TKK Dissertations in Information and Computer Science
Espoo 2010 TKK-ICS-D22

DISTRIBUTED OPTIMIZATION ALGORITHMS FOR MULTIHOP WIRELESS NETWORKS

André Schumacher

Aalto University
School of Science
and Technology

TKK Dissertations in Information and Computer Science
Espoo 2010 TKK-ICS-D22

DISTRIBUTED OPTIMIZATION ALGORITHMS FOR MULTIHOP WIRELESS NETWORKS

André Schumacher

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Information and Natural Sciences for public examination and debate in Auditorium M1 at the Aalto University School of Science and Technology (Espoo, Finland) on the 17th of December, 2010, at 12 noon.

Aalto University School of Science and Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Aalto-yliopiston teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Distribution:
Aalto University School of Science and Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science
PO Box 15400
FI-00076 AALTO
FINLAND
URL: http://ics.tkk.fi
Tel. +358 9 470 01
Fax +358 9 470 23369
E-mail: series@ics.tkk.fi

ISBN 978-952-60-3480-5 (Print)
ISBN 978-952-60-3481-2 (Online)
ISSN 1797-5050 (Print)
ISSN 1797-5069 (Online)
URL: http://lib.tkk.fi/Diss/2010/isbn9789526034812/

Multiprint
Espoo 2010

ABSTRACT:

Recent technological advances in low-cost computing and communication hardware design have led to the feasibility of large-scale deployments of wireless ad hoc and sensor networks. Due to their wireless and decentralized nature, multihop wireless networks are attractive for a variety of applications. However, these properties also pose significant challenges to their developers and therefore require new types of algorithms. In cases where traditional wired networks usually rely on some kind of centralized entity, in multihop wireless networks nodes have to cooperate in a distributed and self-organizing manner. Additional side constraints, such as energy consumption, have to be taken into account as well.

This thesis addresses practical problems from the domain of multihop wireless networks and investigates the application of mathematically justified distributed algorithms for solving them. Algorithms that are based on a mathematical model of an underlying optimization problem support a clear understanding of the assumptions and restrictions that are necessary in order to apply the algorithm to the problem at hand. Yet, the algorithms proposed in this thesis are simple enough to be formulated as a set of rules for each node to cooperate with other nodes in the network in computing optimal or approximate solutions. Nodes communicate with their neighbors by sending messages via wireless transmissions. Neither the size nor the number of messages grows rapidly with the size of the network.

The thesis represents a step towards a unified understanding of the application of distributed optimization algorithms to problems from the domain of multihop wireless networks. The problems considered serve as examples for related problems and demonstrate the design methodology of obtaining distributed algorithms from mathematical optimization methods.

KEYWORDS: ad hoc network, approximation algorithm, distributed algorithm, lifetime maximization, network optimization, network utility maximization, optimization algorithm, primal-dual algorithm, routing, sensor network, sleep scheduling, wireless communication

CONTENTS

1 Introduction 1

2 Multihop wireless networks 4
- 2.1 Graph models . . . 5
- 2.2 Ad hoc networks . . . 9
  - 2.2.1 Medium access control . . . 10
  - 2.2.2 Routing algorithms . . . 12
- 2.3 Wireless sensor networks . . . 14
  - 2.3.1 Topology control . . . 15
  - 2.3.2 Lifetime maximization . . . 16
  - 2.3.3 Sleep scheduling . . . 17
- 2.4 Network simulators . . . 18

3 Linear and convex programming 21
- 3.1 Approximation algorithms . . . 21
- 3.2 Linear and integer linear programming . . . 22
  - 3.2.1 Linear programming duality . . . 23
  - 3.2.2 Primal-dual algorithms . . . 24
  - 3.2.3 Network optimization based on LP relaxation . . . 31
- 3.3 Convex programming . . . 34
  - 3.3.1 Lagrangian duality . . . 36
  - 3.3.2 Algorithms . . . 39

4 Distributed algorithms 41
- 4.1 Model and assumptions . . . 41
- 4.2 Algorithms . . . 45
  - 4.2.1 Spanning trees . . . 45
  - 4.2.2 Shortest paths . . . 47
  - 4.2.3 Independent sets . . . 51
  - 4.2.4 Dominating sets . . . 53
  - 4.2.5 Network synchronizers . . . 55

5 Load balancing via multipath routing 57
- 5.1 Introduction . . . 57
- 5.2 Load balancing by congestion minimization . . . 58
  - 5.2.1 The BMSR algorithm . . . 58
  - 5.2.2 Discussion . . . 60
- 5.3 Simulation and performance evaluation . . . 62
  - 5.3.1 Overview of experiments . . . 62
  - 5.3.2 Two-pair cross setup . . . 63
  - 5.3.3 Multiple source-destination pairs . . . 69

6 Network utility maximization with path constraints 73
- 6.1 Introduction . . . 73
- 6.2 Distributed algorithm . . . 77
  - 6.2.1 Fixed selection of paths . . . 78

6.2.2 Adaptive path selection 81
6.2.3 Distributed implementation 84
6.3 Centralized algorithm 86
6.4 Numerical experiments 87

7 Transmission power assignment in sensor networks 91
7.1 Introduction 91
7.2 Maximum lifetime spanner algorithm 95
7.2.1 Minmax spanner computation 95
7.2.2 Improvement for Euclidean space 99
7.3 Binary search for minmax power spanners 100
7.4 Algorithm initialization and termination 104
7.4.1 Setup stage 105
7.4.2 Notification stage 107
7.4.3 Distributed algorithm for RNG computation 108
7.5 Experimental evaluation 108

8 Sleep-scheduling in sensor networks 115
8.1 Introduction 115
8.2 Sleep scheduling algorithm 116
8.3 Minimum weight dominating set approximation 120
8.3.1 Problem formulation 121
8.3.2 Distributed voting scheme 122
8.3.3 Implementation 125
8.4 Experimental evaluation 130

9 Conclusions 133

Bibliography 135

Index 149

LIST OF FIGURES

2.1 Data gathering scenario 5
2.2 Hidden and exposed terminal problem 10
2.3 Effect of path loss on energy-efficient routing 16
2.4 Difference between fractional and integral domatic partitions 18
3.1 A pair of a primal and dual linear programs. 24
3.2 Space of primal and dual solutions for covering/packing LPs . 24
4.1 Possible execution of a convergecast operation 50
5.1 Setup for NS2 simulations 63
5.2 Average throughput of source-destination pairs for BMSR and DSR 64
5.3 Distribution of packet load, collisions and IFQ overflows in the network for DSR and BMSR 66
5.4 Situation in which non-shortest routes lead to interference . . 67
5.5 Throughput results for DSR, BMSR and random routing . . 67
5.6 Throughput and delay versus packet size for DSR and BMSR 68
5.7 Average throughput for BMSR and DSR for random compared to grid placement 69
5.8 Simulation setups for multiple source-destination pairs: dense, sparse, twisted 70
5.9 Average throughput for dense placement: DSR, BMSR and shortest path routing 71
5.10 Average throughput for sparse placement: DSR, BMSR and shortest path routing 72
5.11 Average throughput for twisted placement: DSR, BMSR and shortest path routing 72
6.1 Transformations between problems: NUM, KNUM, etc. . . 76
6.2 High-level outline of the proposed dual-decomposition method 77
6.3 Problem instance with 22 nodes and 4 source-destination pairs 88
6.4 Evolution of path rates for each source-destination pair 89
6.5 Histogram for the value $K_c - K_m$ for a set of 21 random graphs with 22 nodes. 90
7.1 Sample execution of Algorithm MLS 98
7.2 Single iteration of BSPAN 103
7.3 Minmax spanners of a graph with 100 nodes computed by the various algorithms 112
7.4 Number of control messages for DMMT, MLS and BSPAN . 113
7.5 Number of messages required by BSPAN versus network size . 113
7.6 Execution time for DMMT, MLS and BSPAN 114
8.1 Example of continuous-time greedy algorithm for MWDS . . 122
8.2 Example of voting procedure 124
8.3 Performance of combined algorithm GKSCHED-ASYNMWDS versus node density 131
8.4 Number of messages and time required per iteration for algorithm GKSCHED-ASYNMWDS versus node density 132

LIST OF ALGORITHMS

PDSC	Primal-dual algorithm for weighted set cover	27
GKMMF	Garg-Könemann multicommodity flow algorithm	29
YMMC	Young's algorithm for minmax congestion	31
BBKMMF	Branch-and-Bound for path-constrained flow	34
SHOUT	SHOUT algorithm for spanning trees	46
BF	Distributed asynchronous Bellman-Ford algorithm	48
MIS	Distributed maximal independent set algorithm	52
SYNMWDS	Distributed synchronous MWDS algorithm	54
BMSR	Balanced multipath source routing algorithm	59
CENDD	Centralized dual-decomposition for KNUM	82
DISTDD	Distributed dual-decomposition for KNUM	85
MLS	Maximum lifetime spanner algorithm	97
BSPAN	Binary search for minmax spanner algorithm	101
BEACSET	Beaconing algorithm for network setup	106
GKSCHED	Distributed GK algorithm for sleep scheduling	118
ASYNMWDS	Distributed asynchronous MWDS algorithm	126

LIST OF SYMBOLS

$\|\| \cdot \|\|_2$	Euclidean norm
δ_v	*degree* of v, i.e., the number of neighboring nodes of v
Δ	maximum degree, i.e., $\Delta = \max_{v \in V} \delta_v$
Δ^+	maximum *extended degree* $\Delta^+ = \Delta + 1$
δ	minimum degree, i.e., $\delta = \min_{v \in V} \delta_v$
δ^+	minimum *extended degree* $\delta^+ = \delta + 1$
D	*dominating set*
E	set of *edges*
ϕ	*approximation factor*
G	*graph*, where $G = (V, E)$
∇f	*gradient* of function f
H_n	n-th *harmonic number*, i.e., $H_n = \sum_{k=1}^{n} 1/k$
K	limit on number of non-zero flow paths in a solution to a multi-commodity flow problem
L	Lagrangian function
$N(v)$	set of *neighbors* of v in G
$N^+(v)$	*extended neighborhood* $N(v) \cup \{v\}$ of v
$N_k(v)$	k-hop *neighborhood* of v, i.e., set of nodes at a hop distance of exactly k from v
$N_k^+(v)$	k-hop *extended neighborhood* of v, i.e., $N_k^+(v) = N_{k-1}^+(v) \cup N_k(v)$ for $k \geq 1$, where $N_0^+(v) = \{v\}$.
$N_T(v)$	set of *tree neighbors* of v, where $N_T(v) \subseteq N(v)$
$N_{\mathbf{in}}(v)$	set of *incoming neighbors* of v
$N_{\mathbf{out}}(v)$	set of *outgoing neighbors* of v
p	*path*
$\mathcal{P}_n$	set of all paths between a source-destination pair (s_n, t_n)
$\mathcal{P}$	set of all paths for a collection of source-destination pairs, i.e., $\mathcal{P} = \cup_n \mathcal{P}_n$
V	set of *vertices* (or *nodes*)
$w(S)$	*weight* of set S, where $w(S) = \sum_{e \in S} w(s)$

