \Delta$ because v is of class $k-1$. The total accumulated delay of kind 2 and 3 that any message has incurred when it arrives at v is at most $2^{k-1}\Delta + 2^{k-1}\Delta$, because the maximum class of any other node in V_k is $k-2$ and each message has traversed at most 2^{k-1} nodes. As the clock drift between two nodes is bounded by Δ and the difference between the minimum and maximum delay of any two messages is at most $(2^k\Delta + \Delta) - (2^{k-1}\Delta + 2^{k-1}\Delta) = \Delta$ all messages j with $v_j \in V_k$ which are sent over v in P_{CC} must be sent from this node in a single AlmostSynchronousClock-packet.

Now we are in position to prove our lemma using induction on k. Suppose the lemma holds for $V_0, \ldots, V_k$. Consider set V_{k+1}; this set is partitioned into layers $U_{0,k+1}, \ldots, U_{k,k+1}$. For $\ell = 0, \ldots, k-1$ layer $U_{\ell,k+1}$ is *equivalent* to set $V_{\ell+1}$, hence all messages j with $v_j \in U_{\ell,k+1}$ which are sent from the same node in $U_{\ell,k+1}$ are sent in a single packet. Thus there are at most k packets which arrive at any node $v \in U_{k,k+1}$. As $U_{k,k+1}$ has depth 1, all messages which have v as their release node, are sent from this node in a single packet. Hence, the total number of packets sent from any node in V_{k+1} is bounded by $k+1$. □

Theorem 4.9. AlmostSynchronousClock *is* $(1 + 3\Delta\delta/L, \log^2 \delta)$*-competitive.*

Proof. Consider a packet P sent by the optimal solution. Let $\mathcal{P}_{\mathrm{ASC}}$ be the set of packets sent by the AlmostSynchronousClock-algorithm which contain at least one message from P. Let $N_{i,k} = \{j \in N_i | \tau_{v_j} \in V_k\}$, for $i, k \in \mathbb{N}$, $1 \le i \le \lceil \log U \rceil, 1 \le k \le \lceil \log \delta \rceil$. Observe that for any choice of budget on the latency L, there are at most $2 \log \delta$ nonempty sets $N_{i,k}$. Using this, it follows from Lemma 3.7 and Lemma 4.8 that $|\mathcal{P}_{\mathrm{ASC}}| = O(\log^2 \delta)$. Hence, the communication costs of the AlmostSynchronousClock-solution are at most $O(\log^2 \delta)$ times the cost of an optimal L-bounded solution.

The latency of any message j is at most $\tau_{v_j} + (L - \tau_{v_j}) + \Delta\tau_{v_j} + 2\Delta\tau_{v_j}$, where the sum consists of the communication time, and the delays of kind 1,2,3. Thus, the latency of message j is at most $(1 + 3\Delta\delta/L)$ times the budget on the latency. □

For constant Δ, i.e. a drift which does not depend on the network diameter δ, the algorithm has better (β, α)-competitive ratio than the BiSpreadLatency-algorithm demonstrating the fact that we may use approximately synchronized clocks to obtain better performance. If the drift is of the same order as the network diameter, it is not plausible anymore to consider the clocks to be synchronized in any sense.

4.3 Arbitrary latencies

In the previous section we considered distributed algorithms for BDAP. We assumed a constant latency L for each message j. In this paragraph we analyze the synchronous time model under arbitrary latency L_j for message j. We present a variant of the CommonClock algorithm and give an upper bound on its competitive ratio. The difference in competitive ratio between the constant latency model and the arbitrary latency model is in order of $\log U$, similar to the difference between these two models in LDAP.

We present a version of the CommonClock algorithm which balances the communication and delay costs. We define CommonClock$(I(j))$ as a CommonClock algorithm with arrival interval $I(j)$ for message j. The original algorithm is CommonClock(I_j) where I_j is the arrival interval such that each message arrives within the allowed latency L_j.

We introduce the following notation. Let $\mu := \max\{1, \min_j(L_j - \tau_{v_j})\}$, and let $M_i^* = \{j \in M | (\frac{\log U}{\log\log U})^{i-1}\mu \leq |I_j| < (\frac{\log U}{\log\log U})^i\mu\}$ for $i \in \mathbb{N}$. We choose as arrival interval $I_j^* := [r_j + \tau_{v_j}, r_j + \tau_{v_j} + (\frac{\log U}{\log\log U})^i\mu]$ for message $j \in M_i^*$. We analyze the competitiveness of CommonClock(I_j^*).

To give a competitive analysis of CommonClock(I_j^*) we first derive a bound on the competitive ratio of this algorithm for instances in which the arrival intervals I_j^* have the same length. Let $\mathcal{I}^* = \{1, \ldots, \frac{\log U}{\log\log U - \log\log\log U}\}$.

Lemma 4.10. CommonClock(I_j^*) *is* $(\frac{\log U}{\log\log U}, 3)$*-competitive if* $M_i^* = M$ *for some* $i \in \mathcal{I}^*$.

Proof. First observe that $\forall j \in M_i^*$ we have $|I_j^*| \leq (\frac{\log U}{\log\log U})|I_j|$, hence the latency costs of each message are at most $(\frac{\log U}{\log\log U})$ times the allowed budget L_j. Also $I_j \subset I_j^*$.

We will prove that the communication cost of each node in the CommonClock(I_j^*)-solution is at most 3 times the communication cost of this node in the optimal solution.

Assume that in an optimal L_j-bounded solution packets arrive at s at times $t_1 < \cdots < t_\ell$. Let P_h^* be a packet arriving at time t_h at s. Since $t_h \in I_j\ \forall j \in P_h^*$, $I_j \subset I_j^*$, and $|I_j^*| = (\frac{\log U}{\log\log U})^i\mu\ \forall j$, we have $I_j^* \subset [t_h - (\frac{\log U}{\log\log U})^i\mu, t_h + (\frac{\log U}{\log\log U})^i\mu] =: I$ $\forall j \in P_h^*$. Let $q = \max\{q \in \mathbb{N} | \exists k \in \mathbb{N} : k2^q \in I \text{ and } (k+1)2^q \in I\}$. Note that by definition of q and I we have $(\frac{\log U}{\log\log U})^i\mu = |I|/2 \leq 2^q \leq |I|$. If $t_h = k2^q$ then in the CommonClock(I_j^*)-solution all messages in P_h^* may arrive at s at times t_h

or $t_h + 2^q$. If $t_h \neq k2^q$ then I contains two different multiples of 2^q, say $k2^q$ and $(k+1)2^q$, such that $k2^q < t_h < (k+1)2^q$. In this case, since $|I_j^*| = |I|/2 \geq 2^{q-1}$ $\forall j$, we have $\forall j \in P_h^*$ that $I_j^* \cap \{k2^q, k2^q + 2^{q-1}, (k+1)2^q\} \neq \emptyset$. Lemma 3.6 implies that in a CommonClock(I_j^*)-solution every message $j \in P_h^*$ arrives at s at one of $\{k2^q, k2^q + 2^{q-1}, (k+1)2^q\}$. Hence, $\forall h = 1, \ldots, \ell$, all messages in P_h^* arrive at s at at most 3 distinct time instants in the CommonClock(I_j^*)-solution. CommonClock(I_j^*) does not delay messages at intermediate nodes. This implies that the nodes used by messages in P_h^* are traversed by these messages at most 3 times in the CommonClock(I_j^*)-solution, proving the lemma. □

This suffices to prove the next theorem.

Theorem 4.11. CommonClock(I_j^*) *is* $(\frac{\log U}{\log\log U}, \frac{\log U}{\log\log U - \log\log\log U})$*-competitive.*

Proof. We observe that $\bigcup_{i\in\mathcal{I}} M_i^* = M$, and $|\mathcal{I}^*| = O(\frac{\log U}{\log\log U - \log\log\log U})$. This can be seen as follows. It is easy to see that $j \in M_i^*$ for some $i \in \mathbb{N}$. Let $k := \frac{\log U}{\log\log U - \log\log\log U}$, the highest index in $\mathcal{I}^*$. This gives $\log(\frac{\log U}{\log\log U})k = \log U$, hence $(\frac{\log U}{\log\log U})^k = U$. Thus each $j \in M$ such that $|I_j| < (\frac{\log U}{\log\log U})^k = U$ is contained in $M_1^*, \ldots, M_k^*$.

It follows from Lemma 4.10 that CommonClock(I_j^*) is $(\frac{\log U}{\log\log U}, 3)$-competitive for all messages $j \in M_i^*$. The proof now follows from the observations that the set of all messages is $\bigcup_{i\in\mathcal{I}} M_i^*$. □

Recall from Theorem 3.8 that CommonClock is $(1, \log U)$-competitive, thus an excess in the budget on the latency costs of $O(\log\log U)$ results in a decrease of the competitive ratio on communication costs by factor $O(\log\log U)$.

4.4 Conclusion and open problems

In this chapter we presented online distributed algorithms for the bicriteria data aggregation problem with objectives to minimize communication and latency costs. We focused on algorithms for instances with constant message latencies. We also showed for one of our algorithms how it can be adapted to take into account arbitrary messages latencies. We considered algorithms under three different time models.

For the synchronous time model we presented a $(2, 2)$-competitive algorithm, i.e. it balances the communication and latency costs. We also showed that no distributed algorithm can be $(2, \alpha)$-competitive for $\alpha < 2$.

For the asynchronous time model we presented an algorithm which balances the communication and latency costs up to a factor $\log U$, where U is the ratio between maximum and minimum of the maximum allowed packet delay. We showed that no memoryless algorithm can have a competitive ratio which is more than a factor $\log U$ better than ours, and in case the latency budget is not too small our algorithm is best possible within the class of memoryless algorithms.

For the almost synchronous time model we presented an algorithm which minimizes communication costs under a small excess of the latency budget which depends

linearly on the clock drift. This is the first analysis of algorithms for this model, which models actual wireless networks closer than known time models.

The competitive ratio of our asynchronous algorithm is almost balanced; it would be interesting to find an algorithm with balanced ratios, equal to the lower bounds we presented in this chapter. Another possibility of future research is to make a more careful analysis of the almost synchronous time model, in order to determine the maximum clock drift for which almost synchronous algorithms have better competitive ratio than asynchronous algorithms.

Chapter 5

Data gathering with interference minimizing completion times

5.1 Introduction

In this chapter we study a gathering problem in a wireless network under interference of radio signals, the wireless gathering problem (WGP). We consider a wireless network where data is released over time at the nodes, and nodes should communicate all data to the sink using multi-hop communication. Each node communicates using a wireless transmitter which can send data packets to nodes within communication radius d_T, and causes interference at nodes within interference radius d_I.

WGP consists of finding an interference free schedule, in which packets are sent to the sink as fast as possible. We use completion times and flow times as performance measures for the schedule. In this chapter we consider the objective minimizing the completion time. In the next chapter we consider the objective minimizing the flow time. A completion time model is appropriate for wireless networks which partition data reception and data communication in two phases [34, 39], a flow time model is appropriate for wireless networks where data reception and communication occur simultaneously.

Mathematically, the wireless gathering problem is a generalization of the classic packet radio network model [8, 13]. Given is a graph $G = (V, E)$ with $|V| = n$, sink $s \in V$, and a set of packets $M = \{1, 2, \ldots, m\}$. We assume that each edge has unit length. For each pair of nodes $u, v \in V$ we define the *distance* between u and v, denoted by $d(u, v)$, as the length of a *shortest path* from u to v in G. We have integers d_T and d_I, for communication radius and interference radius and naturally we have $d_I \geq d_T$. Each packet $j \in M$ has a *release node* $v_j \in V$ and a release date $r_j \in \mathbb{Z}_+$ at which it enters the network.

We assume that time is discrete; we call a time unit a *round*. The rounds are

numbered $0, 1, \ldots$. During each round a node may either be *sending* a packet, be *receiving* a packet or be *inactive*. If $d(u, v) \leq d_T$ then u can send some packet j to v during a round. If node u sends a packet j to v in some round, then the pair (u, v) is called a *call* of packet j during that round. Two calls (u, v) and (u', v') *interfere* if $d(u', v) \leq d_I$ or $d(u, v') \leq d_I$; otherwise the calls are *compatible*. We assume that packets cannot be aggregated at nodes.

The solution to the WGP is a schedule of compatible calls such that all packets are sent to the sink. Given a schedule, let v_j^t be the node of packet j at time t. The completion time of a packet j is $C_j = \min\{t : v_j^t = s\}$. To be precise, a packet j can be sent for the first time in round r_j, and v_j^t is the position of the packet at the start of round t. In this chapter we consider the minimization of $\max_j C_j$, called the *makespan*, and we denote this version of WGP as C-WGP. In the next chapter we consider minimizing the flow time.

We use complexity theory to assess WGP, and we use concepts from approximation theory and competitive analysis to analyze the performance of algorithms that we propose for WGP. See Chapter 1 for an introduction to these concepts.

In Section 5.2 we analyze the complexity of C-WGP. We demonstrate that C-WGP is **NP**-hard in case $d_I = d_T$, even in case each node contains exactly one packet. This settles an open problem, posed in [13]. In Section 5.3 we consider online algorithms. We give lower bounds on the competitiveness of online algorithms, and we present a class of greedy online algorithms, which send packets over shortest paths. We prove that this class of algorithms is constant competitive, where the constant depends on d_T and d_I. We demonstrate that a particular greedy algorithm is 4-competitive for arbitrary d_T and d_I, and the algorithm is best possible within the class of shortest paths following algorithms; in case $d_I = d_T$ the algorithm is 3-competitive. We also discuss algorithms and current status of complexity in case the network is a tree or a line. In Section 5.4 we summarize our results, and pose some open problems related to the wireless gathering problem.

Related work

The wireless gathering problem was introduced by Bermond et al. [13], as MINIMUMGATHERINGTIME in the context of providing wireless access to the internet in rural areas. The authors proved that C-WGP is **NP**-hard in case $d_I = d_T$, and in case $d_I > d_T$ the problem remains **NP**-hard even when we restrict to instances with exactly one packet per node. They presented an algorithm which pipelines packets in the network, and proved that this algorithm is 4-approximate, in an offline setting without release dates.