LIST OF ACRONYMS

AODV	Ad Hoc On Demand Distance Vector Routing Protocol
BF	Bellman-Ford Algorithm
BIG	Bounded Independence Graph
BMSR	Balanced Multipath Source Routing Algorithm
BREP	Balanced Route Reply
BREQ	Balanced Route Request
BSpan	Binary Search for Minmax Power Spanner Algorithm
CBR	Constant Bit Rate
CG	Communication Graph
CTS	Clear to Send
DMMT	Distributed Min-Max Tree Algorithm
DSR	Dynamic Source Routing Protocol
GKMMF	Garg-Könemann Maximum Multicommodity Flow Algorithm
IFQ	Interface Queue
ILP	Integer Linear Program
IP	Internet Protocol
KKT	Karush-Kuhn-Tucker Conditions
KMMF	K-path Constrained Maximum Multicommodity Flow Problem
KNUM	K-path Constrained Network Utility Maximization Problem
LP	Linear Program
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MCF	Maximum Concurrent Flow
MLS	Maximum Lifetime Spanner Algorithm
MMC	Minimum Maximum Congestion Problem
MMF	Maximum Multicommodity Flow Problem
MSS	Maximum Length Sleep Schedule Problem
MST	Minimum Spanning Tree
MWDS	Minimum Weight Dominating Set
ns2	network simulator
NUM	Network Utility Maximization Problem

PDSC	Primal-Dual Set Cover Algorithm
PTAS	Polynomial-Time Approximation Scheme
RNG	Relative Neighborhood Graph
RREP	Route Reply
RREQ	Route Request
RSSI	Received Signal Strength Indication
RTS	Request to Send
SINR	Signal-to-Interference-and-Noise Ratio
SPR	Shortest Path Routing
TCP	Transmission Control Protocol
UDG	Unit Disk Graph
WSN	Wireless Sensor Network
YMMC	Young Minimum Maximum Congestion Algorithm

PREFACE

While carrying out research that resulted in this thesis, the author has been supported by the Academy of Finland under grants #209300 and #128823. The author held a graduate student position funded by the Department of Computer Science and Engineering of the former Helsinki University of Technology (TKK) and was later supported by the Helsinki Graduate School in Computer Science and Engineering (Hecse). The author is grateful for having received funding that allowed him to pursue his studies full time. The personal grant awarded by the Nokia Foundation is gratefully acknowledged. The author is also thankful for having received a thesis completion grant by the Faculty of Information and Natural Sciences.

I am deeply grateful to my supervisor Professor Pekka Orponen for the guidance, support and encouragement he provided me during the past five years. Despite a busy schedule, he somehow managed to find time for feedback and discussion, sometimes long after office hours. I am thankful to Harri Haanpää, who, besides being my co-author, colleague and office neighbor for several years, was always there for discussions and helped me navigate bureaucracy and the occasional LaTeX nightmares. I also thank my other co-authors, Shreyas Prasad, Satu Elisa Schaeffer and Thorn Thaler. It was a pleasure to work with you.

This thesis would not have been possible without the enjoyable atmosphere and excellent working conditions at the Department of Information and Computer Science. I would like to thank all my colleagues, both new and old, for their support and for keeping the department running in exciting times of change. In particular, I thank Emilia Oikarinen for letting me use her picture of the computer science building on the cover page. I am also grateful to Yoan Miche for helping me with the creation of the same page and also providing his as a template.

I am indebted to my friends and family, who supported and encouraged me during the last few years. Particularly, I thank the Finnish language course lunch group for entertaining discussions over a variety of topics, ranging from circuit design, over automation technology and architecture to biochemistry and life of a graduate student. I am most grateful to my parents for their support and also to my father for proofreading an early version of the manuscript. For her unconditional support and patience with this thesis project I thank Satu Maarit.

Further, I would like to thank the two pre-examiners of this thesis, Elad Schiller (Chalmers University of Technology and University of Gothenburg) and Patrik Floréen (University of Helsinki), for providing valuable comments that helped to improve the manuscript. I am also grateful to Patrik Floréen for teaching a seminar course on algorithms for ad hoc networking in 2003, which stirred my interest in the topic. Finally, I would like to thank Professor Thomas Erlebach (University of Leicester) for the honor of acting as opponent.

André Schumacher
Helsinki, November 2010

1 INTRODUCTION

In recent years, we have witnessed the emergence of large-scale distributed computing systems. Without a doubt, the best example is the Internet, whose success would have been hard to predict just about two decades ago. These systems are inherently decentralized, non-hierarchical, scalable and resilient to errors. Their properties can only be partly explained by the ingenuity of the engineers who designed the protocols that are providing the basis of today's computing and communication infrastructure. Since also the early engineers could not possibly have foreseen such a rapid growth, the question of what makes some protocols perform and scale so well remains to be answered.

Recently, it was discovered that some protocols can be seen as algorithms that *implicitly* solve an optimization problem that characterizes optimal states of the distributed system. The best example certainly is the Transmission Control Protocol (TCP), which is the workhorse of the Internet. This series of work was initiated by the article of Kelly et al. [74], who modeled congestion control algorithms, such as methods that are part of the different variants of TCP, as algorithms that distributively solve a pair of primal and dual problems. Later work by Low [89] studied particularly congestion control in the Internet and led to insights into the operation of the different TCP variants, which are still in use today.

The observation that it is possible to derive optimization problems by reverse-engineering network protocols leads to a new paradigm of network-protocol design. Rather than considering network optimization based on top of existing protocols, several researchers propose to first look for the proper problem formulation and then design network topology and protocols based on an algorithmic solution. He et al. [59] distinguish these "optimizable networks" from conventional "network optimization" approaches. Besides providing an understanding of the operation of the resulting protocol, in some cases this approach also naturally gives rise to a decomposition of the problem, which may then be solved on different layers of the protocol stack. Chiang et al. [29] introduce a whole theory of network decomposition and layering approaches and extend the methodology to other problem settings.

When formulating a distributed algorithm to solve a certain network optimization problem, one naturally has to make assumptions on the environment the algorithm will eventually be executed in. Typical assumptions include synchronous execution, i.e., the presence of a global clock, absence of failures, or a certain type of network topology. These premises then impact the formulation of the algorithm and the eventual complexity of its implementation, since an algorithm based on a weaker execution model requires a more intricate design, e.g., to handle errors correctly. Assuming a very strong execution model, on the other hand, may render the implementation of the algorithm difficult or even impossible.

Besides assumptions on the execution environment, sometimes one needs to restrict the problem formulation itself. Typical restrictions include certain ranges for problem parameters or the assumption of nodes being located in Euclidean space. If some of these assumptions are not satisfied, the result of

the algorithm, in terms of feasibility and optimality of a solution, should degrade gracefully. A different type of assumption only simplifies presentation and may be removed using relatively minor modifications of the algorithm if required.

When surveying the distributed algorithms and network optimization literature, one cannot avoid but notice the significant gap between the theoretician and practitioner sides of the research community. Obviously, it can hardly be the goal of this thesis to bridge this gap. However, it is our aim to study mathematical properties from network-optimization theory and apply these to practical problems arising in wireless networks to obtain simple algorithms that can be implemented based on a realistic network infrastructure. For this purpose, we approach various problems, such as routing, transmission power assignment and node activity scheduling. For most algorithms proposed in this thesis we also provide network simulations that offer an insight into the operation of an algorithm. Rather than considering the algorithms in isolation, these should be seen as a step towards a mathematical toolbox consisting of techniques for approaching problems by distributed network optimization. In this context, one interesting observation, which we have made on several occasions, is the observation that *dual problems* sometimes lead to algorithms that can be implemented efficiently in a distributed setting. We believe that methods for exploiting this *locality by duality* principle are promising candidates for future generation network protocols.

The following chapters can be broadly separated into two parts. Chapters 2 to 4 introduce preliminaries, formalize the models and techniques and review relevant literature. The second part of the thesis, Chapters 5 to 8, discusses own work and can be subdivided into two parts of its own. Chapters 5 and 6 address problems that are mostly relevant to ad hoc networks, while the topics of Chapters 7 and 8 are more relevant to sensor networks.

More precisely, in Chapter 2 we discuss graph models for ad hoc and sensor networks and highlight algorithmic challenges in their context. Chapter 3 gives a brief overview of relevant topics from linear programming and convex optimization literature. We review the model of distributed computation used in the thesis and introduce further terms and concepts in Chapter 4. The first chapter of the second part, Chapter 5, proposes a multipath routing algorithm for ad hoc networks that is based on a linear programming formulation of a network flow problem to minimize congestion. In Chapter 6 we consider a related problem, which we call the *fair multicommodity flow* problem, since it takes into account fairness between the different source-destination pairs. We then proceed to problems that address the energy-efficient operation of wireless sensor networks. In Chapter 7 we propose two algorithms for assigning transmission power levels to network nodes, which guarantee the connectedness of the resulting topology and are low enough to achieve maximum network lifetime. We approach the problem of lifetime maximization from a different point of view in Chapter 8, where we propose a distributed algorithm for approximating optimal *sleep schedules*.

Chapter 5 is based on publications [112, 125], which were co-authored with Harri Haanpää, Pekka Orponen, Shreyas Prasad and Satu Elisa Schaeffer. Chapter 6 describes work published in [123], which is part of joint work with Harri Haanpää. The following Chapter 7 is based on article [54],

which unifies work published in conference proceedings [55, 126] that were co-authored with Harri Haanpää, Pekka Orponen and Thorn Thaler. Finally, Chapter 8 discusses joint work with Harri Haanpää that was published in [124]. The list of main contributions of the co-authors to material included in this thesis is the following:

- The second part of the simulations of the BMSR algorithm in Chapter 5, as presented in [112], is due to Shreyas Prasad, who used the implementation of the algorithm proposed in [125].

- The idea of the pruning condition in the MLS algorithm of Chapter 7 is due to Harri Haanpää; the original idea of using a proximity graph to lower MLS message complexity is due to Pekka Orponen. The MLS and BSPAN algorithms were implemented in NS2 by Thorn Thaler; the simulations presented in [54, 55, 126] were conducted in collaboration between Thorn Thaler and the author.

The contributions of the author lie in the design, analysis and simulation of distributed algorithms for multihop wireless networks presented in this thesis. The work was done while the author was a member of the Combinatorial Algorithms and Complexity group of Professor Pekka Orponen at the Helsinki University of Technology, TKK, which later merged with two universities to become Aalto University.

The simulations that were used to evaluate the algorithms proposed in this thesis were performed with the network simulator NS2 [93]. Additionally, several experiments were implemented using Matlab and the implementation of the branch-and-bound algorithm of Chapter 6 uses the Mosek optimization tools [97]. The figures and plots in this thesis were generated using gnuplot and xfig and the document itself was written in LaTeX 2_ε.

2 MULTIHOP WIRELESS NETWORKS

A *multihop wireless network* consists of a collection of nodes that are able to communicate via wireless communication without any centralized or fixed infrastructure. The term *multihop* refers to the requirement that in addition to originating and receiving messages, nodes also act as *routers* and forward messages for other nodes. Hence, the operation of the network requires cooperation among them. Since there is no central authority, this cooperation needs to be coordinated in a distributed and self-organizing manner. Furthermore, nodes often only have low computing power. Many applications require them to operate without relying on an inexhaustible energy source, i.e., nodes are battery-powered. Therefore, energy consumption is a critical constraint on the lifetime of the network. Due to their wireless and decentralized nature, these types of networks are attractive for a variety of applications (see [50, 63] and below for an overview). However, their properties also pose significant challenges to developers.

In this thesis, we broadly distinguish between two classes of multihop wireless networks, *ad hoc networks* and *sensor networks*. Although both types of networks have important similarities, some of their design criteria may differ in relative importance. Ad hoc network nodes, for example, may possess rechargeable batteries while sensor network nodes are typically deployed once and operate until their batteries become exhausted. Ad hoc network nodes also may be mobile, whereas sensor networks are normally considered stationary.

Ad hoc networks: Applications Typical areas of applications for ad hoc networks include collaborative computing, disaster-relief scenarios, communication between vehicles and military applications [21, 81]. Two special classes should be mentioned. In the case that nodes are mobile and node mobility is an integral aspect of the network, the term MANET, which stands for mobile ad hoc network, is normally used [30]. Another special type of non-mobile ad hoc networks, *mesh networks*, have recently moved into the focus of attention [22]. Commercial applications that provide wireless Internet access based on mesh networks are already available. Other mesh networks are formed by neighborhood communities for the same purpose. For examples, see the survey by Bruno et al. [22].

Ad hoc networks: Algorithmic challenges Two main challenges in ad hoc networks are the setup and maintenance of routing [103]. Energy-efficient operation is an important design goal [39]. However, given that ad hoc network nodes may consist of devices such as laptops, handheld devices, or smart phones, energy efficiency may be secondary to other goals. The concept of fairness among users of the network is crucial, since routes consist of multiple hops and users compete for network resources. In the presence of node mobility, tasks such as route maintenance become especially challenging.

Sensor networks: Applications Sensor networks are interesting for a large number of possible applications, such as habitat monitoring, vehicle track-

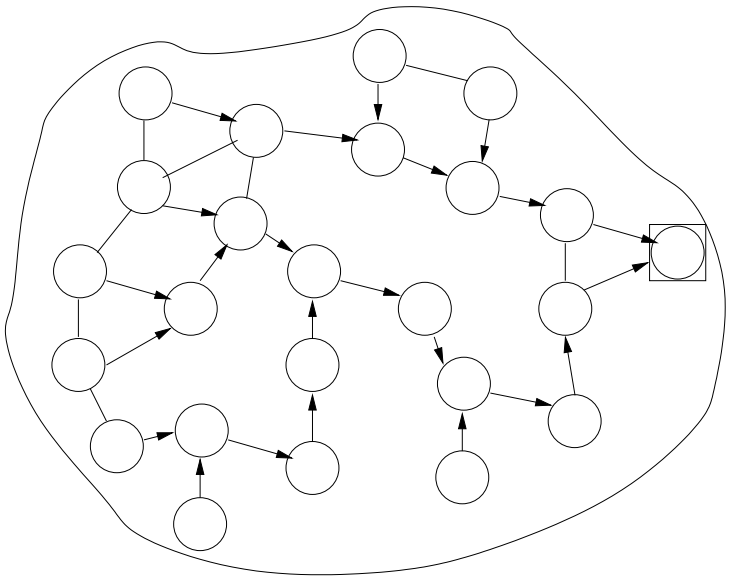


Figure 2.1: Sensor network operating in a data gathering scenario; the sink node is labeled by a square; the direction of messages passed towards the sink is indicated by arrows.

ing, inventory control, parcel tracking, industrial plant monitoring and military applications [2, 36, 60]. In many applications sensor networks operate in a *data-gathering* scenario, where nodes use multihop routing to send data to the sink via the edges of a spanning tree that is rooted at the sink, as shown in Figure 2.1. The data generated by the network can be the result of certain events sensed by the nodes or consist of periodic messages. The sink also may initiate active queries that propagate in the network and may be replied to by a subset of the nodes.

Sensor networks: Algorithmic challenges Sensor networks give rise to interesting algorithmic challenges, such as the setup of routing information, transmission power assignment, sleep scheduling, data aggregation and query processing, just to name a few [2, 28]. Since nodes are battery-powered and the replenishment of batteries is usually impossible, energy-efficient operation takes an important position in the list of objectives to be optimized.

2.1 GRAPH MODELS

Since wireless networks are typically modeled using graph models, we begin by introducing notation that will be used later on. Most of the following definitions and notation can be found in any standard textbook on graph theory, e.g., in the textbook by Diestel [37].

We follow conventional notation and denote an *undirected graph* as a pair $G = (V, E)$, where V is a set of *vertices* (or *nodes*) and $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a set of *edges*. We say that two vertices $u, v \in V$ are *adjacent* (or *neighbors*) if $\{u, v\} \in E$. An edge $e \in E$ is *incident* to a vertex v if $e = \{u, v\} \in E$ for some $u \in V$. The *degree* of v is the number of

neighbors it has in the graph and is also denoted by δ_v. Since most graphs considered in this thesis are undirected, we use the convention of referring to an undirected graph when nothing else is mentioned.

A *path* p from a start-vertex u to an end-vertex v is a sequence of edges $p = (\{u, w_1\}, \{w_1, w_2\}, \ldots, \{w_k, v\})$, where the w_i are mutually distinct.[1] The path is a *cycle* if $u = v$. We write $e \in p$ for an edge $e \in E$ and a path p to denote that e is on the path p. The *length* of a path is the number of edges it contains. If the edges of a graph are weighted, i.e., there exists a function $w : E \mapsto \mathbb{R}$, the length of a path is defined as the sum of the weights of its edges. We call G *connected* if there exists a path between any pair of vertices in V. The length of a shortest path between two vertices in a connected graph is the *distance* between them. The *diameter* of a connected graph is the maximum distance between any pair of vertices. A graph that consists of a single path from some start to some end-vertex is also called a *chain graph*.

Denote by $N(v)$ the neighbors of v in G and define $N^+(v)$ to be the *extended neighborhood* $N(v) \cup \{v\}$ of v. Note that $|N(v)| = \delta_v$ and $|N^+(v)| = \delta_v + 1$. Define $\delta = \min_{v \in V} \delta_v$ and $\delta^+ = \delta + 1$ to be the minimum degree and minimum extended degree, respectively. Similarly, define Δ and Δ^+ to be size of the largest neighborhoods. We define the k-hop neighborhood $N_k(v)$ of v as the set of nodes which are at a hop-distance of exactly k from v, where $N_0(v) = \{v\}$. Note that $N_1(v) = N(v)$. Similarly, we inductively define the k-hop extended neighborhood of v, i.e., all nodes at a hop-distance of at most k from v, by $N_k^+(v) = N_{k-1}^+(v) \cup N_k(v)$ for $k \geq 1$, where $N_0^+(v) = \{v\}$.

Any graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ is called a *subgraph* of $G = (V, E)$. A graph is a *tree* if it is connected and does not contain any cycles. A *spanning tree* of a connected graph G is a subgraph $T = (V, E')$ that is a tree. If the graph G is edge-weighted by a function $w_E : E \mapsto \mathbb{R}_{>0}$, then a *minimum spanning tree* (MST) of G is a spanning tree of minimum weight, where the weight is taken as the sum of the weights of the edges in the tree. The *subgraph* induced by a subset $V' \subseteq V$ of the vertices of G is the graph $H = (V', E')$, where $E' = \{\{v, u\} \in E \mid v, u \in V'\}$.

A tree is called *rooted* if it contains a designated node, called the *root*. In a rooted tree all edges can be oriented towards the root. We call the next node u on a path from node v to the root the *parent* (or *father*) of v and call v a *child* of u. When we are given a tree T that is a subgraph of a graph G, we may denote the set of *tree neighbors* of v by $N_T(v)$, where $N_T(v) \subseteq N(v)$ is the set of neighbors of v that are also its neighbors in the tree, which includes its parent if the tree is rooted. All nodes with degree one in a tree T (except the root if $|V| > 1$) are called *leaf* nodes and the other nodes are called *internal*. The length of the longest path from the root to any leaf is the *height* of the tree, regarding edges as unweighted.

We call a set $D \subseteq V$ a *dominating set* of $G = (V, E)$, if every vertex in V that is not in D has a neighbor in D. The set D is a *minimum dominating set*, if among all dominating sets of G it has the minimum size $|D|$. If the graph G is node-weighted, i.e., if there is a function $w_V : V \mapsto \mathbb{R}_{>0}$, then a *minimum weight dominating set* (MWDS) D is a dominating set of minimum total weight, summed over all nodes in D. Any dominating set D that induces a

[1] This type of path is normally referred to as a *simple path*.

connected subgraph of G is called a *connected dominating set*.

A set $I \subseteq V$ is an *independent set* of $G = (V, E)$, if there is no edge $\{u, v\} \in E$ between any pair of vertices $u, v \in I$. An independent set is *maximal* if adding any vertex from $V \setminus I$ would break the independence property. It follows that a maximal independent set is also always a dominating set. A *maximum independent set* is an independent set with maximum cardinality. The notion of a maximum weight independent set is defined analogously to a minimum weight dominating set.

We define a *minmax spanner* of an edge-weighted graph $G = (V, E)$ to be a subgraph $G' = (V, E')$ that is connected and has minimum maximum edge weight among all such graphs. Note that a minmax spanner does not need to be a tree. More generally, a subgraph $G' = (V, E')$ of G is an α-*spanner* if G' is connected and $w_E(e) \leq \alpha$ for each edge $e \in E'$. Hence, a minmax spanner is an α-spanner with the property that no α'-spanners exist for $\alpha' < \alpha$. One can see easily that an MST is always a minmax spanner.

A graph G is called *directed* if $E \subseteq V \times V$, i.e., E consists of ordered pairs of vertices so that an edge (u, v) has the direction from u to v. For a directed graph $N_{\text{in}}(v)$ denotes the set of *incoming neighbors* of v, i.e., $N_{\text{in}}(v) = \{u \in V \mid (u, v) \in E\}$ and correspondingly the set of *outgoing neighbors* $N_{\text{out}}(v)$. We say that a directed graph is *strongly connected* if there exists a (directed) path from each node to any other node. A directed graph is called *symmetric* if $(u, v) \in E$ implies $(v, u) \in E$ for all $u, v \in V$.

Communication graph

The distributed algorithms discussed in this thesis assume an underlying communication structure, which we call *communication graph*. More precisely, if $G = (V, E)$ is a communication graph for a given network, then V corresponds to the set of network nodes and E represents the links between them. A communication graph is usually directed, but we may assume that its edges are symmetric, which means that network links are bidirectional. In this case we may regard the communication graph as being undirected, so that $N_{\text{in}}(v) = N_{\text{out}}(v) = N(v)$ for all nodes v.