Bermond et al. [11] considered the problem in which each node contains exactly one packet. For this case they presented an optimal algorithm for a half-line, in case $d_T = 1$, and for a line with sink in the center, in case $d_T = 1$ and $d_I \leq 4$.

Bar-Yehuda et al. [8] considered distributed algorithms for C-WGP, in case $d_T = d_I = 1$, and there are no release dates. In their model communication proceeds in phases, and nodes compete for medium access during a phase using the backoff strategy of [7], which is described in more detail in Subsection 2.2.2. The strategy requires nodes to know an upper bound on the maximum graph degree.

Florens et al. [34] studied minimization of the completion time of data gathering in a setting with unidirectional antennas. The main difference with WGP is that unidirectional antennas have a more relaxed notion of interference. E.g., if $d_T = d_I = 1$ a node adjacent to a sender node, but positioned further away from the sink than this sender node, may receive a packet in case of unidirectional antennas, because sender nodes direct communication towards the sink. In our model such a node would receive interfering signals from any adjacent sender node. In case of unidirectional antennas, Florens et al. provided a 2-approximation algorithm for tree networks and an optimal algorithm for line networks. Gargano and Rescigno [39] studied the same problem under the restriction that each node contains exactly one packet. For this model they gave a polynomial time algorithm for arbitrary graphs. This model with one packet per node represents instances with a uniform data distribution, i.e. where the number of packets per node is more or less equivalent.

The gathering problem can be seen as a special case of the packet routing problem in which packets have to be sent from origin node to destination node. Packet routing under interference has been considered using the distance-2 interference model. In this model, the distance between the nodes of two calls should be at least 2. We observe the following relation between distance-2 interference models and WGP interference models: each feasible distance-2 interference schedule is a feasible WGP schedule for $d_T = 1$ and $d_I = 1$, and each feasible WGP schedule for $d_T = 1$ and $d_I = 2$ is a feasible distance-2 interference schedule.

Kumar et al. [55] presented a decentralized algorithm for packet routing under the distance-2 interference model. The authors presented an $O(\Delta \log^2 n)$- approximation algorithm, where Δ is the maximum graph degree, and n is the number of nodes. Their algorithm assumes each node knows upper bounds on maximum number of packets per node, and the network diameter. Especially the first assumption seems restrictive from a practical point of view, where packets arrive continuously. The algorithm proceeds in phases, and at the start of each phases nodes communicate with nodes up to a distance 3 to determine interference free schedules for the round in the next phase. As such the algorithm is decentralized, but not distributed in the sense that nodes use information of neighboring nodes. Hence, this model is more restrictive than the distributed model of Bar-Yehuda et al. [8].

There is a large amount of literature on packet routing, in case interference is not taken into account. Most related to our work on WGP is the work of Cidon et al. [25] and of Mansour and Patt-Shamir [59]. Both articles consider shortest path packet routing using greedy algorithms, under arbitrary release times. Note that in the absence of interference and with fixed paths, the distinction between centralized and distributed algorithms does not seem to be of importance, when designing algorithms. The authors of [59] prove that for any greedy algorithm which uses shortest paths the flow time of a packet is at most the distance to the sink plus the number of packets in the network. However, we observe that they do not consider interferences.

5.2 Complexity

In this section we analyze the complexity of C-WGP. We prove that C-WGP is **NP**-hard, in case $d_I = d_T = 1$, in case each node contains exactly one packet. This settles an open problem.

In an earlier work, Bermond et al. [13] demonstrated **NP**-hardness of C-WGP in case $d_I = d_T$, using a reduction from the satisfiability problem (SAT). Also, Gargano [38] outlined an **NP**-hardness proof using a reduction from CHROMATIC NUMBER, which she attributed to [27]; however, the final version of [27] does not contain the proof. CHROMATIC NUMBER provides a natural framework for a reduction of network routing problems under interference. Also, [55] used a reduction based on CHROMATIC NUMBER, to demonstrate **NP**-hardness of packet routing under a distance-2 interference model.

The authors of [13] also considered C-WGP, for the case where all nodes contain exactly one packet. They posed the following problem: is C-WGP **NP**-hard in case $d_I = d_T = 1$ and each node contains exactly one packet? We present an affirmative answer to this question. First, we describe the problem CHROMATIC NUMBER:

CHROMATIC NUMBER
Instance: a graph G and a number k.
Question: does G have chromatic number at most k?

The chromatic number of a graph is the minimum number of colors needed to color all nodes of the graph such that no two adjacent nodes have the same color. Given a graph G we denote the chromatic number of G as $\chi(G)$. The problem is known to be **NP**-complete [37]. We use this fact to derive **NP**-hardness for C-WGP.

Theorem 5.1. C-WGP *is* **NP***-hard, in case* $d_I = d_T = 1$ *and each node contains exactly one packet.*

Proof. Given is an instance of CHROMATIC NUMBER, on graph G, with $V(G) = \{u_1, \ldots, u_n\}$. Let K be a complete graph with $V(K) = \{v_1, \ldots, v_n\}$. We construct a graph H with $V(H) = V(G) \cup V(K) \cup \{s\}$ and $E(H) = E(K) \cup \{(s, v), v \in V(K)\} \cup \{(u_i, v_j), \forall i, j \text{ s.t.} (u_i, u_j) \in E(G)\} \cup \{(u_i, v_i), i = 1 \ldots, n\}$. We are given an instance of C-WGP on graph H with $d_I = d_T = 1$, sink s, and one packet released at each node, except the sink. See Figure 5.1.

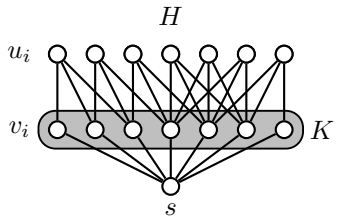

Figure 5.1: The construction in the proof of Theorem 5.1, with $d_T = d_I = 1$.

We prove the theorem by showing that if G has chromatic number at most k, then there is a schedule for the C-WGP instance on H with makespan at most $k+2n$, while if G has chromatic number at least $k+1$, then every schedule for the C-WGP instance on H has makespan at least $k+2n+1$. The theorem then follows from **NP**-completeness of CHROMATIC NUMBER.

Suppose G has chromatic number at most k. Then k rounds suffice to send each packet released at u_i to v_i, $i = 1, \ldots, n$. This can be seen as follows. Consider a coloring of G of at most k colors. Two nodes u_i and u_j are in the same color class only if $(u_i, u_j) \notin E(G)$, hence only if $\{(u_i, v_j), (u_j, v_i)\} \cap E(H) = \emptyset$. Hence all nodes u of the same color can send a packet in the same round to some node v, in particular u_i can send a packet to v_i. Next, another $2n$ rounds suffice to gather all $2n$ packets at s. Hence, the makespan is at most $k+2n$.

Suppose G has chromatic number at least $k+1$. Then, again relating colors to rounds, any algorithm requires at least $k+1$ rounds to send all n packets from a node u to some node v. Also, any algorithm requires $2n$ rounds to send all $2n$ packets from a node v to s, and if a packet is sent from some node $v \in V(K)$, then no node of $V(K)$ can receive a packet because the component K is a complete graph. Hence, the makespan is at least $k+2n+1$. □

The complexity of C-WGP on special instances, e.g. in case the network is a line or a tree, is still an open problem. In the next section we present some partial results.

5.3 Online algorithms

In this section we provide lower bounds on the competitiveness of online algorithms for C-WGP, we present a class of online deterministic algorithms, called shortest paths following algorithms, and we analyze the competitive ratio of algorithms in this class. Most important result is that we present a shortest paths following algorithm with a competitive ratio which is best possible within this class of algorithms.

5.3.1 Lower bounds

First, we provide a general lower bound on the competitive ratio of any online algorithm for C-WGP.

Lemma 5.2. *No deterministic algorithm is better than 4/3-competitive for $d_I = d_T$. No randomized algorithm is better than 7/6-competitive for $d_I = d_T$.*

Proof. See Figure 5.2 on the next page. The adversary releases packet 1 at u at time 0. Observe that for any algorithm that does not send packet 1 in the first round the lemma trivially holds.

First, consider the class of deterministic algorithms. If a deterministic algorithm chooses to send packet 1 to u_1 (u_2) the adversary releases a second packet at u_3 (u_4) at time 1 and hence the algorithm incurs a makespan of at least 4. In the optimal schedule packet 1 is sent to u_2 (u_1) in the first round which yields a makespan of 3.

Consider the class of randomized algorithms. We apply Yao's minimax principle (see [62]). The adversary releases a second packet in round 1 either at u_3 or u_4, both with probability 1/2. Now, the expected number of rounds for any algorithm that sends a packet in the first round is $1/2 * 4 + 1/2 * 3$. In the optimal schedule packet 1 is send to u_2 (u_1) in the first round if the adversary releases a packet at u_3 (u_4) which yields a makespan of 3. □

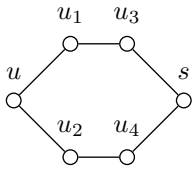

Figure 5.2: No deterministic algorithm is better than 4/3 competitive for $d_I = d_T$.

We can obtain similar results for arbitrary d_T and d_I. The example of Lemma 5.2 contains three packets, and we do not see a way to generalize the example to an instance with more than three packets, without decreasing the lower bound. As such the lower bound is mainly of theoretical use, because the lower bound does not exclude algorithms which are asymptotically near-optimal.

Next, we present a lower bound on the competitiveness of algorithms which send packets along shortest paths. We call such algorithms shortest paths following algorithms.

The motivation to study shortest paths following algorithms, is that they form a natural class of routing algorithms. In Chapter 2 we outlined some advantages regarding network design which stimulate the use of a shortest paths routing tree. Another motivation comes from work of Cidon et al. [25]; the authors demonstrated that for some well known greedy algorithms there is a gap between the completion times of routing over arbitrary paths, and over shortest paths, in favor of routing over shortest paths. The gap depends on the root of the number of packets.

We show that no algorithm which sends each packet j over a shortest path from v_j to s can be better than 3-competitive if $d_I = d_T$, or better than 4-competitive if $d_I > d_T$, asymptotically. To be precise the lower bounds are $3 - 9/(m+3)$ and $4 - 16/(m+4)$. As can be observed the lower bounds also hold for offline shortest paths following algorithms, i.e. the bounds are lower bounds on the approximability as well.

First, consider the example of Figure 5.3 with $d_I = d_T = 1$, on the next page. Nodes u_1, u_2, u_3 have $m/3$ packets each. Any shortest paths following algorithm sends all packets via u, yielding $\max_j C_j = 3m$. On the other hand there is a solution with no packet passing u that implies $\max_j C_j^* \leq 3 + m$. The example can easily be extended for arbitrary $d_I = d_T$ such that any shortest paths following algorithm is 3-competitive.

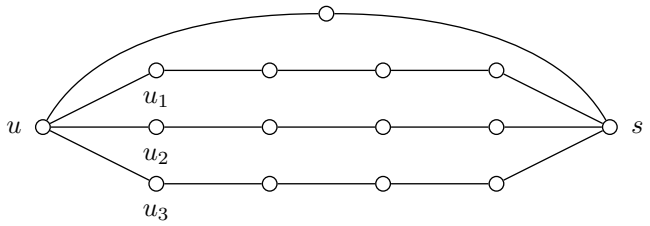

Figure 5.3: No shortest paths following algorithm is better than 3-competitive for $d_I = d_T = 1$.

In case $d_I > d_T$ consider Figure 5.4. The nodes $u_1, \ldots, u_m$ each have 1 packet. Let $d_I = 2$ and $d_T = 1$. Any shortest paths following algorithm sends all packets via u, yielding $\max_j C_j = 4m$. There is a solution with no packet passing u that implies $\max_j C_j^* \leq 4 + m$. The example can easily be extended for arbitrary $d_I = 2d_T$ such that any shortest paths following algorithm is at least 4-competitive.

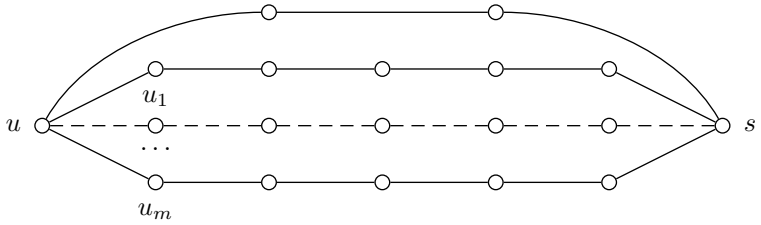

Figure 5.4: No shortest paths following algorithm is better than 4-competitive for $d_I = 2, d_T = 1$.

In the lower bound examples we presented above, the optimal schedule sends each packet over a path whose length exceeds the shortest path length by at most 1. This may suggest to consider algorithms which send packets over paths whose length does not exceed their shortest path length by some constant k. However, as can easily be verified, for each constant k we could change the length of the paths in the examples above, such that the optimal schedule sends each packet over a path whose length exceeds the shortest path length by $k + 1$.

The examples indicate that if there exist algorithms which have a competitive ratio that is lower than the lower bound of shortest paths following algorithms, then these algorithms should balance the number of packets sent over paths towards the sink. One idea for such an algorithm is to balance the number of packets over paths is to use not only the shortest path but the k shortest paths, whichever of them is least congested. However, by adapting the example of Figure 5.3, we obtain the example in Figure 5.5 with the same packets; see the next page. This example shows that for $k = 2$, even if we choose the 2 shortest internally vertex disjoint paths, the lower bound on the competitive ratio for $d_I = d_T$ remains unchanged. The example

can be extended to any fixed k. Similarly the example of Figure 5.4 can be adapted.

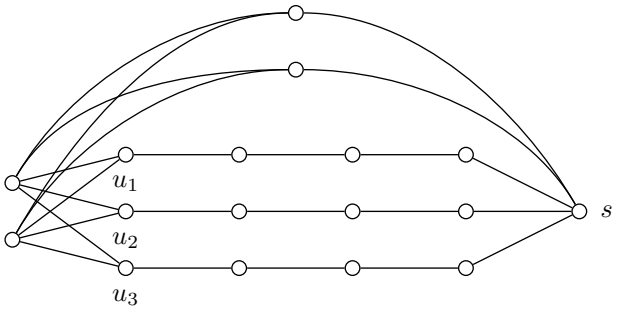

Figure 5.5: Any shortest paths following algorithm using 2 internally vertex disjoint shortest paths is no better than a 3-competitive for $d_I = d_T = 1$.