In general, a node v is able to send messages to its outgoing neighbors in $N_{\text{out}}(v)$ and receive messages from its incoming neighbors $N_{\text{in}}(v)$. However, sometimes a message cannot be transmitted due to the transmitter sensing the channel busy or because the message collides with another transmission at the receiver. Hence, the communication graph models potential communication via neighboring nodes under ideal conditions. We will always assume that the communication graph is (strongly) connected.

Sometimes we make additional assumptions about the network. The communication graph, for example, may not be predetermined by the deployment, but can be induced by variables, such as transmission-power levels assigned to the nodes. As a special case one can then consider the communication graph that results from all nodes transmitting at their maximum available transmission power.

Some papers assume a communication graph that is time-varying (see for example [38] and the references therein). A time-varying graph may be used to model node mobility and permanent link or node failures. In this thesis,

however, we only consider transient link errors, i.e., transmission errors, and no node failures. We further assume that if node mobility is present, then it happens at a slow time-scale so that algorithms can be rerun periodically to handle changes in topology. The communication graphs considered here are therefore static.

Disk graphs

Disk graphs have been used extensively as models for wireless networks. See, for example, [83] for a review on algorithmic graph issues in wireless networks. Disk graphs are popular candidates for communication graphs due to their simplicity. A disk graph has several geometric representations, also called *realizations*, besides its representation as a graph.

In the geometric *containment model* of a disk graph, the graph is given as a set of n circles in the Euclidean plane. Each circle represents a vertex at its center. A vertex u is a neighbor of vertex v if the disk of v contains the center of u, where one assumes that the circle area contains its boundary. Thus, given this realization, one can construct a graph that contains a vertex for each disk and an edge between vertices that are neighbors.

The *proximity model* of a disk graph is given by a set of n points in the Euclidean plane and a radius for each of the points. A vertex u is a neighbor of vertex v if u is at distance of at most r_v from v, where r_v is the radius of v. In this context, r_v is called the *transmission radius* of v.

A disk graph where each circle has the same size (or radius) is called *unit disk graph* (UDG), since one can scale the radius appropriately without changing the graph structure. Although a number of interesting optimization problems remain NP-hard for UDG, there exist efficient approximation algorithms for a variety of problems. See, for example, Clark et al. [32] for results on the computational complexity for various problems and Marathe et al. [91] for approximation algorithms for UDG. It is important to note that the problem of deciding whether a given graph $G = (V, E)$ has a realization as a unit disk graph is NP-hard [20].

Other graph models for wireless networks

It is generally accepted that unit disk graphs model an idealized situation that does not represent realistic conditions in wireless networks accurately [79]. In reality, for instance, there is no sharp distance threshold for the existence of links between nodes and therefore the area limited by the transmission range does not have the shape of a circle. Also, disk graphs do not allow the inclusion of obstacles and other effects that frequently occur in indoor environments. Due to the limitations of the UDG model, several other models for wireless networks have been proposed. For an overview see [80, 122].

The *bounded independence graph* (BIG) [122] model is an interesting generalization of a UDG. Let $I_k(v)$ denote an independent set in the subgraph induced by $N_k^+(v)$. Then a graph $G = (V, E)$ satisfies the *bounded independence condition* if and only if $|I_k(v)| \leq f(k)$ for all v, where $f(k)$ is a polynomial function in k. A UDG is a BIG, since one can show that for a UDG this condition is satisfied for $f(k) := (2k + 1)^2$ [3].

Relative neighborhood graphs

Relative neighborhood graphs were originally introduced by Toussaint [133] for nodes that represent points in the plane. Their definition, however, can easily be extended to non-geometric contexts. Given an edge-weighted graph $G = (V, E)$ with weight function $w_E : E \mapsto \mathbb{R}_{>0}$, we informally define a *relative neighborhood graph* (RNG) of G as follows.

Definition 1. *Given a graph $G = (V, E)$ and function w_E, the relative neighborhood graph of G is the graph with vertex set V and edge set*

$$E' = \{e \mid e \in E \text{ and there is no path } p \text{ in } G \text{ between the endpoints of } e, \text{ s.t. } p = (e_1, e_2) \text{ and } w_E(e_1) < w_E(e), w_E(e_2) < w_E(e)\}.$$

Hence, the RNG is obtained from G by removing all edges whose endpoints can be connected by a path consisting of two edges with smaller weight. Such a generalization of the concept of RNG has been already successfully applied to other problems, such as searching and broadcasting in peer-to-peer networks [40].

Claim 1. *For any α, the* RNG *of G contains an α-spanner if G does.*

Proof. Consider the edges in $E \setminus E'$ that were removed during the construction of the RNG and order these in decreasing order of cost as $e_1, e_2, \ldots, e_k$. Let $E_0 = E$ denote the edge set of the original graph, and let $E_i = E_{i-1} \setminus \{e_i\}$ for $0 < i \leq k$. Assume that G admits an α-spanner. For $0 < i \leq k$ it follows that since E_{i-1} admits an α-spanner and the endpoints of e_i are connected by a path of two cheaper edges, E_i also admits an α-spanner. Hence, the RNG $(V, E_k) = (V, E')$ admits an α-spanner. □

The RNG of a given graph has beneficial properties that are interesting for certain applications in wireless networks. For related geometric graph models see, e.g., the work by Alzoubi et al. [3].

2.2 AD HOC NETWORKS

Ad hoc networks have been the topic of intensive research for the past few years and have their origin in the ALOHAnet and PRNET packet radio networks of the 1970s [81]. Despite their extensive range of application and the progress that has been reported in various publications, ad hoc network adoption to real-world applications has been unexpectedly slow [22, 135]. This is partly due to the complexity of the ad hoc networking paradigm, requiring the development of new solutions to problems that have been already solved efficiently for regular wired networks with a fixed infrastructure. For ad hoc networks, however, many of these are still awaiting efficient solutions although a variety of protocols have been proposed in the recent past.

Due to the lack of centralized control, an ad hoc network needs to integrate nodes into its operation and handle changes in connectivity, for example caused by node mobility, in a self-organizing manner. In addition to the lack of centralized coordination and reliance on battery power, the wireless

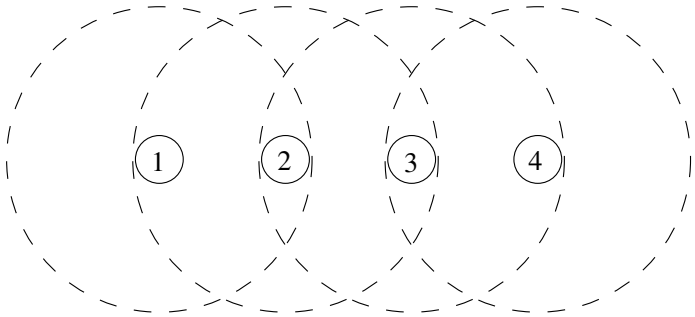


Figure 2.2: Situation in which the hidden terminal and the exposed terminal problem may occur; the dotted circle around each node represents the transmission radius.

transmission medium poses challenges to communication protocols. The communication channel has limited bandwidth and is prone to transmission errors caused by interference, signal attenuation and time-varying channel conditions. Although the broadcast nature of the medium offers potential benefits, it also gives rise to unique problems.

We now briefly discuss two classes of algorithms that have been proposed for ad hoc networks, medium access control (MAC) and routing algorithms. Algorithms that also have relevance for sensor networks are the topic of the following section.

2.2.1 Medium access control

A large number of protocols have been proposed that address the unique challenges of wireless medium access control. See the survey by Kumar et al. [81] for an extensive overview. Besides addressing typical design goals, such as low signaling overhead and large data throughput, these protocols also aim to solve issues related to the broadcast nature of the channel.

Figure 2.2 depicts a situation in which two problems, which are typically called the *hidden terminal problem* and the *exposed terminal problem*, may occur. Four nodes are located on the line. Each node is able to communicate directly only with its closest neighbors. Hence, the figure shows the realization of a four-node chain graph in the proximity model. Consider now the case where nodes 1 and 3 initiate transmissions to node 2 simultaneously. Since node 3 is not in the transmission range of node 1 and vice versa, neither of them notices that the channel is busy, which leads to a collision at node 2. This is referred to as the hidden terminal problem. In a different situation, when node 2 transmits to 1, node 3 senses the channel busy and delays its own transmission to node 4, which results in reduced throughput since the two transmissions could have proceeded concurrently. This problem is called the exposed terminal problem. Note that the effect of the hidden terminal problem is network collisions, whereas the exposed terminal problem leads to performance loss due to unrealized transmission capacity.

Protocols for wireless MAC typically take different approaches for avoiding or mitigating the effect of the two problems described above. Based on the approach, one can broadly classify MAC protocols as either *contention*

free or *contention based* [81]. Contention-free protocols employ techniques that allow for unimpeded channel access by dividing transmission time, frequencies, or coding schemes among nodes. Since these protocols require at least some kind of complex coordination, such as clock synchronization, contention-based protocols are used frequently. Contention-based protocols usually employ some kind of reservation of the channel and also formulate rules for recovering after collisions have occurred.

One contention-based protocol that forms the basis for a variety of ad hoc network MAC protocols is the method specified by the set of standards commonly known as IEEE 802.11. The functionality of MAC that is suitable for implementation in infrastructure-less networks is referred to as the *Distributed Coordination Function* [63]. In the following, we briefly outline the operation of this function. For details omitted here see, e.g., [63, 104].

According to the protocol specification, before transmitting a message, nodes wait for a fixed amount of time and sense whether the channel is busy. If the channel is free after the waiting period, a node transmits its data packet and waits for an acknowledgment, which the receiver will transmit after itself waiting for some time if the packet was received successfully. If the sender senses the channel busy, it delays its transmission until it becomes free. Thereafter, the sender delays its transmission further according to a scheme referred to as *binary exponential back-off*: The sender chooses a number from an interval $(0, \mathrm{CW})$ uniformly at random, where CW is a value called the *contention window*. It then determines the amount of time to wait as proportional to this number and waits until the channel has been free for this amount of time. If the channel is still busy after the waiting period, the sender doubles the contention window and repeats the process.[2] The back-off process may also be initiated after the transmission of a single message in order to introduce some gaps between consecutive message transmissions. The back-off procedure is intended to desynchronize transmissions locally and reduce the number of message collisions.

Since successful data transmissions are immediately acknowledged by the node receiving the data, the sender can determine when an unsuccessful transmission occurred directly after the end of the transmission and initiate a retransmission. Retransmissions also follow the backoff procedure described above, up to a certain *retry limit*, after which the message is discarded permanently. It is then up to the higher levels of the protocol stack to decide which actions shall be taken. Assuming the channel is not excessively congested, due to its use of acknowledgments, the MAC protocol can be considered a method for reliable transmission of data messages from one node to another via the wireless channel.

The operation described above refers to *unicast* transmissions, which are identified by a (sender, receiver) pair of node identifiers (i.e., MAC addresses). The protocol also allows for *broadcast* transmissions, which are processed by all nodes that actually receive the transmission correctly. Since there is not a single receiver, broadcast messages are not acknowledged and therefore have to be considered less reliable. For this reason, they are typically transmitted at a lower rate compared to unicast messages. However, in some cases broad-

[2] Time is slotted and a node is only allowed to transmit at the beginning of a slot.

cast messages can be used to lower protocol overhead, since they effectively exploit the *wireless broadcast advantage* [39, 148]. By this term, we refer to the broadcast nature of wireless networks, which enables a node to reach a potentially large number of receivers by a single transmission. For an analysis of the effect of broadcast messages on MAC performance see [104].

As an optional extension to this protocol, nodes may reserve the channel for a unicast transmission by sending a *request-to-send* (RTS) control message prior to the data transmission to the receiving node. In the case that this message is received correctly, the receiver replies with a *clear-to-send* (CTS) message that tells the sender to start the data transmission. Both messages contain the size of the data packet and allow other nodes to estimate the time the channel will be busy due to this transmission. This scheme mitigates the hidden terminal problem, since nodes that receive either message will postpone their own transmission during that time. However, collisions of control messages can still occur even with RTS-CTS handshakes enabled.

One limitation of RTS-CTS handshakes is the fact that transmissions between nodes that are unable to communicate directly may still interfere. In order to receive a signal correctly, the so-called *signal-to-interference-and-noise* ratio (SINR) needs to be above a certain threshold value, which depends on properties of the radio device.[3] As a result, it is possible that the combined effect of several distant nodes transmitting may prevent a node from successfully sending a packet to its nearest neighbor. Since the resulting interference relation becomes very involved, one popular abstraction for the design of distributed algorithms is the assumption of a common *interference range*, which is typically larger than the transmission range. An interference model that is more realistic than the interference-range model yet simpler than the SINR model is employed by the network simulator NS2. For this and other interference models, see [21, 64].

2.2.2 Routing algorithms

To coordinate research and work out key problems, the Internet Engineering Task Force (IETF) Mobile Ad-hoc Network working group [62] was founded during the 1990's. Its purpose was declared to standardize Internet Protocol (IP) routing protocol functionality suitable for wireless routing applications in static as well as dynamic topologies. From their work resulted several Requests for Comments (RFCs) and Internet-Drafts that are concerned with the specification of routing protocols and related topics. Maybe the ones that had the greatest influence on the research community so far are the two experimental RFCs specifying the Ad Hoc On Demand Distance Vector (AODV) protocol [111] and the Dynamic Source Routing (DSR) protocol [70].

Both AODV and DSR belong to the class of *reactive* or *on-demand* protocols, which aim at a lower routing overhead by eliminating the continuous update of routing information that is performed by traditional algorithms for wired networks. Reactive protocols are therefore considered to be more appropriate for ad hoc networks with frequent node mobility. We will now briefly discuss the DSR protocol, since it forms the basis of routing algorithms developed later in this thesis. A more exhaustive overview of routing

[3] This is a simplification; for details refer to [64].

techniques in ad hoc networks can be found in [63].

Operation of DSR

The DSR [70] protocol is a *source-routing* protocol, which means that the source includes the whole route in every packet sent and intermediate nodes forward the packet to the next hop indicated in its header. This property eliminates the need of actively maintaining routing information at intermediate nodes and also enables an easy integration of multipath routing. Nodes keep routing information in their *route cache*, which can also contain information that was overheard from neighboring nodes. Source routing comes along with benefits, such as inherent loop freedom or the possibility for route shortening, as source nodes have knowledge about the entire route.

The basic DSR protocol consists of two operations: *route discovery* and *route maintenance*. If a source node wishes to send a packet to a destination to which it does not have a route in its route cache, it initiates the route discovery process by broadcasting a *route-request* (RREQ) message to its neighbors. Upon receiving the RREQ, nodes consult their route cache and can decide to send a *route-reply* (RREP) message back to the source. If they do not know a route to the destination, they append their own address to the list of nodes in the RREQ and forward the request further, until it eventually reaches the destination. The destination obtains a route from the source to itself by consulting the list of nodes that forwarded the RREQ. In the presence of bidirectional links, it can simply reverse this route and use it for sending a RREP message along this route to the source.

A sequence number ensures limited forwarding of RREQ's by intermediate nodes. In route discovery, a node only forwards a RREQ once. Since shorter routes require fewer hops, the first RREQ to reach the destination is likely to have taken a route that is (close to) minimal in terms of hop count. Therefore, DSR essentially routes packets along paths with shortest hop-distance. Although in principle multiple routes to the same destination may be contained in the route cache, e.g., by overhearing other routes, the nodes always pick the shortest route from the cache.

Basic route maintenance includes reliable packet transmissions from one hop to the next based on link-layer acknowledgments. Additionally, there are other operations initiated on-demand. If a source route breaks, the source is notified by an intermediate node detecting the break. The source can then choose to select an alternative route to the destination by consulting its route cache, or initiate a new route discovery. In the case that the intermediate node has a different route to the destination in its own cache, it can initiate *packet salvaging* and forward the packet using this alternative route.

Factors affecting DSR performance

The source-routing character of DSR may increase routing overhead compared to other algorithms that rely on routing information stored at intermediate nodes [21]. However, source routing offers the potential of interesting extensions to DSR, as discussed later in Chapter 5. One critical parameter of DSR in this context is the maximum source-route length.

Two factors causing packet loss in DSR are *packet collisions* and *interface queue* (IFQ) overflows. Packet collisions occur due to the wireless nature of

the transmission medium and can only be mitigated by underlying MAC layer protocols. Interface queue overflows, however, occur due to network congestion. The IFQ contains packets that are scheduled to be transmitted over the network interface and has a fixed maximum length. Hou and Tipper [61] observed that one of the main reasons for the decline in throughput for DSR is the overflow of the IFQ of congested nodes. In Chapter 5 we show that network throughput can be improved by choosing multiple routes for each source-destination pair that balance load over the network.

2.3 WIRELESS SENSOR NETWORKS

Although a wireless sensor network (WSN) can be regarded as a special type of ad hoc network, sensor networks typically distinguish themselves from ad hoc networks by having a predominant many-to-one type of communication pattern compared to many-to-many communication typically found in ad hoc networks. Further, sensor networks are usually considered to be more application-specific and more restricted in terms of available computing and communication resources. Once deployed in the area of interest, sensor network nodes have to self-organize and operate without outside control until the breakdown of the network, typically when battery power has run out.

After the initial network deployment, the location of nodes is normally considered fixed and the setup therefore static. The deployment itself may be considered random, e.g., when nodes are dropped from an airplane, or regular, when nodes are placed at predetermined locations. Also, nodes may or may not be aware of their location.

The flow of information within a sensor network is data-centric, and it can considered to be of low intensity, depending on the application [28]. The existence of one or several special *sink nodes* is assumed, which are responsible for collecting data and form a gateway to provide access to the network. The remaining nodes, however, are assumed to be homogeneous. Considering message routing towards a sink, nodes closer to the sink are expected to carry the burden of more message transmissions compared to nodes located further from the sink. Therefore, *data aggregation techniques*, which combine information contained in several messages into a single but possibly larger message, are highly desirable.

Since potential applications of sensor networks may require operation in a harsh environment, such as natural-disaster areas, node operation can be prone to errors and availability of nodes and the transmission medium may vary [2]. This effect is usually mitigated to a certain degree by the redundancy achieved by a dense deployment of nodes. For an overview of WSN see the survey by Akyildiz et al. [2] and for the effect of data aggregation on routing performance see the study by Krishnamachari et al. [78]. The deployment of a sensor network from a design problem point of view, including dimensioning of battery energy and node clustering for data aggregation, was addressed by Mhatre and Rosenberg [94]. For data-aggregation methods based on distributed optimization techniques see the work by Rabbat and Nowak [114]. Further applications and technological developments are discussed in the introduction by Culler et al. [36].

Over the years, a variety of algorithms have been proposed for WSN, among them MAC and routing algorithms similar to those for ad hoc networks, which have been optimized for energy-efficient operation. See, for example, the S-MAC protocol proposed by Ye et al. [152] and the references in [2]. We now briefly outline areas of interest different from MAC and routing algorithms that are particularly relevant to sensor networks.

2.3.1 Topology control

The transmission power that is required to communicate over a distance of d is typically modeled as

$$\beta d^{\alpha},$$

where β is a constant and α is the *path-loss exponent* that depends on the environment [39]. A typical choice for α satisfies $2 \leq \alpha \leq 4$. This simple model only applies when the receiver is located in the line-of-sight of the transmitter. It is a simplification, since it does not take into account certain properties of the wireless channel, such as reflections and other effects that occur when the signal interacts with the ground, obstacles or other parts of the environment.

Many topology control protocols assume that nodes can vary their transmission power and thereby influence which nodes are reachable when transmitting a message. The property that communication links between nodes can be actively influenced thus distinguishes wireless networks from tethered networks. In practice, however, there is a limit for the available transmission power. Besides having an influence on energy consumption, topology control may also facilitate *spatial reuse*, since transmissions that do not interfere can be performed simultaneously. Examples for topology control algorithms that adjust transmission powers can be found in [99, 117, 147].

Related problems that involve finding transmission-range assignments that induce communication graphs, which satisfy a desired property, have been studied by several researchers. The *range assignment problem* was introduced by Kirousis et al. [76] and asks for a minimum total power assignment that guarantees the communication graph to be strongly connected, when the nodes are located in a Euclidean space. This problem was shown to be NP-hard in both two and three dimensions [33, 76]. Later work has concentrated on approximation algorithms and various other graph properties that may be desirable in practice. For an overview see [87, 103, 120] and the references therein.

The term topology control is also used to refer to algorithms that do not modify transmission power or range, but compute a subgraph of the communication graph that satisfies certain properties, such as connectivity, but also minimizes a cost function. Since the resulting graph is then used for routing, sending messages along the edges of the graph should incur a low energy consumption. For sufficiently large path-loss exponents the total energy consumption of multihop paths can be lower compared to single-hop routing, as depicted in Figure 2.3. Problems that involve optimizing total energy consumption by routing have been studied extensively. Most distributed algorithms, for example, those proposed in [118, 147], make extensive assumptions on the path-loss model in order to establish optimality of a so-

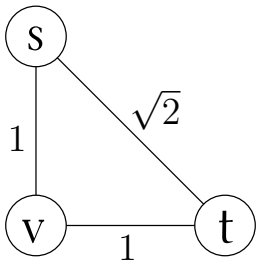


Figure 2.3: Although the direct path from s to t is shorter, for path-loss exponents $\alpha > 2$ transmitting a message via node v consumes less energy than a direct transmission.

lution based on local information at the nodes. For an overview of other applications and techniques for topology control see [83, 120, 146].

2.3.2 Lifetime maximization

The topology control methods described above preserve battery capacity at some nodes in the network. However, it is not immediately clear which transmission range or power assignment is the best in terms of the operation of the network. Ultimately, the goal of any method for energy-efficient operation of WSN is to extend its *lifetime*. Unfortunately, the notion of lifetime that is most appropriate depends on the application and may vary even during the time of its operation.