5.3.2 A greedy class of algorithms

We present a class of online algorithms for C-WGP. Each algorithm of this class orders packets according to some priority rule. This priority rule can be any arbitrary but fixed ordering of packets. Then, in each round the algorithm creates a schedule of interference free calls. The algorithm is a type of *greedy* algorithm. A greedy algorithm makes decisions sequentially, and each decision is the solution to an auxiliary optimization problem, whose input data depend on previous decisions.

Algorithm 10 PRIORITY GREEDY

Order packets according to some priority rule.
while not all packets have reached s **do**
- Consider next round t;
- Consider packets by non-increasing priority; send each next packet j as far as possible along a shortest $v_j^t - s$ path, without creating interference with any higher-priority packet.

end while

PRIORITY GREEDY is an online, polynomial-time algorithm. In each round the algorithm sequentially minimizes the distance to the sink of packets, under the condition that each packet must be sent over some shortest path.

We analyze the competitive ratio of PRIORITY GREEDY algorithms. First, we introduce some extra notation. Let $\delta_j = \lceil \frac{d(v_j,s)}{d_T} \rceil$, the minimum number of calls required for packet j to reach s. We define the critical radius R^* as the greatest integer R such that no two nodes at distance at most R from s can receive a packet in the same round. The critical region is the set of nodes at distance at most R^* from s. We define $\gamma = 1 + \lceil \frac{d_I+1}{d_T} \rceil$, and $\gamma^* = \lceil \frac{R^*+1}{d_T} \rceil$ where R^*, $\lfloor \frac{d_I-d_T}{2} \rfloor \leq R^* \leq d_I + 1$,

depends on graph G. We will use the ratio γ/γ^* in our analyses. This ratio depends on d_T, d_I, and R^*. The following lemma provides a useful upper bound on γ/γ^*.

Lemma 5.3. $\frac{\gamma}{\gamma^*} \leq 2 + \frac{4}{l+1}$ *if* $ld_T \leq d_I \leq (l+2)d_T - 1, \forall$ *odd* $l \in \mathbb{N}$. *And* $\frac{\gamma}{\gamma^*} \leq 3$ *if* $d_I \leq 2d_T - 1$.

Proof. Given are $d_T, d_I \in \mathbb{N}$, $d_I \geq d_T$. Note that $\gamma^* = \left\lceil \frac{R^*+1}{d_T} \right\rceil \geq 1$. Let l be an odd integer such that $ld_T \leq d_I \leq (l+2)d_T - 1$. Then this yields $\gamma \leq l + 3$ and as $R^* \geq \lfloor \frac{d_I - d_T}{2} \rfloor$ it follows that $\gamma^* \geq (l+1)/2$ which proves the first part of the lemma. If $d_I \leq 2d_T - 1$ then $\gamma = 3$, which proves the second part of the lemma. □

We say that packet j is *blocked* in round t if, in round t, j is not sent over a shortest path towards s over distance d_T (or it is not sent to s if $d(v_j^{t-1}, s) \leq d_T$). We define the following *blocking relation* on a PRIORITY GREEDY schedule: $k \prec j$ if in the last round in which j is blocked, k is the packet closest to j that is sent in that round and has a priority higher than j (ties broken arbitrarily). The blocking relation induces a directed graph $F = (M, A)$ on the packet set M with an arc (k, j) for each $k, j \in M$ such that $k \prec j$. Observe that for any PRIORITY GREEDY schedule F is a directed forest and the root of each tree of F is a packet which is never blocked. For each j let $T(j) \subseteq F$ be the tree of F containing j, $b(j) \in M$ be the root of $T(j)$, and $P(j)$ the path in F from $b(j)$ to j. Let $\pi_j = \min\{\delta_j, \gamma^*\}$, and $R_j = r_j + \delta_j - \pi_j$.

To analyze the competitive ratio of PRIORITY GREEDY we derive an upper bound on the completion time C_j of each packet j in a PRIORITY GREEDY schedule, and a lower bound on the maximum completion time of an optimal schedule.

Lemma 5.4. *For each packet* $j \in M$,

$$C_j \leq R_{b(j)} + \sum_{i \in P(j)} \min\{\delta_i, \gamma\}.$$

Proof. The proof is by induction on the length of $P(j)$. Any packet j with $|P(j)| = 1$ is never blocked, hence $b(j) = j$, and the lemma is obviously true.

Otherwise, let t be the last round in which j is blocked and let k be the packet such that $k \prec j$. By definition of the blocking relation we have $d(v_j^t, v_k^t) \leq d_T + d_I$ and if $d(v_j^t, v_k^t) > d_I + 1$ then j, although blocked, is sent to v_j^{t+1} with $d(v_j^{t+1}, v_k^t) = d_I + 1$. Also, $d(v_k^t, s) \leq (C_k - t)d_T$, otherwise k would not reach s by time C_k. From time $t+1$ on, j is forwarded to s over distance d_T each round, reaching s at

$$\begin{aligned} C_j &\leq t + 1 + \left\lceil \frac{d(v_k^t, s) + d(v_j^{t+1}, v_k^t)}{d_T} \right\rceil \leq t + 1 + C_k - t + \left\lceil \frac{d_I + 1}{d_T} \right\rceil \\ &= C_k + 1 + \left\lceil \frac{d_I + 1}{d_T} \right\rceil = C_k + \gamma. \end{aligned}$$

Also, $C_j \leq C_k + \delta_j$, since after k reaches s, j will need no more than δ_j rounds to reach s. Thus $C_j \leq C_k + \min\{\delta_j, \gamma\}$ and the lemma follows by applying the induction hypothesis to C_k. □

Let C_j^* denote the completion time of packet j in an optimal solution.

Lemma 5.5. *Let $S \subseteq M$ be a nonempty set of packets. Then there is $k \in S$ such that*

$$\max_{j \in S} C_j^* \geq R_k + \sum_{j \in S} \pi_j.$$

Proof. Since in every round at most one packet can move inside the critical region, any feasible solution to WGP gives a feasible solution to a preemptive single machine scheduling problem in which the release time of job j (corresponding to packet j) is R_j and its processing time is π_j. By ignoring interference outside the critical region we can only decrease the optimum cost, thus a lower bound on the scheduling cost is also a lower bound on the gathering cost.

Now let k be the first packet in S entering or being released in the critical region in the optimal schedule. In the scheduling relaxation, the makespan is at least the time at which the first job starts processing plus the sum of the processing times which is precisely what is stated in the lemma. □

Lemma 5.4 and Lemma 5.5 suffice to provide an upper bound on the competitive ratio of any Priority Greedy algorithm.

Theorem 5.6. *For any priority function* Priority Greedy *is (γ/γ^*+1)-competitive.*

Proof. Let j be the packet having maximum C_j, and consider $T(j)$, the tree containing j in the forest induced by the blocking relation. We can apply Lemma 5.5 with $S = T(j)$ to obtain

$$\max_{i \in T(j)} C_i^* \geq R_k + \sum_{i \in T(j)} \pi_i = r_k + \delta_k + \sum_{i \in T(j), i \neq k} \min\{\delta_i, \gamma^*\}$$

where k is some packet in $T(j)$. Also, we have $\max_{i \in T(j)} C_i^* \geq r_{b(j)} + \delta_{b(j)}$. On the other hand, by using Lemma 5.4,

$$\begin{aligned} C_j &\leq r_{b(j)} + \delta_{b(j)} - \min\{\delta_{b(j)}, \gamma^*\} + \frac{\gamma}{\gamma^*} \min\{\delta_k, \gamma^*\} + \frac{\gamma}{\gamma^*} \sum_{i \in P(j), i \neq k} \min\{\delta_i, \gamma^*\} \\ &\leq r_{b(j)} + \delta_{b(j)} + \frac{\gamma}{\gamma^*}[(r_k + \delta_k) + \sum_{i \in T(j), i \neq k} \min\{\delta_i, \gamma^*\}] \\ &\leq (1 + \gamma/\gamma^*) \max_{i \in T(j)} C_i^*. \end{aligned}$$

where we used the fact that $\min\{\delta_k, \gamma^*\} \leq (r_k + \delta_k)$. □

The theorem demonstrates that a class of greedy algorithms with a fixed priority ordering on the packets is constant competitive. The property that the ordering is fixed is necessary to obtain this constant competitive ratio. This can be seen as follows. Consider a line graph of length δ with end nodes v and s, and n packets with release node v. We consider a packet ordering which in each round is based on the distance of a packet to the sink. Note that this ordering may change over time.

An algorithm which sends the furthest packet first yields a schedule with makespan $O(\delta/d_T n)$. By Theorem 5.6 any Priority Greedy algorithm yields a makespan of at most $\delta + \gamma/\gamma^* n$, hence the gap between the competitive ratio of these two classes of algorithms is $O(\delta)$, which depends on the network size.

Next, we consider two Priority Greedy algorithms with a specific priority, which we call LIS and RPG.

The LIS algorithm

Algorithm LIS is a Priority Greedy algorithm with priority ordering based on release times r_j: if $r_j < r_k$ then j will have higher priority than k, ties broken arbitrarily. We call this algorithm LIS (longest in system), because in each round it sends the packet which is longest in the system first.

The following theorem demonstrates that LIS can be strictly worse than γ/γ^* competitive.

Theorem 5.7. LIS *is strictly worse than γ/γ^*-competitive.*

Proof. We are given an instance on a graph as depicted in Figure 5.6, with $d_T = 1, d_I = 2$.

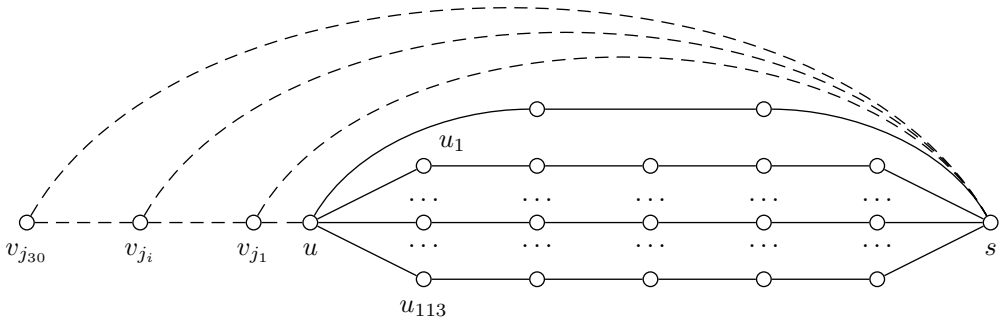

Figure 5.6: LIS is strictly worse than a γ/γ^*-approximation for $d_I = 2, d_T = 1$.

The instance consists of 30 packets $j_i, i = 1 \ldots, 30$, with $\delta(j_i) = 5i - 1, r_{j_i} = 0$. There are also 113 packets j'_k with release node $u_k, k = 1, \ldots, 113$, $\delta(j'_k) = 4, r_{j'_k} = 1$.

We assume that LIS has fixed the paths of all packets, such that all packets pass node u. The optimal algorithm can send each packet j_i from its release node over path of length $\delta(j_i)$ towards the sink, and each packet j'_k over a path of length $\delta(j'_k) + 1$, such that any two of these paths are internally vertex disjoint.

In the LIS solution packets j_i are sent without any delay. Packets j'_k are delayed by all packets j_i, hence $j'_k \in T(j_{30})$ for each packet j'_k. Because all packets j'_k are sent over the same $u - s$ path the last packet has completion time $r_{j_{30}} + \delta(j_{30}) + 113 \cdot 4 = 601$.

Consider the following solution: all packets j'_k with release node u_k are sent to the node on the $u_k - s$ path adjacent to s simultaneously. Because all packets

j'_k have release time 1 this is possible, and hence in round 5 all 113 packets j are adjacent to the sink. Moreover, in rounds 0 to 4 the packets j_i are sent towards the sink over the $v_{j_i} - s$ path that does not pass u. Also these packets can be sent simultaneously, hence in round 5 packet j_i is at distance $\delta(j_i) - 5 = 5i - 6$ from the sink for $i \in \{2, \ldots, 30\}$, and j_1 is at the sink. In rounds 5 to 117 the packets j'_k are sent to the sink, and packets j_i are forwarded to a node at distance 2 from the sink. Note that this is feasible, because packets j_i do not interfere with each other or with a packet j'_k which is sent to the sink. Thus, in round 118 all packets j'_k are at the sink, packets $j_2, \ldots, j_{24}$ are at distance 2 from the sink, and packet j_i is at distance $5i - 119$ for $i \in \{25, \ldots, 30\}$. In round 118 all packets at distance 2 are sent simultaneously to a node adjacent to the sink; also all packets j can be forwarded. Thus, in round 119 packets $j_2, \ldots, j_{24}$ are adjacent to the sink, and packet j_i is at distance $5i - 120$ for $i \in \{25, \ldots, 30\}$. In rounds 119 to 141 the packets $\{j_2, \ldots, j_{24}\}$ are sent to the sink, and packets $\{j_{25}, \ldots, j_{30}\}$ are forwarded towards a node at distance 2 from the sink. Hence, in round 142 packets $j_{25}, \ldots, j_{29}$ are at distance 2 from the sink, and packet j_{30} is at distance 7 from the sink. In round 142 all packets at distance 2 are sent simultaneously to a node adjacent to the sink; also packet j_{30} can be forwarded. In rounds 143 to 147 packets $j_{25}, \ldots, j_{29}$ are sent to the sink, and after this packet j_{30} needs two more rounds to arrive at the sink. Hence the completion time of this schedule is 150 rounds.

This proves the theorem because $601/150 > 4 = \gamma/\gamma^*$. □

We study LIS in more detail in the next chapter when minimizing flow time.

The RPG algorithm

Algorithm RPG is a Priority Greedy algorithm with priority ordering based on R_j: if $R_j < R_k$ then j will have higher priority than k, ties broken arbitrarily. The following theorem provides an upper bound on the competitive ratio of RPG, that improves upon the bound of Theorem 5.6.

Theorem 5.8. RPG *is γ/γ^*-competitive.*

Proof. Let j be the packet having maximum C_j, and consider $T(j)$, the tree containing j in the forest induced by the blocking relation. We can apply Lemma 5.5 with $S = T(j)$ to obtain

$$\max_{i \in T(j)} C^*_i \geq R_k + \sum_{i \in T(j)} \pi_i \tag{5.1}$$

where k is some packet in $T(j)$. On the other hand, by using Lemma 5.4,

$$C_j \leq R_{b(j)} + \sum_{i \in P(j)} \min\{\delta_i, \gamma\} \leq R_{b(j)} + \frac{\gamma}{\gamma^*} \sum_{i \in P(j)} \pi_i \leq R_k + \frac{\gamma}{\gamma^*} \sum_{i \in P(j)} \pi_i \tag{5.2}$$

where we use that $b(j)$, being the root of $T(j)$, has minimum R_i-value amongst the nodes in $T(j)$. The theorem follows by direct comparison of (5.1) and (5.2). □

Corollary 5.9. RPG *is* 3*-competitive if* $d_I = d_T$ *and* 4*-competitive in general for* WGP.

Proof. Directly from Lemma 5.3 and Theorem 5.8. □

It follows from the lower bounds on shortest paths following algorithms and the corollary above that RPG is best possible in the class of shortest paths following algorithms. Also, it follows from Theorem 5.7 and Theorem 5.8 that not all Priority Greedy algorithms have the same competitive ratio. Finally, there may be algorithms which have a competitive ratio which is lower than that of RPG. But, as we observed above such an algorithm should diverge packets from their shortest path to the sink if this path becomes congested, and analyzing such an algorithm may not be straightforward.

5.3.3 Special instances

We briefly discuss algorithms for special networks which are trees or lines. Bermond et al. [11] presented an algorithm, and demonstrated it is optimal on a half-line, in case $d_T = 1$. In a subsequent paper [12] near-optimal algorithms for lines and other special graph instances are claimed, but proofs are omitted.

We show that RPG is optimal in a special case on a line network, similar to the result in [11]. And we demonstrate that RPG can be non-optimal on arbitrary trees. As such, the existence of a polynomial time optimal algorithm for C-Wgp remains an open problem.

Corollary 5.10. RPG *is optimal if* G *is a half-line and* $d_T = 1$.

Proof. If G is a half-line, then s is an endpoint of the line. In this case the critical radius is $d_I + 1$. Thus, for $d_T = 1$ we have $\gamma^* = d_I + 2$. The claim follows since $\gamma = d_I + 2$ for $d_T = 1$. □

The corollary demonstrates optimality of a rather restrictive class of instances. In fact, the corollary does not hold if we relax either the assumption that the graph is a half-line, or the assumption that communication radius is unit. We present examples which demonstrate that in either case RPG is not necessarily optimal.

RPG can be non-optimal on a line if the sink is not at an extreme node of the line. Consider the instance given by the graph in Figure 5.7. Let $d_T = 1$ and $d_I = 2$ and assume packets are released in u_1, u_2 and u_3 at time zero. RPG would first send the packet in u_1 to the sink, and then send the packet in u_2 to the sink, resulting in a makespan of 7, while in an optimal solution the packets in u_2 and u_3 are forwarded until the packet of u_2 reaches the sink at time 2, then the packets of u_3 and u_1 are forwarded simultaneously, yielding a cost of 6. Note that if the third packet would have been released at node u_3' instead of at node u_3 then RPG would have been optimal. This shows that any optimal algorithm for the problem on a line should take into account the position of the other packets when deciding which of the two packets nearest to the sink to send first.

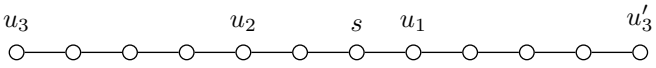

Figure 5.7: RPG is not optimal on a line if the sink is not at an endpoint.

RPG can be non-optimal on a half-line if $d_T > 1$. Given a line with extreme s. Let $d_T = 3, d_I = 6$ and consider packets 1,2,3 which are released at distance, respectively, 8,9 and 27 from the sink. See Figure 5.8. In this case packets 8 and 9 have the same priority. We consider the case when RPG chooses to send the closest packet, first. If RPG sends packet 1 before packet 2, then packet 1 is sent in rounds 1,2 and 3 and packet 2 is sent in rounds 4,5 and 6. packet 3 is sent in rounds 1 to 10. packet 2 blocks packet 3 in round 4, hence packet 3 needs 10 rounds. In an optimal schedule packet 2 is sent before packet 1. As no packet blocks packet 3 in round 4, this packet needs 9 rounds. The example demonstrates that RPG may be non-optimal if it does not break priority ordering ties properly.

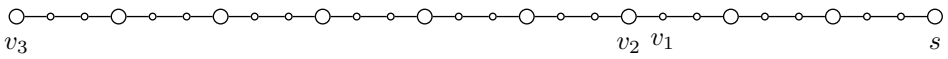

Figure 5.8: RPG is not optimal on a line with extreme s for $d_T = 3, d_I = 6$.

Finally, the results in this chapter are based on examples on arbitrary graphs, which provide a very general model for wireless networks. As described in Chapter 2 there exist other, more restricted, graph models, e.g. unit disk graphs or planar graphs. However, we note that an example similar to that of Figure 5.3 easily leads to a lower bound of 2 on these restricted classes of algorithms: simply consider a network with a shortest path and two other disjoint paths with a length which exceeds the shortest path length by a small constant. A third path with longer length can be added to achieve lower bounds which approach 3, hence the study of these restricted graph classes does not seem to improve the competitive ratio of the algorithms significantly.

5.4 Conclusion and open problems

In this chapter we studied gathering of data packets in a wireless network under interference of radio signals, the wireless gathering problem. In particular, we considered the problem of minimizing the maximum packet completion time, called C-Wgp. We proved that C-Wgp is **NP**-hard, in case the communication radius and interference radius are equal, even when we restrict to the class of instances where each node contains exactly one packet.

We presented a lower bound of 7/6 on the competitive ratio of any algorithm. This lower bound is based on an instance with 3 packets, and increasing the number of packets decreases this value. Therefore, it remains an open problem to determ-

ine whether there exists a constant lower bound for an instance with an arbitrary number of packets. We also presented a lower bound on shortest paths following algorithms, which depends on the communication and interference radius, and ranges between 2 and 4.

We analyzed a class of online greedy algorithms, called PRIORITY GREEDY, that send packets over a shortest path to the sink. We showed that all PRIORITY GREEDY algorithms are constant competitive, with the constant depending on the communication and interference radius. We introduced two PRIORITY GREEDY algorithms, called LIS and RPG. We use LIS extensively in the next chapter when we analyze WGP minimizing flow times.

We showed that RPG is 4-competitive in general, and 3-competitive in case communication radius equals interference radius. In fact, for $d_I = d_T$ and $d_I = 2d_T$ the competitiveness of RPG matches the lower bounds on the competitive ratio of any shortest paths following algorithm, demonstrating that RPG is best possible within this class of algorithms.

The gap between the competitive ratio of RPG and the general lower bound of 7/6 leaves open the possibility that there are algorithms with competitive ratio which is strictly better than that of RPG or any shortest paths following algorithm.

We briefly discussed design implications of algorithms which do not follow shortest paths, and presented an example which demonstrates that finding algorithms with an improved competitive ratio is not straightforward.

We considered C-WGP in special graphs, such as trees and lines. In case the graph is a line, with sink at one end, and unit communication radius, RPG is in fact an optimal algorithm. However, in general it is not known if there exist algorithms for tree graphs which have a competitive ratio strictly better than the competitive ratio of RPG, and also the complexity of C-WGP on a tree is an open problem.

Chapter 6

Data gathering with interference minimizing flow times

6.1 Introduction

In this chapter we consider a wireless gathering problem under interference of radio signals, minimizing packet flow times. We call the problem with objective minimizing maximum flow times F-WGP; this problem is closely related to C-WGP which we considered in the previous chapter. The difference between these two problems lies in the objective; minimizing flow times is the more natural objective in case packets are released over time, and receiving and communicating data occurs simultaneously; see also Chapter 5. We consider both the objective minimizing the maximum flow time, as well as minimizing the average flow time. The first objective provides a quality of service measure for any data packet, the second objective provides a quality of service measure for a packet on average.

We use the same notation as in the previous chapter, in particular an instance of the problem is defined by a graph G, a packet set M, communication radius d_T, and interference radius d_I. We introduce some extra notation specific to the flow version of WGP. Given a schedule of WGP with completion time C_j for packet j, we define the flow time of j as $F_j := C_j - r_j$. Let $\delta = \max_{j \in M} \delta_j$, the maximum number of calls required to send a packet to the sink, and let Δ be the maximum node degree.

We use complexity theory to assess WGP, and we use concepts from approximation theory and competitive analysis to analyze the performance of algorithms that we propose for WGP. In particular we use resource augmentation to assess the competitiveness of our algorithms. See Chapter 1 for an introduction to these concepts.

In Section 6.2 we analyze the complexity of F-WGP. We prove that F-WGP

on m packets cannot be approximated within $\Omega(m^{\frac{1-\epsilon}{2}})$, for any $\epsilon, 0 < \epsilon < 1/2$, unless $\mathbf{P} = \mathbf{NP}$. We also show that any algorithm using shortest paths in order to route the packets to the sink is no better than an $\Omega(m)$-approximation. In Section 6.3 we present a polynomial time resource augmented online algorithm for F-WGP. The algorithm is σ-speed optimal for F-WGP, for $\sigma \geq 5$. In Section 6.4 we study simple distributed algorithms for WGP minimizing flow times. We prove that F-WGP on m packets cannot be approximated within $\Omega(\log m)$ by a general class of distributed algorithms, called simple distributed algorithms. Also, we argue that we need an augmented speed of factor $\Omega(\log m)$ to obtain a constant competitive simple distributed algorithm, contrasting with the resource augmentation results for centralized algorithms. Then, we present a simple distributed algorithm, and we show that using an augmented speed of factor $O(\log \delta \log \Delta)$ times the speed of an offline algorithm the algorithm is in expectation near optimal with respect to average flow times. In Section 6.5 we summarize our results and pose some open problems related to F-WGP.

6.2 Complexity

In this section we analyze the complexity of F-WGP. We use a reduction similar to the reduction of Theorem 5.1. In case of flow times, this yields an even stronger result, namely non-approximability. I.e., we present a lower bound on the approximation ratio which is a polynomial function of the number of messages.

Given a graph G let the chromatic number be $\chi(G)$, and let the size of the maximum independent set in G be $\alpha(G)$. Zuckerman [80] proved that it is **NP**-complete to approximate CHROMATIC NUMBER within $n^{1-\epsilon}$. The proof follows from the following reduction.

Theorem 6.1 (Zuckerman [80]). *For all $\epsilon > 0$, there is a polynomial time reduction from an* **NP***-complete language L to* CHROMATIC NUMBER *with the following properties. On input x, the (reduction) algorithm outputs a graph G on n vertices such that:*
if $x \in L$, then $\chi(G) \leq (1 + \log n)n^{\epsilon}$;
if $x \notin L$, then $\alpha(G) \leq n^{\epsilon}$, so $\chi(G) \geq n^{1-\epsilon}$.

We use this theorem to give a lower bound on the approximability of F-WGP, using the instance on the graph of Figure 5.1.

Theorem 6.2. *Unless* $\mathbf{P} = \mathbf{NP}$*, no polynomial time algorithm can approximate* F-WGP *within ratio $\Omega(m^{\frac{1-\epsilon}{2}})$. for any ϵ, $0 < \epsilon < 1/2$.*

Proof. Given is an instance of CHROMATIC NUMBER, on graph G derived from Theorem 6.1. We construct the same graph H as in the proof of Theorem 5.1, and depicted in Figure 5.1

Our F-WGP instance consists of $m = n^2$ packets released at some node u_i. The packets are divided into n groups of size n. Fix any ϵ, $0 < \epsilon < 1/2$. Each group contains exactly one packet which is released at node u_i, for $i = 1, \ldots, n$. The

release date of each packet in group h is $((1+\log n)n^\epsilon + n)h$, $h = 0, \ldots, n-1$. Rounds $((1+\log n)n^\epsilon + n)h$ till $((1+\log n)n^\epsilon + n)(h+1) - 1$ together are a *phase*.

We prove the theorem by showing that if G has chromatic number at most $(1+\log n)n^\epsilon$, then there is a schedule for the F-WGP instance on H with maximum flow time $n+(1+\log n)n^\epsilon$, while if G has a maximum independent set of at most n^ϵ then every schedule for the F-WGP instance on H has maximum flow time $\Omega(n^{2-\epsilon})$. The theorem then follows from Theorem 6.1, and the observation that the ratio of these maximum flow times is:

$$\Omega(\frac{n^{2-\epsilon}}{n+(1+\log n)n^\epsilon}) = \Omega(n^{1-\epsilon}) = \Omega(m^{\frac{1-\epsilon}{2}}).$$

Suppose $\chi(G) \leq (1+\log n)n^\epsilon$. Then $(1+\log n)n^\epsilon$ rounds suffice to send each packet from u_i to v_i. Next, another n rounds suffice to gather all packets at s. Thus, all packets released in phase h, can reach the sink before phase $h+1$ starts. Therefore, the maximum flow time of any packet in this schedule is at most the length of a phase, i.e. at most $n+(1+\log n)n^\epsilon$.

Suppose $\alpha(G) \leq n^\epsilon$. Then, any algorithm requires at least n^2/n^ϵ rounds to send all n^2 packets from a node u_i to v_i. Also, any algorithm requires n^2 rounds to send all packets from a node v_i to s, and if a packet is sent from some node $v \in V(K)$, then no node of $V(K)$ can receive a packet because the component K is a complete graph. Therefore, any schedule requires at least $n^2 + n^{2-\epsilon}$ rounds to gather all packets. As all packets are released before round $((1+\log n)n^\epsilon + n)(n-1) \leq (1+\log n)n^{1+\epsilon} + n^2$ the maximum flow time is $\Omega(n^{2-\epsilon})$, for any $\epsilon < 1/2$. □

In case the packets are routed via shortest paths to the sink, the result of Theorem 6.2 can be strengthened further.

Theorem 6.3. *No shortest paths following algorithm can approximate* F-WGP *within a ratio better than* $\Omega(m)$.