One popular definition of lifetime is the *time until the first node dies*, also called n-of-n lifetime in the extensive review by Dietrich and Dressler [38]. This notion of lifetime has been considered extensively, for example, by Chang and Tassiulas [26] who study a routing problem. The definition implicitly considers all nodes to be equally important and crucial to the operation of the network. This form of lifetime also avoids the complications to handle topology changes induced by failing nodes due to depletion of battery power. Although appealing in its simplicity, it cannot exploit the redundancy that may be present in real-world sensor networks [38].

Several alternative definitions of lifetime are possible. Most of these take into account additional constraints on network operation. Maybe the two most common ones are *network connectivity* and *network coverage*. Both of these constraints may either be guaranteed completely, or only to a certain extent, which is then a parameter of the lifetime. When considering network coverage one can distinguish between the spatial coverage of the deployment area and *target coverage*, which requires a finite set of special targets to be observed. Among both of these variations are possible, e.g., it may be required that the area or target is covered by several sensors at any time. Further variations include the constraint that sensors need to be able to detect intruders, which is also referred to as *barrier coverage* [38].

Independent of the notion of lifetime used, any algorithm for lifetime maximization needs to make some assumptions on an underlying model for energy consumption. It is generally accepted that for most applications data communication dominates data processing in terms of energy consumed [2].

Hence, in applications which require continuous updates transmitted to the sink, a node may use transmission-power control to conserve battery power and possibly extend the time after which its battery is depleted. If one applies this method to all nodes simultaneously, power control can be used for network lifetime maximization.

In other situations, however, the generated data stream may be of low intensity and most time is spent in idle states. In these cases it may be beneficial to exploit low-energy states that are most commonly supported by recent hardware [60]. In a low-energy state the radio is usually powered off and the node is unable to perform computing operations, except for monitoring a wake-up timer. However, various stages of low-energy states may exist. See the survey by Hempstead et al. [60] for an overview of different energy-efficient hardware architectures for sensor nodes. We will discuss next the general technique of power-saving by partitioning the network into active and inactive nodes.

2.3.3 Sleep scheduling

Algorithms for *sleep scheduling* assume that sensor-network nodes have several system states, which differ with respect to the amount of energy consumed while residing in the state [23]. In the simplest case, there are two states, *active* and *inactive*, and nodes may decide to switch between states to preserve battery capacity and thus extend lifetime. Therefore, sleep scheduling can be seen as an alternative to transmission power control and may be more applicable to sensor networks with a low duty cycle, i.e., when nodes are idle most of the time. This is particularly the case when the application requires the detection of rare events. Since the power consumption in an inactive state, when the CPU and radio are powered down, can be several orders of magnitude lower than in an active state [127], it is crucial that nodes reside in active states only when necessary.

Sleep scheduling relies on the existence of *redundant* nodes, where the definition of redundancy depends on the application. Several algorithms have been proposed for having a sensor network self-organize by choosing subsets of nodes to be active, which serve as a *backbone* for routing or providing sensor coverage. Many algorithms that were proposed in the literature can be regarded as heuristics that do not support performance guarantees compared to an optimal solution. For an overview see, for example, [8, 27, 38, 103].

A formal model of the sleep-scheduling problem needs to capture which nodes can be inactive under the condition that other nodes are active. A relatively simple yet powerful model assumes a pairwise redundancy relationship between sensor nodes for data collection. In the resulting *redundancy graph* [45] adjacent nodes represent sensors that can measure the same data. This model is applicable when node density is relatively large, so that within the vicinity of a node there exist several other nodes which are able to collect the same data.

Other notions of redundancy are also possible. Cărbunar et al. [35] assume a geometric setting where nodes have a fixed sensing radius and consider a node redundant when its sensing area can be completely covered by

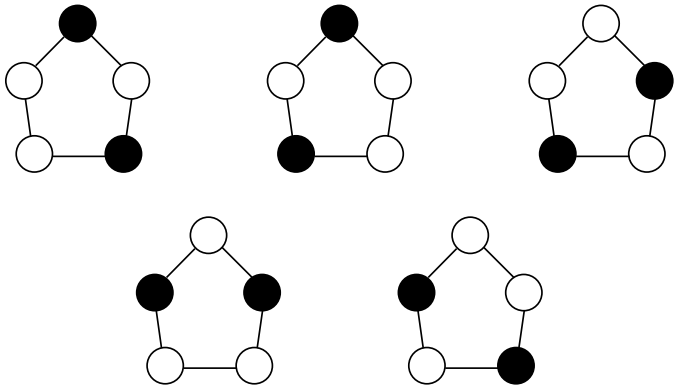


Figure 2.4: All possible dominating sets with two nodes for a five-node cycle; the domatic number is two, while the optimal fractional domatic partition achieves a lifetime of $5/2$ by assigning an activation time of $1/2$ to all of the five dominating sets, assuming unit capacities.

active neighboring nodes. Cao et al. [23] propose an algorithm that guarantees only partial coverage but ensures that every persistent event is detected within finite time.

If complete coverage is required and when backbone connectivity is not a concern, e.g., because the data generation and collection occur on different time scales, one can consider the network operational as long as each inactive node has an active neighbor in the redundancy graph at all times. In graph-theoretic terms, we are left with the problem of finding dominating sets in the redundancy graph and assigning them durations so that the total duration is maximized while satisfying node-battery constraints. This notion of sleep scheduling implicitly assumes the presence of a global clock to activate sets of nodes in turn for executing the schedule. However, it is usually sufficient that nodes are loosely synchronized, so that the complete system can still be considered asynchronous.

An important special case of the problem assumes that nodes have uniform battery capacities and a node can only be a member of at most one dominating set during its lifetime. The problem for pairwise node redundancies then reduces to the problem of finding the maximum number of disjoint dominating sets in a graph, which is also known as its *domatic number* [42]. In Chapter 8, we consider a version of this problem in which dominating sets can be assigned arbitrary activation times, as long as node battery constraints are satisfied. Considering fractional activity times facilitates the application of linear-programming approximation techniques and also allows for a longer n-of-n lifetime in some networks. Figure 2.4 shows an example where the integral problem has maximum lifetime two, while a lifetime of $5/2$ is achievable when nodes are allowed to switch between dominating sets.

2.4 NETWORK SIMULATORS

A number of commercial and open-source simulators have been developed which allow the simulation of multihop wireless networks. Maybe the most

important feature of a simulator from a research perspective is the property of being easily extensible and modifiable. All simulations that are part of this thesis were performed using the NS2 simulator [93]. The NS2 simulator is the result of a collaboration between several research institutions that started in the early 1990s, including University of California in Berkeley. Later extensions to NS2 that enabled the simulator to model wireless communication are due to a project at Carnegie Mellon University [21]. Over the years, a large community of developers has contributed code to the NS2 project and the simulator has evolved significantly [93].

Since NS2 is developed as an open-source project, it allows for simple extensions and modifications to all parts of the system. Additionally, the existence of a large user base and good documentation are beneficial. The NS2 simulator has been used extensively by a large number of research groups for validation of network protocols, so in this sense it is an established tool for baseline comparison. Furthermore, the simulator provides implementations of a large number of protocols, e.g., MAC layer and routing protocols such as DSR. Besides network protocols, the simulator also contains physical models for wireless signal propagation at various levels of abstraction. The features of the simulator further include support for node mobility and a graphical tool for visualizing simulation runs. All data is readily accessible and logging information can be collected at all layers and is easily processed by external tools of one's own choosing.

The simulator itself is based on discrete events, which also model packet transmission and reception. In this sense, when a node transmits a packet, it informs the global scheduling process, which then creates a reception event for the receiving node after a certain processing and transmission delay (assuming the packet was received correctly by the MAC protocol). Based on this model, the developer of a new protocol can usually rely on implementations of lower level protocols and only has to implement functions that are required to process packets introduced by the new protocol. The event scheduler also provides timers and supports user-defined events.

The code of the simulator is written in two programming languages. The parts that require an efficient, low-level implementation, are written in C++. The higher level of the code is written in OTcl, which is an object-oriented extension to the Tcl scripting language. Typically, the same object has a representation on both sides, where the interaction between the two code domains is enabled by an interface provided by a C++ library. This separation facilitates the goal to use a simple scripting language to write simulation runs that contain parameters that may change frequently. Also protocols may be partly specified in form of an OTcl script, which typically would contain parameters that change often. The remaining part of a protocol, which changes less frequently and requires inspecting single packets and therefore has higher demands for an efficient implementation, would then be implemented in C++. Since simulation runs are written in the form of OTcl scripts, the NS2 binary can itself be seen as an interpreter for this scripting language. As of 2010, the simulator is currently undergoing a major revision and NS2 will eventually be replaced by the – to a large extent rewritten – NS3 [102].

The best evaluation for any proposed protocol is indisputably a reference

implementation in combination with an actual deployment and a careful and extensive trial period in an appropriate environment. However, it is also clear that this method may not always be available due to the lack of resources or expertise in network deployments. We consider network simulations a reasonable compromise between an actual deployment and a purely analytic evaluation, since a simulator provides a relatively realistic abstraction of the computing environment. In fact, the simulations of the algorithms presented in this thesis were started at an early stage of the development of the algorithms and have usually led to additional insights. Concurrent events, reordered message transmissions, packet loss, and the occurrence of rare unexpected events are only some examples for aspects of an algorithm that are hard to foresee without performing simulations. Additionally, the controlled environment that enables the algorithm designer to run repeatable trials can be used to isolate effects caused by parameter variations.

3 LINEAR AND CONVEX PROGRAMMING

This chapter outlines topics from two related areas of mathematical modeling and optimization theory relevant to this thesis: linear and convex programming. The intention is not to give a concise overview but rather to fix notation to be used later. For an overview of linear programming see [137] and for convex optimization in particular see [13].

Linear programming has been a tool in operations research since the early years of computer science, e.g., for modeling planning and decision problems. Its popularity can be partly explained by the advent of computers and algorithmic breakthroughs. The simplex algorithm, proposed by Dantzig in the 1940's, is still popular today. Newer techniques, such as interior point methods, combine theoretical and empirical properties that have led to their widespread application. Convex optimization can be seen as a generalization of linear programming and inherits some of its beneficial properties, such as the global optimality of local optima. Relatively recent advances, e.g., subgradient methods, have led to new algorithms for network optimization.

For some of the problems discussed here, such as linear and convex programs with only fractional variables, there exist efficient algorithms that compute optimal solutions in polynomial time if a solver knows the complete problem instance. The purpose of discussing variations of these methods is to build a basis for distributed algorithms presented in later chapters. Other problems, e.g., the set-cover problem, are even hard to solve to optimality in a centralized setting so that we focus on approximation algorithms.