Proof. Consider the instance in Figure 6.1. The adversary releases a packet at each of the nodes u_1, u_2, u_3 at times $5i$, $i = 0, \ldots, m/3$. Any shortest paths following algorithm sends all packets via u, yielding $\max_j C_j \geq 3m$. As $r_j \leq 5m/3$ for each packet j, we have $\max_j F_j \geq 3m - 5m/3 = 4m/3$.

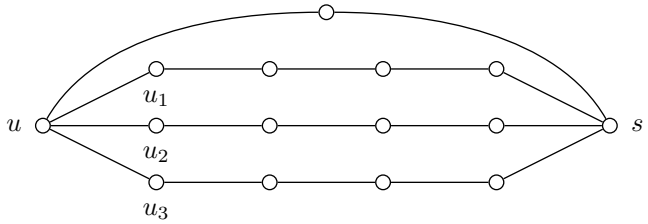

Figure 6.1: No shortest path based algorithm is better than $\Omega(m)$-competitive for $d_T = d_I = 1$.

The adversary sends each packet over the path which does not contain u. We claim that it is possible to do this such that all packets released at time $5i$ have completion time at most $5(i+1)+1$. If the claim holds, then we have $\max_j F_j^* \leq 5(i+1)+1-5i = 6$. This proves the theorem, since then $\max_j F_j / \max_j F_j^* \geq (4m/3)/6 = 2m/9$.

We prove the claim by induction. Suppose the claim holds for packets released in round $5(i-1)$. Then, the last packet released at time $5(i-1)$ latest is sent to the sink in round $5i$. This packet does not block any packet released in round $5i$. Now, the adversary sends the packets released in round $5i$ to a node adjacent to s in 3 rounds, i.e. in the rounds $5i$, $5i+1$ and $5i+2$. Then, it requires another 3 rounds to send all 3 packets to the sink, i.e. the rounds $5i+3$, $5i+4$, and $5(i+1)$, which proves the claim. □

Note that the theorem also provides a lower bound on the competitive ratio of shortest paths following algorithms, because the proof holds for any shortest paths following algorithm.

6.3 An online algorithm

In this section we analyze the Priority Greedy algorithm LIS which sends packets over some shortest path to the sink using a priority order based on increasing release dates r_j. LIS was introduced in Subsection 5.3.2, and we showed that LIS is 5-competitive for C-Wgp. Theorem 6.2 implies that it is rather unlikely that there exists a polynomial time algorithm for F-Wgp, with a constant competitive ratio. We prove that LIS is $O(m)$-competitive. Then we analyze LIS using speed resource augmentation; we prove that LIS is a σ-speed optimal algorithm, for any $\sigma \geq 5$.

Theorem 6.4. LIS *is $O(m)$-competitive for* F-Wgp.

Proof. Since every packet must be gathered at the sink, clearly $\max_j F_j^* \geq \max_j \delta_j \geq \max_j \pi_j$. Now let j be the packet incurring the maximum flow time in the schedule obtained by LIS. Since $r_j \geq r_{b(j)}$ (by definition of LIS), we have

$$R_{b(j)} - r_j = r_{b(j)} + \delta_{b(j)} - \pi_{b(j)} - r_j \leq \delta_{b(j)} \tag{6.1}$$

Using Lemma 5.4 and (6.1), we get

$$\begin{aligned} F_j = C_j - r_j &\leq R_{b(j)} - r_j + \frac{\gamma}{\gamma^*} \sum_{i \in P(j)} \min\{\delta_j, \gamma\} \\ &\leq \delta_{b(j)} + \frac{\gamma}{\gamma^*} \sum_{i \in P(j)} \pi_i \\ &\leq \max_i F_i^* + \frac{\gamma}{\gamma^*} \cdot |P(j)| \cdot \max_i F_i^* \\ &\leq \left(1 + \frac{\gamma}{\gamma^*} m\right) \max_i F_i^*. \end{aligned}$$

□

Hence, it follows from the theorem above and Theorem 6.3 that LIS is best possible among the class of shortest paths following algorithms. Next, we consider the σ-speed augmented version of LIS.

Algorithm 11 σ-LIS

1. Create a new instance $\mathcal{I}'$ by multiplying release dates: $r'_j := \sigma r_j$;
2. Run LIS on $\mathcal{I}'$;
3. Speed up the schedule thus obtained by a factor of σ.

Note that the algorithm can be implemented online as a LIS algorithm using adjusted release dates r'_j.

The schedule constructed by σ-LIS is a feasible σ-speed solution to the original problem because of step 1. We will show that σ-LIS is optimal for both C-Wgp and F-Wgp, if $\sigma \geq \gamma/\gamma^* + 1$. The following lemma is crucial.

Lemma 6.5. *If* σ-LIS *is a* σ*-speed optimal algorithm for* C-Wgp*, then it is also a* σ*-speed optimal algorithm for* F-Wgp*.*

Proof. Let F_j^* and $F_{j,\sigma}$ be the flow time of data packet j in an optimal solution and in a σ-LIS solution, respectively, to F-Wgp and let C_j^* and $C_{j,\sigma}$ be the completion time of data packet j in the same solutions. Suppose σ-LIS is a σ-speed optimal algorithm for C-Wgp, hence we have $\max_{j\in M} C_{j,\sigma} \leq \max_{j\in M} C_j^*$. We show that this inequality implies, for any time t,

$$\max_{j\in M,\, r_j=t} C_{j,\sigma} \leq \max_{j\in M,\, r_j\leq t} C_j^*. \tag{6.2}$$

We prove inequality (6.2) by contradiction. Suppose it is false, then there is an instance $\mathcal{I}$ of minimum size (number of data packets) for which it is false. Also, let t_0 be the first round in such an instance for which it is false. By definition, σ-LIS schedules each data packet j definitively in round r_j; no data packet is rescheduled in a later round. I.e., the algorithm determines the completion time $C_{j,\sigma}$ of data packet j in round r_j. If the inequality is false, then we must have

$$C_{i,\sigma} > \max_{j\in M,\, r_j\leq t_0} C_j^*, \tag{6.3}$$

for some data packet i with $r_i = t_0$, and because $\mathcal{I}$ is a minimum size instance the instance does not contain any data packets released after round t_0. But then (6.3) contradicts $\max_{j\in M} C_{j,\sigma} \leq \max_{j\in M} C_j^*$. Using (6.2) we have

$$\begin{aligned}\max_{j\in M} F_{j,\sigma} &= \max_t \left(\max_{j\in M,\, r_j=t} C_{j,\sigma} - t \right) \leq \max_t \left(\max_{j\in M,\, r_j\leq t} C_j^* - t \right) \\ &\leq \max_t \left(\max_{j\in M,\, r_j\leq t} F_j^* \right) = \max_{j\in M} F_j^*.\end{aligned}$$

□

Theorem 6.6. *For $\sigma \geq \gamma/\gamma^* + 1$, σ-*LIS *is a σ-speed optimal algorithm for both* C-WGP *and* F-WGP*.*

Proof. By Lemma 6.5, it suffices to prove that σ-LIS is σ-speed optimal for C-WGP.

Let C_j be the completion time of any data packet j in the σ-LIS solution on instance $\mathcal{I}$, and let C'_j be the completion time of j in the LIS solution on the instance $\mathcal{I}'$ (see Step 1 of σ-LIS). By construction $C_j = C'_j/\sigma$. Let $R'_j := \sigma r_j + \delta_j - \pi_j$. Then the upper bound of Lemma 5.4 applied to instance $\mathcal{I}'$ implies $C'_j \leq R'_{b(j)} + (\sigma - 1)\sum_{i \in P(j)} \pi_i$. Hence,

$$C_j = C'_j/\sigma \leq \frac{1}{\sigma} R'_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i \in P(j)} \pi_i \quad \leq \quad r_{b(j)} + \frac{1}{\sigma}\delta_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i \in P(j)} \pi_i \tag{6.4}$$

Since in any solution $b(j)$ has to reach the sink we clearly have

$$\max_{i \in P(j)} C^*_i \geq C^*_{b(j)} \geq r_{b(j)} + \delta_{b(j)}. \tag{6.5}$$

Also, by Lemma 5.5, for some $k \in P(j)$,

$$\max_{i \in P(j)} C^*_i \geq R_k + \sum_{i \in P(j)} \pi_i \geq r_k + \sum_{i \in P(j)} \pi_i \geq r_{b(j)} + \sum_{i \in P(j)} \pi_i, \tag{6.6}$$

where the last inequality follows from $b(j)$ having lowest release time in $P(j)$, by definition of LIS. Combining (6.4), (6.5) and (6.6), we obtain

$$\begin{aligned} \max_{i \in P(j)} C^*_i &= \frac{1}{\sigma} \max_{i \in P(j)} C^*_i + \frac{\sigma - 1}{\sigma} \max_{i \in P(j)} C^*_i \\ &\geq \frac{1}{\sigma}\left(r_{b(j)} + \delta_{b(j)}\right) + \frac{\sigma - 1}{\sigma}\left(r_{b(j)} + \sum_{i \in P(j)} \pi_i\right) \\ &= r_{b(j)} + \frac{1}{\sigma}\delta_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i \in P(j)} \pi_i \geq C_j. \end{aligned}$$

□

Corollary 6.7. 5-LIS *is a 5-speed optimal algorithm for* C-WGP *and* F-WGP*.*

Proof. Directly from Lemma 5.3 and Theorem 6.6. □

6.4 Distributed algorithms

In this section we consider distributed algorithms for WGP, minimizing flow times. We introduce a class of distributed algorithms, and we give a lower bound on the approximability of F-WGP for this class of algorithms. Next, we analyze the average flow time of a distributed algorithm from this class. For ease of presentation, we

assume throughout this section that $d_T = d_I = 1$. At the end of the section we outline how our results can be extended to the case with arbitrary d_T and d_I.

Our main interest is to study a class of simple distributed algorithms. We consider a class of distributed algorithms which only use distance to the sink, a random number generator, and a synchronous clock, to determine whether or not to send a packet. If a node decides to send a packet, it can send any packet present at that node at that time. We call this the class of *simple distributed algorithms*. This class of algorithms has been studied in the context of wireless gathering by Bar-Yehuda et al. [8]. We use the distance of a node to the sink to assign nodes to different time slots; this decreases the possibility of interference. Note that even in this case not all interference can be avoided a priori. A coloring of nodes at the same distance from the sink may further reduce the possibility of interference; however, such an approach typically requires some local communication between nodes to choose the colors [55], resulting in decentralized algorithms, rather than distributed algorithms. We only consider simple distributed algorithms in this section.

6.4.1 Lower bounds

We are interested to derive a lower bound on the approximability of the class of simple distributed algorithms. In case a node receives signals from multiple nodes in a round, no packet can be received by this node. We describe two relaxation models of this notion of interference. For both interference relaxation models, we assume a receiver node only receives signals from sender nodes which try to send a packet to that node, and in this case one packet may be received during this round.

Relaxation interference models

- The *random selection model*: in each round, in case the schedule contains interfering calls, randomly remove interfering calls until the schedule is interference free;

- The *adversarial selection model*: in each round, in case the schedule contains interfering calls, an adversary removes interfering calls until the schedule is interference free.

The relaxation interference models replace the interference constraints with a restriction on the choice of the packet that advances. The random selection model seems a reasonable relaxation model for simple distributed algorithms, given the fact that such an algorithm may only use a random number generator to choose when to send, and it has no information on the packets at other nodes.

In Section 6.2 we showed that F-Wgp cannot be approximated within a factor of $O(m^{(1-\epsilon)/2})$, unless $\mathbf{P} = \mathbf{NP}$. Here, we give an *unconditional* lower bound on the approximability of F-Wgp for simple distributed algorithms.

Theorem 6.8. *In the random selection model, the approximation ratio of any algorithm is at least* $\Omega(\log m)$ *for* F-WGP.

Proof. Consider a constant $k \geq 1$. An adversary releases $m = 2h^2$ packets, for h some integer power of 2. In round $i \cdot k \cdot h$ it releases 2 packets, and in each round in interval $[ikh + 1, ikh + h - 2]$ it releases one packet, for $i = 1, \ldots, 2h$. The only use of k in this setting is provision of a control over the average arrival rate of packets.

We use a star network with three rays, with the sink as the center of the star. Pairs of packets released in the same round are released on two different rays of the star, say ray 1 and ray 2; all the other packets are released on ray 3.

Both the algorithm and the adversary can send only one of the packets released at time ikh in that round, and have to send the other packet in a later round. We call the packet which is released but not sent in round ikh the *target packet*. We prove the theorem by demonstrating that the expected flow time of one of the $2h$ target packets exceeds $\log h + 1$ for any algorithm, whereas the maximum flow time of the adversary is at most 2.

The adversary sends one of the packets, released at time ikh, in that same round, and the target packet in the next round. The adversary can send all other packets one round after their release dates, hence the maximum flow time of the adversary is at most 2.

Consider any algorithm in the random selection model. Clearly the algorithm can forward at most one packet to the sink each round. Because of the random selection rule, for $f \geq 1$, we have

$$\mathbf{Pr}[F_j = f + 1 \,|\, j \text{ is a target packet}] = 1/2^f.$$

So, if p is the probability that the flow time of a target packet is at most $\log h + 1$, we have $p = \sum_{f=1}^{\log h} (1/2)^f = 1 - 1/h$. Consider the set of $2h$ target packets. For $\ell = 1, \ldots, 2h$, let $X_\ell = 1$ be the event that target packet ℓ has flow time exceeding $\log h + 1$. Otherwise $X_\ell = 0$. Then $\mathbb{E}[X_\ell] = 1/h$ for all $\ell = 1, \ldots, 2h$ and therefore the expected number of target packets with flow time exceeding $\log h + 1$ is $\mathbb{E}[\sum_{\ell=1}^{2h} X_\ell] = \sum_{\ell=1}^{2h} \mathbb{E}[X_\ell] = 2$. Hence, in expectation at least one target packet has flow time at least $\log h + 1$. □

The result of Theorem 6.8 is unconditional, i.e. it does not depend on the assumption $\mathbf{P} \neq \mathbf{NP}$, as in Theorem 6.2. In Section 6.3 we showed that allowing a constant increase in speed one can obtain a solution with maximum flow time which is less than that of the optimal solution to the original instance. The proof of Theorem 6.8 indicates that an increase in speed of $\Omega(\log m)$ is required to offset the adversarial selection. The reason is that in the random selection model distributed algorithms have no control on which packet to advance; and the probability of obtaining such an adversarial selection depends on the number of packets, and not on the speed of the algorithm.

In the next section we will analyze a simple distributed algorithm. It cannot determine which packet is advanced from each layer to the next. Theorem 6.8 and the observations above should thus explain why we focus on the analysis of the

expected average flow time, as opposed to maximum flow time, and why we need to use resource augmentation. Finally, we remark the intuitive fact that, if conflicts are solved adversarially as opposed to randomly, Theorem 6.8 can be strengthened further.

Theorem 6.9. *In the adversarial selection model, the approximation ratio of any deterministic algorithm is at least* $\Omega(m)$ *for* F-WGP.

Proof. Consider the instance in the proof of Theorem 6.8. The maximum flow time of the adversary is two. In the adversarial selection model, no deterministic algorithm can send the target packet to the sink before sending the $h-2$ packets released at ray 3. Hence, each of the $2h$ target packets incurs a flow time of $h = \sqrt{m/2}$; this yields a lower bound of $\Omega(\sqrt{m})$. We can improve this bound to $\Omega(m)$ on an instance with m packets, if we consider a sub-instance which only contains h packets released in $[ikh, ikh + h - 2]$, for some i. □

6.4.2 A synchronous distributed algorithm

We consider a distributed algorithm for WGP first introduced by Bar-Yehuda et al. in the context of minimizing the maximum completion time for the gathering problem without release dates [8]. We focus on flow times.

To reduce interference between nodes, the algorithm partitions nodes into layers, and assigns a label to nodes in a layer. A *layer* is as a set of all nodes at a distance d from the sink. A node at distance d from the sink is assigned *label* d mod 3. Each node can be either *active* during a round or *inactive*; only active nodes will transmit a packet. A node will not be active if its packet buffer is empty.

Before we describe the algorithm, we give a formal description of the decay procedure described in Subsection 2.2.2. The procedure, first introduced and studied by Bar-Yehuda, Goldreich and Itai [7], is called DECAY and requires $2\log\Delta$ rounds; the time needed for a single execution of the procedure is called a *phase*. DECAY enables communication from a set of active nodes.

Algorithm 12 DECAY(u, v)

for $j = 1, 2, \ldots, 2\log\Delta$ **do**
  u sends to v the oldest packet from its buffer;
  u deactivates itself for the rest of the phase with probability 1/2.
end for

We can now state a distributed algorithm for WGP.

Algorithm 13 DISTRIBUTEDGREEDY (DG) [8]

for each next phase $k = 1, 2, \ldots$ **do**
  Activate each node with label k mod 3 having a nonempty packet buffer;
  Execute DECAY$(u, \text{parent}(u))$ in parallel for each active node u.
end for

We assume the existence of a communication tree T for forwarding packets. Although the algorithm does not model acknowledgement of packets explicitly, it is easy to include them, e.g. by doubling the number of rounds, having communication in odd rounds and acknowledgements in even rounds, as in [8]. Using this, we can assume that successful receipt of a packet (by the parent of the sending node in the communication tree) is acknowledged immediately. Only at that time it gets removed from the sender's buffer.

By the transmission protocol in DG, where in phase k only nodes of layer k mod 3 transmit, if two nodes transmit, then either they are at the same layer or they are at least distance 3 apart. Hence, in DG two nodes can only interfere if both sender nodes are in the same layer.

A super-phase consists of three consecutive phases. Another important ingredient in the analysis of DG is the following.

Theorem 6.10 ([8]). *Let i be a layer of the tree containing some packet at the beginning of a super-phase. There is probability at least $\mu := e^{-1}(1-e^{-1})$ that during this super-phase* DG *sends a packet from a node u in layer i successfully to the parent node of u in the communication tree.*

This theorem shows that, during a super-phase, each nonempty layer forwards a packet with probability μ to the following layer. Notice however that there is no guarantee on which particular packet is advanced. The use of super-phases and labels, i.e. a synchronous model, is essential to the proof of Theorem 6.10. In case the DECAY procedure would be applied in an asynchronous model it is not clear whether a similar constant probability μ is a attainable.

For our analysis we define three algorithm classes. For a given instance $\mathcal{I}$ of WGP, we construct relations between the completion times of packets in these three classes. This approach is similar to the approach of [7].

- In Class 1 each layer sends with probability at least μ at least one packet every super-phase;
- In Class 2 each layer sends with probability $p := 1-(1-\mu)^a$ one packet in one of the three last rounds of every a super-phases, $a \in \mathbb{N}$, and with probability $(1-\mu)^a$ no packet;
- In Class 3 each layer sends with probability p one packet in each round, and with probability $1-p$ no packet.

Note that the probabilities stated above refer to the case in which the layer contains a packet at the start of the period considered (a single super-phase in Class 1, a super-phases in Class 2, and a single round in Class 3). This restriction is similar to the restriction on the DECAY procedure. Thus, it follows from Theorem 6.10 that DG is an algorithm which fits into Class 1. We define the completion time and the flow time of a packet j in a Class k-schedule, respectively, as $C_j^{(k)}$ and $F_j^{(k)}$, $k = 1, 2, 3$.

Motivated by the negative results of Theorem 6.8 we focus on deriving a bound on the expected average flow time of DG. We use resource augmentation to analyze the performance of DG. A σ-speed algorithm sends data packets at a speed that is σ times faster than an offline algorithm. In particular a σ-speed DG schedule is a σ-speed Class 1-schedule, i.e. each layer sends with probability at least μ at least one packet every t rounds, where t is $1/\sigma$ times the number of rounds in a super-phase.

Through a sequence of steps we relate the expected flow times of Class 1 to those of Class 2. Subsequently, we demonstrate that the expected flow time of σ-speed DG is bounded by the expected flow time of a Class 3-schedule. Finally, we relate the expected flow times of a Class 3-schedule to the expected flow times of an optimal offline schedule.

Our analysis extends proof techniques of Bar-Yehuda et al. [7] to derive a bound on the sum of completion times for instances where packets may have release dates. For the proofs we have to introduce some notation. We define a distribution vector d, a move vector m and an arrival vector a of dimension $\delta + 1$. We use these three types of vectors to characterize schedules in a specific round. For a given round, d_i is the number of packets in layer i, m_i is the number of packets which are sent from layer i to layer $i-1$, and a_i is the number of packets which arrive at a node in layer i. We assume there exists a layer 0 which contains all packets which have been sent to the sink; by definition $a_0 = 0, m_0 = 0$ for each round. Note that we must have $m_i \leq d_i + a_i$ for each layer i and each round.

The initial distribution vector and all arrival vectors are input data of the instance. The move vectors can be derived from the algorithm. Given vectors d^t, m^t and a^t for round t the distribution vector of round $t+1$ can be calculated as follows:

$$\begin{aligned} d_i^{t+1} &= d_i^t + a_i^t - m_{i+1}^t - m_i^t, \quad i = 0, \ldots, \delta - 1, \\ d_\delta^{t+1} &= d_\delta^t + a_\delta^t - m_\delta^t. \end{aligned}$$

We denote this operation as $d^{t+1} = Move(d^t, m^t, a^t)$.

Given vectors d, m and a for a round, we may define an *abstract move vector* $\bar{m}$ as follows. For each layer $i \in 0, \ldots, \delta$:

$$\begin{aligned} \bar{m}_i &= m_i \text{ if } d_i + a_i > m_i; \\ \bar{m}_i &\in [d_i + a_i, \infty) \text{ if } d_i + a_i = m_i. \end{aligned}$$

Similarly, given vectors $d, \bar{m}$ and a we can calculate m. Due to this many-to-one relation we can define $Move(d, \bar{m}, a) := Move(d, m, a)$.

We define an ordering $\preceq$ on n-dimensional vectors. We say $a \preceq a'$ if there exists a $k \in \{1, \ldots, n\}$ such that either $a_k < a'_k$ and $a_j \leq a'_j$ for $j = 1, \ldots, k-1$ or $a_j \leq a'_j$ for $j = 1, \ldots, n$. Next, we cite a useful lemma, which holds for any (abstract) move vector m.

Lemma 6.11. *[7] If $m' \preceq m$ and $d \preceq d'$ then $Move(d, m, a) \preceq Move(d', m', a)$.*

This lemma suffices to relate Class 1 and Class 2.

Lemma 6.12. *The expected sum of completion times of a Class 1-schedule is at most the expected sum of completion times of some Class 2-schedule, for every* WGP*-instance.*

Proof. Suppose we are given an instance of WGP, with arrival vectors $a^0, \ldots, a^{T-1}$, and a Class 1-schedule. This solution defines distribution vectors $d^1, \ldots, d^T$ and move vectors $m^1, \ldots, m^{T-1}$, where T is the completion time of the solution. We define abstract move vectors $\bar{m}^1, \ldots, \bar{m}^{T-1}$ as follows. For each round $t \in 1, \ldots, T-1$, and layer $i \in 0, \ldots, \delta$:

$$\begin{aligned} \bar{m}_i^t &:= m_i^t \text{ if } d_i^t + a_i^t > 0; \\ \bar{m}_i^t &:= 1 \text{ if } d_i^t + a_i^t = 0. \end{aligned}$$

It is straightforward to see that $\bar{m}^t$ is indeed an abstract move vector with respect to m^t. Next, we define move vector $\tilde{m}$ as follows. Let $p := P(\bar{m}_i^t \geq 1)$; by definition of Class 1 we have $p \geq \mu/K$.

$$\begin{aligned} \tilde{m}_i^t &:= 1 \text{ with probability } \mu/Kp \text{ if } \bar{m}_i^t \geq 1; \\ \tilde{m}_i^t &:= 0 \text{ with probability } 1 - \mu/Kp \text{ if } \bar{m}_i^t \geq 1; \\ \tilde{m}_i^t &:= 0 \text{ if } \bar{m}_i^t = 0. \end{aligned}$$

As a consequence $P(\tilde{m}_i^t = 0) = P(\bar{m}_i^t = 0) + (1 - \mu/Kp) \cdot P(\bar{m}_i^t \geq 1) = (1 - P(\bar{m}_i^t \geq 1)) + (1 - \mu/Kp) \cdot P(\bar{m}_i^t \geq 1) = (1 - p) + (p - \mu/K) = 1 - \mu/K$. Also, $\tilde{m}^t \leq \bar{m}^t$. Hence, by construction $\tilde{m}$ is a realization on the same instance of a schedule where each layer sends with probability exactly μ a packet from each layer in each super-phase, and with probability $1 - \mu$ no packet. We call this a Class 2'-schedule. Also, from move vector $\tilde{m}$ we can identify distribution vectors $\tilde{d}$.

Now, we prove the lemma by analyzing the distribution vectors d and $\tilde{d}$. We prove by induction that $d^t \preceq \tilde{d}^t$ for each round t. Initially we have $d^0 = \tilde{d}^0$, i.e. $d^0 \preceq \tilde{d}^0$. Suppose the claim holds for round t. Then we have

$$d^{t+1} = Move(d^t, m^t, a^t) = Move(d^t, \bar{m}^t, a^t) \preceq Move(\tilde{d}^t, \tilde{m}^t, a^t) = \tilde{d}^{t+1},$$

where the second equality follows from definition of the abstract vector, and the ordering follows from Lemma 6.11, and the orderings $d^t \preceq \tilde{d}^t$, and $\tilde{m}^t \preceq \bar{m}^t$.

Because $d^t \preceq \tilde{d}^t$ for each round t, the completion time of the i-th packet to arrive at the sink in the Class 1-schedule is at most the completion time of the i-th packet to arrive at the sink in the Class 2'-schedule.

Consider this Class 2'-schedule. For each layer, the probability that after aK super-phases no packet is sent is $(1 - \mu)^a$. Hence, with probability $1 - (1 - \mu)^a$ at least one packet has been sent from each layer after each aK super-phases. Using the same argument as above to relate Class 1 and Class 2' we can prove that the sum of completion times of a Class 2'-schedule is at most the sum of completion times of a Class 3-schedule. The lemma follows by taking expectations. □

Lemma 6.13. *The expected sum of flow times of σ-speed* DG *is at most the expected sum of flow times of a Class 3-schedule, for $\sigma \geq 6a \log \Delta$.*

Proof. Consider instance $\mathcal{I}$. Let F_j (C_j) be the flow time (completion time) of packet j in a DG schedule, and let $F_{j,\sigma}$ ($C_{j,\sigma}$) be the flow time (completion time) of packet j in a σ-speed DG schedule of $\mathcal{I}$.

It follows from Lemma 6.12 and the fact that DG is a Class 1-algorithm that $\mathbb{E}[\sum_j C_j] \leq \mathbb{E}[\sum_j C_j^{(2)}]$. This implies $\mathbb{E}[\sum_j F_j] = \mathbb{E}[\sum_j C_j] - \sum_j r_j \leq \mathbb{E}[\sum_j C_j^{(2)}] - \sum_j r_j = \mathbb{E}[\sum_j F_j^{(2)}]$.

Consider a Class 2-schedule. If we speed up this schedule with factor $6a \log \Delta$, then each layer sends with probability p a packet every t rounds, where t is $a/(6a \log \Delta)$ times the number of rounds in a super-phase, i.e. in each single round. Hence, such a schedule is equivalent to a Class 3-schedule, i.e. the sum of flow times of this schedule is at most the sum of flow times of a Class 3-schedule. Applying a higher speedup factor to the Class 2-schedule can only improve the flow times of the Class 2-schedule. Hence, for $\sigma \geq 6a \log \Delta$, $\mathbb{E}[\sum_j F_{j,\sigma}] = \mathbb{E}[\sum_j F_j/\sigma] \leq \mathbb{E}[\sum_j F_j^{(2)}/\sigma] \leq \mathbb{E}[\sum_j F_j^{(2)}/(6a \log \Delta)] = \mathbb{E}[\sum_j F_j^{(3)}]$. □

A deterministic tandem queue with unit processing times is a network which consists of a sink M_0, and a set of machines $M_i, i = 1, \ldots, \delta$, which are positioned in sequence. I.e. a job which has been processed on machine M_i is sent to machine $M_{i-1}, i = 1, \ldots, \delta$. The processing time of a job is 1 on each machine. Jobs arrive on some machine. We relate the expected flow time of a Class 3-schedule to the flow time of a tandem queue, in which the layers are the machines.