From both areas we discuss several algorithmic approaches that share a common similarity: the algorithms solve a pair of primal and dual problems concurrently. From this property, not only approximation guarantees can be obtained, but also usually the constraints in the dual are more local, in the sense that few variables compete for a shared resource. This *locality by duality* principle will be used later to develop distributed algorithms for multihop wireless networks. Techniques based on linear-programming duality have only been recently applied to obtain distributed approximation algorithms for combinatorial optimization problems, such as capacitated vertex cover and weighted matching problems [51, 106, 130].

3.1 APPROXIMATION ALGORITHMS

It is generally accepted that it is not possible to find optimal solutions to all instances of certain optimization problems efficiently, i.e., in polynomial time. The class of problems that is considered computationally hard includes general linear programming problems which contain variables that are required to take integral values. Therefore, the field of *approximation algorithms* [139] offers a promising direction. An approximation algorithm is not required to produce an optimal solution, but guarantees a certain quality of the resulting solution for *every instance* of the problem.

We follow conventional definitions [139]. Assume a given minimization problem and let I be an instance of this problem of size $|I|$. Let ϕ be a

function in the size of the instances, where $\phi(|I|) \geq 1$ for all instances I. We say that an algorithm A is a *factor ϕ approximation algorithm* for the problem if, for every instance I, A returns a solution that has an objective value of at most $\phi(|I|)$ times the objective value of an optimal solution. In this case we also say that A has an *approximation factor* ϕ for the given problem. An approximation factor is *tight*, if there is an instance that achieves this bound. Similarly, for $\phi(|I|) \leq 1$, we say that an algorithm A' is a factor ϕ approximation algorithm for a given maximization problem, if, for every instance I, A' returns a solution that has an objective value of at least $\phi(|I|)$ times the objective value of an optimal solution. Sometimes it is convenient to let ϕ be a function in certain parameters of the instance, rather than its size. Also, we may even allow ϕ to depend on the instance itself, which then results in an instance-dependent approximation guarantee.

Depending on ϕ, there are several other interesting special cases. When ϕ is a constant, we call the algorithm a *constant factor approximation algorithm*. For some problems one is able to find approximation algorithms whose approximation factor depends on a parameter $\epsilon > 0$. The runtime of the algorithm is required to be polynomial when ϵ is fixed. We say that such an algorithm A is a *polynomial-time approximation scheme* (PTAS) for a given minimization problem if, for any instance I, it returns a solution with objective value of at most $(1+\epsilon)$ times the objective value of an optimal solution. If the algorithm takes an instance of a maximization problem as input, then we require that the objective value of a solution is at least $(1-\epsilon)$ times the objective value of an optimal solution.

Assume that the given problem is a minimization problem. Establishing an approximation factor for a given algorithm typically involves finding an upper bound for the objective value obtained by the algorithm and a lower bound for the respective objective value of an optimal solution. Both bounds typically depend on the instance of the problem. The algorithms discussed in Section 3.2.2 solve the problem of lower bounding optimal solutions by generating solutions to the dual problem of a linear programming formulation of the problem. For a general overview of approximation algorithms for combinatorial optimization problems see Vazirani's textbook [139].

3.2 LINEAR AND INTEGER LINEAR PROGRAMMING

This section introduces basic models and notation and then presents algorithmic ideas used later in this thesis. Since most linear optimization problems in this thesis are naturally formulated as minimization problems, we use this representation as our primary choice.

A *linear program* (LP) is an optimization problem of the form

$$\begin{aligned}
\text{minimize} \quad & f(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j & \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, \ldots, m \\
& x_j \geq 0, & j = 1, \ldots, n,
\end{aligned}$$

where the x_j are variables and the coefficients a_{ij}, b_i and c_j are constants. The function f is called the *objective function*. We refer to the representation above as an LP in *canonical form*. We say that for a given solution an inequality constraint is *tight* if the constraint holds with equality.

Note that instead of the formulation above, one can also write the LP using $m \times n$ matrix $A = (a_{ij})$, $m \times 1$ vector $b = (b_i)$ and $n \times 1$ vector $c = (c_j)$. Further transformations are possible. One can transform the minimization into a maximization problem, replace inequalities by equalities by introducing a *slack variable* for each inequality constraint, etc. We shall prefer the formulation above. Sometimes the problem also contains integrality constraints for the x_j, i.e., special constraints that require $x_j \in \mathbb{N}_0$. In this case we refer to the problem as an *integer linear program* (ILP). If only some variables are required to be integral, we use the term *mixed integer program*.

We call a value assignment for the x_j a *solution* to the problem. A solution that satisfies all constraints is called *feasible* and *infeasible* otherwise. A problem that has no feasible solutions is called infeasible. Unless stated otherwise, we always assume that there exists some feasible solution. A problem can be *unbounded*, if it has feasible solutions that achieve an arbitrarily small objective value. A solution that attains the minimum value for the objective function is called an *optimal solution* or just *optimal*. We usually denote such a solution by x^*, where $x^* = (x_1^*, \ldots, x_n^*)$. The set of all optimal solutions is denoted by X^* and is non-empty for any problem that is not infeasible or unbounded. We sometimes denote the value of the objective function at an optimal solution by OPT, i.e., $\text{OPT} = f(x^*)$.

Depending on the range of possible values for the a_{ij}, b_i and c_j, specific classes of LP's are possible. An important class is the class of *covering* LP's (or *covering* ILP's), for which $a_{ij}, b_i, c_j \geq 0$ holds.

3.2.1 Linear programming duality

To every linear programming problem there exists a so-called *dual problem*. The dual problem can be obtained from the original problem, which is then called *primal problem*, by a technical transformation. Figure 3.1 shows a pair of primal and dual problems, where the primal is in canonical form. Assuming the primal problem is a covering LP, the dual in Figure 3.1 takes the form of a *packing* LP. If one applies the same transformation to the dual problem, that is, if one forms the dual of the dual problem, then one obtains again the primal by using appropriate transformations of the LP. We say that a solution $\hat{x}$ to the primal problem is *primal feasible* if it satisfies all constraints in the primal problem. Similarly, we call a solution $\hat{y}$ to the dual problem *dual feasible* if it satisfies all constraints in the dual problem. We define the notions of primal (respectively dual) optimality accordingly.

We summarize two important theorems of linear programming that will be used later to derive approximation algorithms. The following theorems are known as the *weak duality theorem* and *strong duality theorem*, respectively.

Theorem 1 (LP weak duality). *Any pair $(\hat{x}, \hat{y})$ of primal and dual feasible solutions satisfies*

$$\sum_{j=1}^{n} c_j \hat{x}_j \geq \sum_{i=1}^{m} b_i \hat{y}_i.$$

$$
\begin{array}{lll}
\textbf{Primal:} & \text{minimize} & \displaystyle\sum_{j=1}^{n} c_j x_j \\
& \text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad i = 1, \ldots, m \\
& & x_j \geq 0, \qquad j = 1, \ldots, n \\
\textbf{Dual:} & \text{maximize} & \displaystyle\sum_{i=1}^{m} b_i y_i \\
& \text{subject to} & \displaystyle\sum_{i=1}^{m} a_{ij} y_i \leq c_j, \qquad j = 1, \ldots, n \\
& & y_i \geq 0, \qquad i = 1, \ldots, m
\end{array}
$$

Figure 3.1: A pair of a primal and dual linear programs.

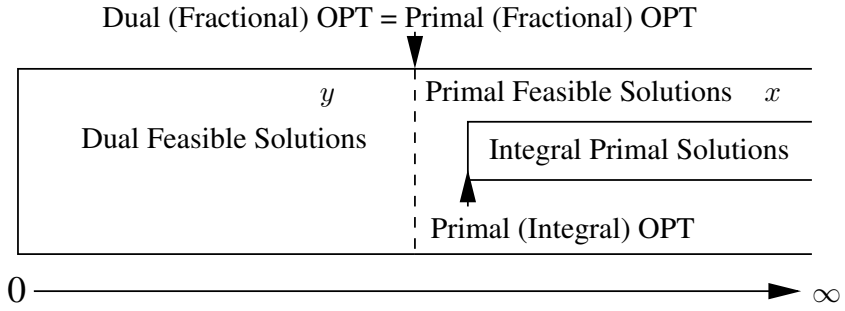


Figure 3.2: Space of primal and dual solutions for a pair of covering/packing linear programs.

Theorem 2 (LP strong duality)**.** *Any pair* (x^*, y^*) *of primal and dual optimal solutions satisfies*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Suppose one plots the values of the primal and dual objective functions over the range $[0, \infty]$ for a pair of covering/packing LPs. Figure 3.2 depicts the space of primal and dual feasible solutions over this range of their objective values.

3.2.2 Primal-dual algorithms

We now describe the main algorithmic idea that has been used to develop approximation algorithms for various NP-hard optimization problems, distributed and centralized. Suppose we have an ILP minimization problem in the form of the primal problem in Figure 3.1 with additional integrality constraints $x_j \in X_j \subseteq \mathbb{N}_0 \ \forall j$. We call the primal problem in Figure 3.1 obtained by disregarding the integrality constraints the *linear relaxation* (or fractional version) of the original problem.

Assume that for some instance I one is able to construct some feasible solution $\hat{x}$ that also satisfies the integrality constraints. Suppose that for the same instance I one is also able to construct some feasible solution $\hat{y}$ to the dual problem (obtained as the dual of the linear relaxation) such that

$$\sum_{j=1}^{n} c_j \hat{x}_j \leq \phi(I) \sum_{i=1}^{m} b_i \hat{y}_i, \tag{3.1}$$

where $\phi(I) \geq 1$. Note that (3.1) is required to hold by design of the algorithm. Then from Figure 3.2 it is clear that the solution $\hat{x}$ achieves an objective value of at most $\phi(I)$ times the optimum. This follows from weak duality and the fact that an optimal fractional solution can be used to obtain a lower bound for the objective value of an optimal integral solution. If one is able to show that the $\phi(I)$ in (3.1) only depends on the size $|I|$ of the instance I, then it follows that the algorithm is in fact a factor ϕ approximation algorithm for this problem. We call this type of algorithms that maintain a pair of primal and dual solutions *primal-dual* algorithms. Note that even if ϕ depends on I, one obtains an instance-dependent approximation guarantee, which may be valuable in practice. In the following, we discuss two algorithms that belong to this class and will be used later to obtain distributed approximation algorithms for problems in wireless sensor networks.

In the context of linear relaxations it is important to consider the *integrality gap* [139] of a problem. The integrality gap for a minimization ILP is defined as the largest ratio of the objective value of optimal integral and fractional solutions over all instances. Note that if an optimal solution to the linear relaxation of a given problem always satisfies the integrality constraints of the original problem, the integrality gap is 1. We will now show a simple but important fact, namely that the integrality gap bounds the approximation factor that is achievable by any primal-dual algorithm.

Theorem 3. *For any primal-dual algorithm with approximation factor ϕ for a given minimization problem*

$$\phi \geq \sup_{I} \frac{OPT_{int}(I)}{OPT_{frac}(I)},$$

where $OPT_{int}(I)$ is the objective value of an optimal integral solution and $OPT_{frac}(I)$ is the corresponding value for its relaxation.