Lemma 6.14. *The expected sum of flow times of a Class 3-schedule is at most* $1/p^\delta$ *times the sum of flow times of a schedule of a deterministic tandem queue with unit processing times.*

Proof. Consider a Class 3-schedule S for instance $\mathcal{I}$. First, assume that when in a round some layer fails to communicate, then all layers fail in that round. In this case, the probability of having no communication in a round is $1 - p^\delta$. The schedule S' that remains after removing rounds without communication is equivalent to a schedule of a deterministic tandem queue with unit processing times, where each packet j of the Class 3-schedule is equivalent to a job arriving at machine $M_{d(v_j,s)}$. Consider any set of k rounds of a schedule of a deterministic tandem queue with unit processing times. The expected number of rounds required to schedule these k rounds in schedule S' is $kp^\delta \cdot \sum_{i=0}^{\infty} (1-p^\delta)^i \cdot (i+1) = k/p^\delta$. Let F'_j be the flow time of packet j in S'. Then we have $\mathbb{E}[F_j^{(3)}] = F'_j/p^\delta$, for each packet j. Flow times are not increased if only some layers fail, but others forward a packet. □

Lemma 6.15. *The sum of flow times of a deterministic tandem queue with unit processing times is at most the sum of flow times of an optimal off line* WGP *schedule.*

Proof. Let S be a deterministic tandem queue schedule, and let S^* be a schedule where in each round each layer, except layer 1, can forward any number of packets, and layer 1 can forward at most one packet.

Consider schedule S^*. Let M_t^* be the set of packets which have not arrived at the sink in round t. Because the schedule can forward any number of packets over an edge, we have that if no packet is sent to the sink in S^* in round t, then no packet in M_t^* can arrive at the sink before or at round t, i.e. $r_j + \delta_j > t$ for each $j \in M_t^*$. We prove the lemma by demonstrating that if some packet is sent to the sink in S^* in round t, then also some packet is sent to the sink in S in round t. This suffices to prove the lemma, because an offline optimal WGP schedule can not send packets faster than in schedule S^*.

Suppose to the contrary that there is a first round t in which some packet is sent to the sink in S^*, but no packet is sent to the sink in S. Let $t', t' < t$, be the last round before t' in which no packet is sent to the sink in S^*. Then, there is a set of $t - t'$ packets in S^* which arrive at the sink in rounds $(t', t]$. Hence, it follows from this and the observation above that there are $t - t'$ packets j such that $t' < r_j + \delta_j \leq t$. Now consider schedule S; in this schedule $t - t' - 1$ packets are sent to the sink in rounds $(t', t-1]$, hence there is a packet j with $t' < r_j + \delta_j \leq t$ which has not arrived at the sink in round t. But then, j must have been in layer $1 + i$ or higher in rounds $t - i$, $i = 1, \ldots, \delta_j$. I.e. j must have been in layer $1 + \delta_j$ or higher in round $r_j \leq t - \delta_j$ which gives a contradiction. □

Theorem 6.16. *Let $\epsilon > 0$ and $\sigma = 6\mu^{-1} \cdot \log \Delta \cdot \ln(\delta/\epsilon)$. Then σ-speed* DG *is in expectation e^ϵ-competitive for the objective of minimizing the average flow time.*

Proof. It follows from Lemmas 6.12, 6.13, 6.14 and 6.15 that the expected sum of flow times of σ-speed DG is at most $1/p^\delta$ times the sum of flow times of an optimal offline schedule, for $\sigma = 6a \log \Delta$. As σ-speed DG is an online algorithm, DG is σ-speed $p^{-\delta}$-competitive when minimizing average flow times. The probability $p = 1 - (1 - \mu)^a$ depends on the choice of the speedup a. We set $a := \mu^{-1} \ln(\delta/\epsilon)$, which gives

$$\begin{aligned} p &= 1 - (1-\mu)^{\mu^{-1} \ln(\delta/\epsilon)} \\ &\geq 1 - e^{-\ln(\delta/\epsilon)} = 1 - \frac{\epsilon}{\delta} \end{aligned}$$

so that

$$p^{-\delta} \leq \left(1 - \frac{\epsilon}{\delta}\right)^{-\delta} \leq e^\epsilon.$$

□

It follows from the theorem that the competitive ratio of DG can be made arbitrarily close to 1 with an appropriate increase in speed. I.e. DG yields in expectation an average flow time close to the optimal offline solution in case it can send packets at a higher speed.

Algorithm DG can be extended to the case of arbitrary d_T and d_I. This can be seen as follows. We assign to a node in layer d the label $d \bmod d_I + d_T + 1$ and we use super-phases consisting of $d_I + d_T + 1$ phases each. In this way one can avoid interference between nodes from different layers of the tree. It is easy to see that Theorem 6.10 can be extended to this setting.

Using this fact, we can extend our analysis to prove the following result.

Theorem 6.17. *Let $\epsilon > 0$ and $\sigma = \Theta(d_I^2 \cdot \log \Delta \cdot \ln(\delta/\epsilon))$. Then σ-speed* DG *is in expectation e^ϵ-competitive for the objective of minimizing the average flow time.*

We notice that one d_I factor is due to the longer super-phases, and another one is due to DECAY having to cope with larger neighborhoods (of size Δ^{d_I}).

6.5 Conclusion and open problems

We considered the wireless gathering problem under interference of radio signals, with the objective of minimizing the flow time of data packets, and we denote this problem F-WGP when minimizing the maximum flow time. We showed that no polynomial time algorithm can approximate F-WGP within $O(m^{\frac{1-\epsilon}{2}})$, unless $\mathbf{P} = \mathbf{NP}$. Also no shortest paths following algorithm can approximate F-WGP within $O(m)$. Next, we analyzed LIS, a greedy shortest paths following algorithm which we introduced in Chapter 5. We showed that LIS is $O(m)$-competitive, and a resource augmented version of LIS is optimal for an augmented speed of factor 5 or more. Finally, we considered distributed algorithms. We presented lower bounds for a general class of distributed algorithms, called simple distributed algorithms. We showed that no simple distributed algorithm can approximate F-WGP within $O(\log m)$, and simple distributed algorithms which use an extra speed of factor less than $O(\log m)$ do not have a constant competitive ratio. We analyzed a simple distributed algorithm, and proved that it is near optimal in case the algorithm is allowed to send packets a factor $O(\log \Delta \log \delta)$ faster than an offline algorithm, in case $d_T = d_I = 1$. This result can be extended to the case $d_I \geq d_T$ at the expense of a multiplicative extra factor $O(d_I^2)$. The main results are summarized in Table 6.1.

Objective	**Model**	**Ratio**	**Augmentation**	**Lower bound**	**Condition**
$\max F_j$	*centralized*	1	$\sigma \geq 5$	$O(m^{\frac{1-\epsilon}{2}})$	$\mathbf{NP} \neq \mathbf{P}$
$\max F_j$	*distributed, simple*	*open*	–	$\Omega(\log m)$	–
$\sum F_j$	*distributed, simple*	$\mathbb{E}[e^\epsilon]$	$\sigma = O(d_I^2 \log \Delta \ln(\delta/\epsilon))$	*open*	–

Table 6.1: Competitive ratio results on WGP minimizing flow times.

There remain some interesting open problems. For the centralized online problem we do not know whether optimality can be achieved by augmenting the communication rate by a factor smaller than 5, and whether an efficient algorithm exists that matches the lower bound on the approximability of F-WGP. Another interesting set of questions concerns resource augmentation by allowing the algorithms to use extra frequencies, meaning that more than one data packet can be sent simultaneously over the same channel. For instance, it is an open problem whether there exists a 5-frequency optimal algorithm using LIS. For the distributed problem we presented a simple algorithm, under a synchronous clock model. The algorithm uses extra

resources of $O(\log \Delta \log \delta)$; it is interesting to obtain an algorithm with constant competitive ratio for a smaller amount of extra resources. Also, the synchronous model is essential for the distributed algorithm that we analyzed; it is interesting to find efficient simple distributed algorithms under the asynchronous time model.

Bibliography

[1] N. Abramson. The ALOHA system - another alternative for computer communications. In *Proceedings of the Fall 1970 AFIPS Computer Conference*, pages 281–285, 1970.

[2] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM*, 32(4):871–895, 1985.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks Journal*, 38(4):393–422, 2002.

[4] S. Albers and H. Bals. Dynamic TCP acknowledgment: Penalizing long delays. *SIAM Journal on Discrete Mathematics*, 19(4):938–951, 2005.

[5] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[6] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[7] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: an exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

[8] R. Bar-Yehuda, A. Israeli, and A. Itai. Multiple communication in multihop radio networks. *SIAM Journal on Computing*, 22(4):875–887, 1993.

[9] L. Becchetti, P. Korteweg, A. Marchetti-Spaccamela, M. Skutella, L. Stougie, and A. Vitaletti. Latency constrained aggregation in sensor networks. SPOR-report 2006-08, TU Eindhoven, www.win.tue.nl/bs/spor, 2006.

[10] L. Becchetti, P. Korteweg, A. Marchetti-Spaccamela, M. Skutella, L. Stougie, and A. Vitaletti. Latency constrained aggregation in sensor networks. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Computer Science*, pages 88–99, 2006.

[11] J.-C. Bermond, R. C. Corrêa, and M.-L. Yu. Gathering algorithms on paths under interference constraints. In *Proceedings of the 6th Italian Conference Algorithms and Complexity (CIAC)*, volume 3998 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2006.

[12] J.-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Gathering in specific radio networks. In *8èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel06), Trégastel*, 2006.

[13] J.-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of gathering in static radio networks. *Parallel Processing Letters*, 16(2):165–183, 2006.

[14] P. L. Bernstein. *Against the gods: The remarkable story of risk*. John Wiley & Sons, Inc., 1996.

[15] R. E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22, 1994.

[16] R. E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[17] V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela, and L. Stougie. The distributed wireless gathering problem. Accepted for the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM), 2008.

[18] V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela, and L. Stougie. An approximation algorithm for the wireless gathering problem. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 328–338, 2006.

[19] V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela, and L. Stougie. Minimzing flow time in the wireless gathering problem. In *Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.

[20] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[21] A. Boukerche, editor. *Handbook of Algorithms for Wireless Networking and Mobile Computing*. Chapman & Hall/CRC, 2005.

[22] C. Brito, E. Koutsoupias, and S. Vaya. Competitive analysis of organization networks or multicast acknowledgement: how much to wait? In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–635, 2004.

[23] CBS. Mobiele telefonie volledig ingeburgerd. www.cbs.nl, 2007.

[24] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, GSIA, Carnegie-Mellon University, Pittsburgh, 1976.

[25] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg. Greedy packet scheduling. *SIAM Journal on Computing*, 24(1):148–157, 1995.

[26] K. G. Coffman and A. M. Odlyzko. Internet growth: Is there a "Moore's Law" for data traffic? In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 47–93, 2002.

[27] S. Coleri Ergen and P. Varaiya. Energy efficient routing with delay guarantee for sensor networks. *Wireless Networks*, 13(5):679–690, 2007.

[28] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001.

[29] M. Ehrgott and X. Gandibleux. A survey and annoted bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(3):425–460, 2000.

[30] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, pages 1965–1970, 2001.

[31] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th ACM symposium on Operating Systems Design and Implementation (OSDI)*, pages 147–163, 2002.

[32] J. Elson and K. Römer. Wireless sensor networks: a new regime for time synchronization. *Computer Communication Review*, 33(1):149–154, 2003.

[33] G. Even. Personal communication, 2007.

[34] C. Florens, M. Franceschetti, and R. J. McEliece. Lower bounds on data collection time in sensory networks. *IEEE Journal on Selected Areas in Communications*, 22:1110– 1120, 2004.

[35] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53:79–133, 1994.

[36] D. Ganesan, A. Cerpa, W. Ye, Y. Yu, J. Zhao, and D. Estrin. Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):799–814, 2004.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[38] L. Gargano. "Time Optimal Gathering in Sensor Networks", invited lecture at the 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO), 2007.

[39] L. Gargano and A. A. Rescigno. Optimally fast data gathering in sensor networks. In *Proceedings of the 31st Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 399–411, 2006.

[40] A. Goel and D. Estrin. Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk. In *Proceedings of the 14th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 499–505, 2003.

[41] J. Håstad, F. T. Leighton, and B. Rogoff. Analysis of backoff protocols for multiple access channels. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 241–253, 1987.

[42] S. M. Hedetniemi, T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[43] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy efficient communication protocols for wireless microsensor networks. In *Proceedings of the 33rd Hawaiian International Conference on Systems Sciences (HICCS)*, pages 3005–3014, 2000.

[44] D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[45] J. Hromkovič. *Algorithmics for Hard Problems — Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer-Verlag, 2001.

[46] J. Hromkovič, R. Klasing, A. Pelc, P. Ruzicka, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance*. Springer-Verlag, 2005.

[47] M. Ilyas and I. Mahgoub. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC Press, 2004.

[48] ITU. Cellular mobile telephone subscribers (itu,syb50), code 13100. In *Yearbook of Statistics*. International Telecommunications Union. www.itu.int or http://unstats.un.org, 2007.

[49] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[50] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgment and other stories about $e/(e-1)$. *Algorithmica*, 36(3):209–224, 2003.

[51] F. P. Kelly. Stochastic models of computer communication systems. *Journal of the Royal Statistical Society, Series B*, 47(3):279–395, 1985.

[52] S. Khanna, J. Naor, and D. Raz. Control message aggregation in group communication protocols. In *Automata, Languages and Programming, 29th International Colloquium (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 135–146, 2002.

[53] P. Korteweg, A. Marchetti-Spaccamela, L. Stougie, and A. Vitaletti. Data aggregation in sensor networks: Balancing communication and delay costs. In *Proceedings of the 14th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 4474 of *Lecture Notes in Computer Science*, pages 139–150, 2007.

[54] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[55] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. End-to-end packet-scheduling in wireless ad-hoc networks. In J. I. Munro, editor, *Proceedings of the 15th Symposium on Discrete Algorithms (SODA)*, pages 1021–1030, 2004.

[56] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[57] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Travelling Salesman Problem*. Wiley, 1985.

[58] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[59] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14(3):449–465, 1993.

[60] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt-III. Bicriteria network design problems. *Journal of Algorithms*, 28(1):142–171, 1998.

[61] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[62] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[63] K. Pahlavan and A. H. Levesque. *Wireless Information Networks*. John Wiley & Sons, 1995.

[64] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[65] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[66] C. E. Perkins, editor. *Ad Hoc Networking*. Addison-Wesley Professional, 2001.

[67] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[68] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[69] C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors. *Wireless Sensor Networks*. Springer, 2004.

[70] R. Rajagopalan and P. Varshney. Data-aggregation techniques in sensor networks: a survey. *Communications Surveys & Tutorials, IEEE*, 8(4):48–63, 2006.

[71] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt-III. Many birds with one stone: Multi-objective approximation algorithms (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 438–447, 1993.

[72] K. Römer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad hoc Networking & computing (MobiHoc)*, pages 173–182, 2001.

[73] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[74] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[75] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21(3):115–122, 1997.

[76] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[77] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, 2005.

[78] A. Tanenbaum. *Computer Networks*. Pearson Education International, 2003. Fourth Edition.

[79] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[80] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 681–690, 2006.

Summary

Online Gathering Algorithms for Wireless Networks

This thesis addresses optimization problems in wireless communication networks.

An optimization problem describes a situation in which one wants to find an optimal solution among all solutions. Mathematicians study an optimization problem to find a general method that provides an answer to each instance of the problem. We call such a method an algorithm. In particular we are interested in finding an efficient algorithm, an algorithm that provides an answer fast.

We use complexity theory to classify a problem based on the existence of an efficient algorithm for this problem. Complexity theory focuses on a worst-case analysis, which reflects the uncertainty we have on future instances of the problem. There is a class of optimization problems, called **NP**-hard problems, for which mathematicians do not know whether there exists an efficient algorithm, and it is widely believed that such an algorithm does not even exist. Therefore, for this class of problems we are interested in finding efficient approximation algorithms that provide an approximate optimal solution.

Wireless networks have become an important means of communication, and the use of wireless networks is likely to increase in future years. A main optimization problem in these wireless networks is to communicate data over the network to a central node. This is called data gathering. The quality of a solution depends on multiple criteria, which include the energy costs of communication, and the delay in communication.

In this thesis we formulate mathematical models for several gathering problems in wireless networks. The problems we consider are online and distributed by nature; i.e. problem information becomes known over time, and the problem information is distributed over the network. Therefore we focus on online distributed algorithms, which take these restrictions into account. An online approximation algorithm that provides a good approximate solution is also called competitive.

For each of the problems we consider, we analyze the theoretical complexity of the problem, we devise online algorithms, and we analyze the quality of the solutions of these algorithms.

In Chapter 1 we introduce general concepts and definitions related to optimization problems, and their mathematical formulation. Also, we outline the wireless communication problems addressed in this thesis, and our results.

In Chapter 2 we give a background on wireless networks, introducing definitions, concepts and common problems in this field. These problems form the motivation for the optimization problems which we describe and analyze in the following chapters.

Chapters 3 and 4 discuss a gathering problem in a wireless network, with constraints on packet latency, and the possibility of data aggregation. Nodes may delay messages in order to aggregate multiple messages into a single packet, and forward this packet to the sink. This aggregation reduces the communication costs at the expense of an increased message latency. We are interested in finding online algorithms that minimize both the message latencies, and the network communication costs, i.e. the energy use of the nodes.

In Chapter 3 we focus on the problem with arbitrary latency constraints which are modeled as hard constraints. The objective is to minimize the maximum communication costs of a node. Our main results are the following. We prove that the problem is **NP**-hard, and we provide offline and online approximation algorithms. We show that our algorithms are robust, in the sense that they have similar competitive ratios in case we choose as objective to minimize the sum of communication costs, in case we assume the cost function to be concave, or in case we limit the possibilities of aggregation.

In Chapter 4 we consider the same problem, but focus on constant latencies which are modeled as soft constraints. We use bicriteria optimization to find solutions with both low costs and small packet latencies. The main contributions of this chapter are that we present a constant competitive online distributed algorithm for this problem, in case the latency constraints are not too strict, and an analysis of the almost synchronous time model, a new time model.

Chapters 5 and 6 discuss a gathering problem in a wireless network with interference. Interference influences the design of communication algorithms, because it prevents nodes within the same region to communicate simultaneously. We are interested in finding online algorithms that send packets to the sink as fast as possible, explicitly taking into account interference constraints. We use completion times and flow times as performance measures for the solution.

In Chapter 5 we analyze the problem with objective to minimize completion times. We focus on minimizing the maximum completion time. We prove that this problem is **NP**-hard, even in a special case, solving an open problem. We present a class of constant competitive online algorithms.

In Chapter 6 we consider the wireless gathering problem with objective to minimize flow times, the time packets are in the network. We classify the problem with objective minimizing maximum flow times, and with objective minimizing average flow times; our results indicate that it is unlikely to find efficient algorithms for these problems. Then we present an online distributed algorithm which is near-optimal in case the algorithm can send packets at a faster speed than currently available; the speed factor gives an indication of the effect of faster communication devices on the quality of the solution.

Samenvatting

Dit proefschrift behandelt optimaliseringsproblemen in draadloze netwerken.

Een optimaliseringsprobleem beschrijft een situatie waar we geïnteresseerd zijn in het vinden van een optimale oplossing uit een verzameling van oplossingen. Wiskundigen bestuderen een optimaliseringsprobleem om algemene methoden te vinden voor het oplossen van een dergelijk probleem. We noemen zo'n methode een algoritme. Ter illustratie, een bekend optimaliseringsprobleem is het kortste-padprobleem waar we de kortste route van A naar B willen bepalen. Een algoritme stelt ons in staat deze route te bepalen voor willekeurige bestemmingen. In het bijzonder zijn we geïnteresseerd in het vinden van een efficiënt algoritme, een algoritme dat de oplossing snel vindt.

We gebruiken complexiteitstheorie om problemen te classificeren op basis van het feit of er een efficiënt algoritme bestaat voor dit probleem. Complexiteitstheorie is gebaseerd op een worst-case analyse, waarin we er vanuit gaan dat de minst gunstige situatie zich voordoet. De reden voor deze analyse schuilt in de onzekerheid die we hebben over welke situatie van het probleem zich in de toekomst voordoet. Er bestaat een klasse van optimaliseringsproblemen, genaamd **NP**-moeilijke problemen, waarvan wiskundigen niet weten of er een efficiënt algoritme voor bestaat, en er wordt algemeen aangenomen dat een dergelijk algoritme voor deze klasse niet bestaat. Om die reden zijn we bij de klasse van **NP**-moeilijke problemen geïnteresseerd in het vinden van efficiënte algoritmen die een oplossing geven die bij benadering optimaal is. We noemen zo'n algoritme een benaderingsalgoritme.

Draadloze netwerken, zoals mobiele-telefoonnetwerken, vormen een belangrijk communicatiemiddel, en het gebruik van draadloze netwerken zal in de toekomst waarschijnlijk nog toenemen. Een belangrijk optimaliseringsprobleem in draadloze netwerken bestaat uit het communiceren van data naar een centraal punt in het netwerk; we noemen deze problemen verzamelproblemen. De kwaliteit van een oplossing voor dit probleem hangt af van meerdere criteria, zoals de energiekosten van communicatie en de tijd die nodig is om data te verzamelen.

In dit proefschrift formuleren we wiskundige modellen voor verschillende verzamelproblemen in draadloze netwerken. De problemen die we bestuderen zijn online en gedistribueerd; een online probleem is een probleem waar de gebruiker gaandeweg nieuwe informatie krijgt die van belang is voor het oplossen van het probleem; een gedistribueerd probleem is een probleem waar informatie verspreid is over het

netwerk, en een algoritme alleen lokale informatie mag gebruiken voor het maken van een oplossing. Ter onderscheid noemen we een probleem waarbij alle informatie vooraf bekend is een offline probleem. We richten ons op online gedistribueerde algoritmen, die rekening houden met bovengenoemde restricties. Een online benaderingsalgoritme waarvan de oplossingen een goede benadering van de optimale oplossing geven noemen we ook wel een competitief algoritme.

Voor elk probleem dat we bestuderen doen we het volgende: we analyseren de complexiteit van het probleem, we ontwikkelen online algoritmen, en we analyseren de kwaliteit van de oplossingen die deze algoritmen geven.

In hoofdstuk 1 introduceren we concepten en definities die gerelateerd zijn aan optimaliseringsproblemen. Daarnaast geven we een overzicht van de draadloze communicatieproblemen die we in dit proefschrift bestuderen, en een overzicht van onze resultaten.

In hoofdstuk 2 geven we een algemene beschrijving van draadloze netwerken, we introduceren definities, concepten en enkele bekende problemen uit dit onderzoeksgebied. Deze problemen vormen de motivatie voor de optimaliseringsproblemen die we in de volgende hoofdstukken beschrijven en analyseren.

In hoofdstukken 3 en 4 bespreken we een verzamelprobleem waarbij voorwaarden zijn gesteld aan de maximale vertraging van een datapakket. We gebruiken hier de techniek van data-aggregatie. Dit komt erop neer dat de communicatiepunten, knopen genaamd, de mogelijkheid hebben om data te vertragen om zodoende verschillende data te aggregeren in een enkel pakket, en deze vervolgens te versturen. Data-aggregatie vermindert de communicatiekosten, maar levert wel extra vertraging op voor data. We zijn geïnteresseerd in het vinden van online algoritmen die zowel de vertraging als de communicatiekosten minimaliseren.

In hoofdstuk 3 richten we ons op het probleem met willekeurige beperkingen op de maximale vertraging; we modelleren deze beperkingen als harde voorwaarden. Het doel is om de maximale communicatiekosten van een knoop te minimaliseren. Onze belangrijkste resultaten zijn als volgt. We bewijzen dat dit probleem **NP**-moeilijk is, en we geven offline en online benaderingsalgoritmen. We tonen aan dat onze algoritmes robuust zijn, in die zin dat we dezelfde competitieve resultaten vinden als we als doel kiezen het minimaliseren van de totale communicatiekosten, als we kiezen voor een concave kostenfunctie, of als we de mogelijkheden tot aggregatie beperken.

In hoofdstuk 4 beschouwen we hetzelfde probleem, maar hier richten we ons op een constante maximale vertraging voor alle data; we modelleren deze beperking als een zachte voorwaarde, dat wil zeggen dat we er niet aan hoeven te voldoen. We gebruiken concepten uit bicriteria optimalisering voor de analyse van onze algoritmen. Onze belangrijkste resultaten zijn een online gedistribueerd algoritme dat, in het geval de voorwaarde op de maximale vertraging niet te strikt is, constant competitief is. Dit wil zeggen dat zowel de communicatiekosten als de berichtvertragingen niet veel afwijken van de beste offline oplossing. Daarnaast geven we een analyse van de kwaliteit van ons algoritme voor het bijna-synchrone tijdsmodel, een nieuw tijdsmodel.

In hoofdstukken 5 en 6 behandelen we een verzamelprobleem op een draadloos netwerk waarbij we expliciet interferentie van radiosignalen modelleren. Interferentie beïnvloedt het ontwerp van algoritmen voor draadloze netwerken, omdat het een beperking oplegt aan het aantal knopen dat tegelijk kan communiceren. We zijn geïnteresseerd in het vinden van online algoritmen voor dit probleem die data zo snel mogelijk verzamelen. We gebruiken de completeringstijd en de doorlooptijd als maten om de kwaliteit van een oplossing te meten.

In hoofdstuk 5 analyseren we het probleem met als doel het minimaliseren van de completeringstijden. We richten ons op het minimaliseren van de maximale completeringstijd, dat wil zeggen de tijd die nodig is om alle data te verzamelen. We bewijzen dat dit probleem **NP**-moeilijk is, zelfs in het speciale geval dat elke knoop precies één bericht te versturen heeft. Hiermee geven we antwoord op een openstaand probleem. Daarnaast presenteren we een klasse van competitieve online algoritmen.

In hoofdstuk 6 behandelen we het verzamelprobleem met als doel het minimaliseren van de doorlooptijden, de tijd dat berichten in het netwerk zijn. We classificeren het probleem voor maximale en gemiddelde doorlooptijden; onze resultaten duiden erop dat het onwaarschijnlijk is dat er een efficiënt algoritme bestaat voor deze problemen. Vervolgens presenteren we een online gedistribueerd algoritme dat bijna optimale oplossingen geeft, als het algoritme berichten sneller kan versturen dan momenteel mogelijk is. Dit resultaat geeft aan welke kwaliteit het algoritme kan leveren na de aanschaf van snellere communicatiemiddelen.

Curriculum Vitae

Peter Korteweg was born on September 5, 1978 in Utrecht, The Netherlands. In 1996 he obtained a VWO diploma from the Christelijk Gymnasium in Utrecht. From 1996 to 2002 he studied Operations Research & Management at the University of Amsterdam. During his studies he spent the spring semester of 2001 at the University of Rome "La Sapienza", in Italy. He wrote his Master thesis under supervision of dr. A. Volgenant. In 2002 he graduated with honours and received a Master of Science degree. From September 2002 to January 2004 he worked as a consultant for ORTEC, a logistics IT supplier.

In February 2004 he started as a Ph.D. student at the Faculty of Mathematics & Computer Science of Eindhoven University of Technology. He worked under supervision of dr. L. Stougie and prof. A. Marchetti-Spaccamela from University of Rome "La Sapienza". During his Ph.D. studies he completed the Ph.D. program of the LNMB, the Dutch network on the mathematics of operations research. In 2007 he was awarded the second prize in the ROADEF Challenge, in the student category. The challenge consisted of designing an algorithm for a planning system for France Telecom, and was organized by ROADEF, the French society of operations research and decision analysis.
The results of his theoretical research are presented in this thesis.

Peter lives in Amsterdam, and he is co-founder and board member of Creditino, a non-profit microfinance organization.