Proof. Let I be any instance of the problem and $(\hat{x}, \hat{y})$ the solution obtained by the ϕ-approximation algorithm. It holds that

$$\mathrm{OPT}_{\mathrm{int}}(I) \leq \sum_{j=1}^{n} c_j \hat{x}_j \leq \phi \sum_{i=1}^{m} b_i \hat{y}_i \leq \phi \, \mathrm{OPT}_{\mathrm{frac}}(I),$$

from which the claim follows since I was arbitrary. □

Set-cover approximation

The first algorithm we describe that fits into the primal-dual framework is an approximation algorithm for the set-cover problem. The presentation of

the problem and the analysis of the algorithm are based on Chapter 13 in Vazirani's book on approximation algorithms [139]. In Chapter 8 we use this algorithm to develop an efficient distributed algorithm for the minimum weight dominating set problem.

The weighted set-cover problem can be described as follows. One is given a set U, called the *universe of elements*, a collection of subsets $\mathcal{S} \subseteq 2^U$ and a weight function $w : \mathcal{S} \mapsto \mathbb{R}_{>0}$. The problem is to find a minimum weight set $C \subseteq \mathcal{S}$ that *covers* every *element* $e \in U$, i.e., $U = \bigcup_{S \in C} S$. The weight of a cover C is defined as $w(C) = \sum_{S \in C} w(S)$. One can formulate this problem as an ILP by introducing a binary variable x_S for each $S \in \mathcal{S}$ that when set to 1 indicates that S is chosen to be part of the cover and otherwise is set to 0. We obtain the following problem.

$$\begin{aligned}
\textbf{IWSC:} \quad \text{minimize} \quad & \sum_{S \in \mathcal{S}} w(S) x_S \\
\text{subject to} \quad & \sum_{S \ni e} x_S \geq 1, \qquad \forall e \in U \qquad (3.2) \\
& x_S \geq 0, \qquad \forall S \in \mathcal{S} \\
& x_S \in \{0, 1\}, \qquad \forall S \in \mathcal{S} \qquad (3.3)
\end{aligned}$$

Note that (3.2) formulates the constraint that each element must be covered by some set in the cover. Now consider the linear relaxation of problem IWSC, which we denote by FWSC. Correspondingly, one obtains the dual of problem FWSC, which we denote as DWSC, as follows.

$$\begin{aligned}
\textbf{DWSC:} \quad \text{maximize} \quad & \sum_{e \in U} y_e \\
\text{subject to} \quad & \sum_{e \in S} y_e \leq w(S), \qquad \forall S \in \mathcal{S} \qquad (3.4) \\
& y_e \geq 0, \qquad \forall e \in U \qquad (3.5)
\end{aligned}$$

The dual variables in problem DWSC have an intuitive interpretation as prices an element has to pay in order to be covered by a set. The objective value of an optimal dual solution then corresponds to the maximum profit that can be earned by the sets. We now discuss a simple primal-dual algorithm formulated as Algorithm PDSC. The algorithm maintains the pair of primal and dual solutions $(\hat{x}, \hat{y})$ and guarantees that the $\hat{x}$ are always integral.

Both primal and dual solutions are initialized to 0. Then the algorithm starts raising all dual variables at unit rate until one of the dual constraints (3.4) becomes tight, say for set S. At this point the set S is added to the cover set C and all dual variables y_e for the elements in S become *frozen*, which means that their values will not be changed after this point. If there is more than one set whose constraint becomes tight, one chooses one of these arbitrarily. The set F in Algorithm PDSC contains the elements of the frozen variables. The algorithm continues to raise the value for all variables that are not frozen and sets enter the cover C until the universe U is entirely covered by the sets in C. One can show that this algorithm is in fact a continuous-time version of Chvátal's greedy algorithm [31, 139].

It is easy to see that upon termination, the set C is a valid cover and that $\sum_{S \in \mathcal{S}} w(S) x_S = \sum_{e \in U} y_e$, since all sets that enter the cover are *fully paid*

Algorithm PDSC: Primal-dual algorithm for weighted set cover

initially:
$\hat{x}_s \leftarrow 0 \quad \forall S \in \mathcal{S}; \quad \hat{y}_e \leftarrow 0 \quad \forall e \in U;$
$C \leftarrow \emptyset; \quad F \leftarrow \emptyset;$
while $U \neq \bigcup_{S \in C} S$ **do**
  while $\sum_{e \in U \setminus F} y_e < w(S) \quad \forall S \in \mathcal{S} \setminus C$ **do**
    raise $\hat{y}_e$ for all $e \in U \setminus F$ at unit rate;
  end
  choose $S \in \mathcal{S} \setminus C$ such that $\sum_{e \in U \setminus F} y_e = w(S)$;
  $\hat{x}_S \leftarrow 1; C \leftarrow C \cup \{S\}; F \leftarrow F \cup S;$
end

for and no element pays for being covered by two different sets. We now establish the approximation guarantee of $\mathcal{O}(\ln \Delta)$, where Δ is the maximum cardinality of any set in $\mathcal{S}$, by the technique of *dual fitting* [65].

Theorem 4. *Algorithm* PDSC *is a factor* $\mathcal{O}(\ln \Delta)$ *approximation algorithm, where* Δ *is the maximum cardinality of any set in* $\mathcal{S}$.

Proof. We will actually prove that by scaling the dual solution $\hat{y}$ down by a factor of H_Δ, where H_n is the n-th harmonic number, one obtains a dual feasible solution. From weak duality, $H_n = \mathcal{O}(\ln n)$ and by noticing that $\sum_{S \in \mathcal{S}} w(S) x_S = \sum_{e \in U} y_e$ before the scaling the result then follows.

Consider any set $S \in \mathcal{S}$ and sort its elements by the time they were covered from latest to earliest, so $S = \{e_1, \ldots, e_{|S|}\}$, $e_{|S|}$ was covered first and e_1 was covered last. Then for all $e_i \in S$ we have that $\hat{y}_{e_i} \leq w(S)/i$. This can be seen as follows. If e_i was covered by adding S to the cover, we have that $\hat{y}_{e_i} = w(S)/|S \setminus F|$, where F is the set of frozen variables (corresponding to covered elements) before S was added to C, and $|S \setminus F| \geq i$. Assume now e_i was covered by some other set S'. Then it must be that $\hat{y}_{e_i} = w(S')/|S' \setminus F| \leq w(S)/|S \setminus F|$, where F is the set of frozen variables before S' was added to C, since otherwise S would have been added to C before S'. So we have that

$$\sum_{e_i \in S} \hat{y}_{e_i} \leq \sum_{e_i \in S} \frac{w(S)}{i} \leq H_\Delta w(S).$$

□

Maximum multicommodity flow

We now discuss an algorithm by Garg and Könemann [48] that is best explained in the setting of the maximum multicommodity flow problem. Since the algorithm can be adopted to a variety of different problems, including general packing and covering problems, we will describe it in detail. The same algorithmic idea will be used in Chapter 8 to develop a distributed algorithm for a lifetime maximizing problem in sensor networks.

The maximum multicommodity flow (MMF) problem can be formulated as follows. Given a directed (or undirected) graph $G = (V, E)$ with edge

capacities $c : E \mapsto \mathbb{R}^+$ and N distinct source-destination pairs $(s_1, t_1), \ldots, (s_N, t_N)$, the problem asks for the maximum total flow that can be sent from the source to the destination nodes without violating the edge capacities. This problem can be used to model a variety of routing problems. Note that instead of the edge capacities, it is also possible to associate capacities with the nodes. Although this problem can be solved to optimality in polynomial time, there have been efforts to develop approximation algorithms that for larger instances may compute good solutions relatively quickly. Note that the problem that asks for integral flows is computationally more complex than the fractional case.

The problem can be modeled as follows. Associate a flow commodity with each of the pairs. Denote the set of all paths from source s_n to target t_n by $\mathcal{P}_n$ and let $\mathcal{P}$ be the set of all paths. By introducing a variable x_p for the flow on path p for all $p \in \mathcal{P}$, the problem can be formulated as follows.

$$\begin{array}{lll} \textbf{MMF:} \quad \text{maximize} & \displaystyle\sum_{p \in \mathcal{P}} x_p & \\ \text{subject to} & \displaystyle\sum_{p \ni e} x_p \leq c(e), & \forall e \in E \\ & x_p \geq 0, & \forall p \in \mathcal{P} \end{array} \tag{3.6}$$

The sum in (3.6) runs over all paths containing the edge e and represents the capacity constraint for that particular edge. Note that the MMF problem as formulated above has a potentially exponential number of variables. Instead of this formulation using *path flows*, a more common formulation is an LP that has one variable for the flow of each commodity over that edge, leading to a polynomial number of variables. However, when one compares such a formulation with *edge flows* to a path-flow representation, the latter appears more natural when considering distributed algorithms in communication networks.

Introduce a dual variable y_e for each edge $e \in E$. Then the dual problem can be formulated as follows.

$$\begin{array}{lll} \textbf{DMMF:} \quad \text{minimize} & \displaystyle\sum_{e \in E} c(e)\, y_e & \\ \text{subject to} & \displaystyle\sum_{e \in p} y_e \geq 1, & \forall p \in \mathcal{P} \\ & y_e \geq 0, & \forall e \in E \end{array} \tag{3.7}$$

A solution to the dual problem can be interpreted as an assignment of lengths (or weights) to the edges. The length $l(p)$ of a path p is then given as $l(p) = \sum_{e \in p} y_e$. A solution is feasible if and only if the shortest path between any source-destination pair has length at least 1. Define $\alpha(y)$ to be the length of the shortest path between any pair for a given length assignment y. The Garg-Könemann (GK) algorithm for the MMF problem is formulated as Algorithm GKMMF.

Algorithm GKMMF calls a shortest-path oracle in each iteration and augments the flow along the path by a value that is equal to the minimum edge capacity along the path. Then the value for the weight of the edges that lie

Algorithm GKMMF: Garg-Könemann algorithm for maximum multicommodity flow

initially:
$\hat{x}_p \leftarrow 0 \quad \forall p \in \mathcal{P}; \quad \hat{y}_e \leftarrow \delta \quad \forall e \in E;$
repeat
  use oracle to compute a path p^* with $l(p^*) = \alpha(\hat{y})$;
  let c be the minimum capacity on p^*, i.e., $c = \min_{e \in p^*} c(e)$;
  route c units of commodity over path p^*;
  for $e \in p^*$ **do**
    $\hat{y}_e \leftarrow (1+\epsilon)\hat{y}_e$;
  end
until $l(p^*) \geq 1$;

on the path is multiplied by a factor of $(1+\epsilon)$, where ϵ is a parameter of the algorithm. The initial edge weight δ is discussed later. Note that the path p^* in Algorithm GKMMF minimizes path length over all paths and all source-destination pairs.

Clearly, upon termination, the dual solution is feasible. The primal solution, on the other hand, may be infeasible due to violation of the edge capacity constraints. However, following the technique in [48], one can show that by scaling down the flow on all paths by a factor of $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ a primal feasible solution is obtained. Moreover, the ratio of the dual optimum and the primal feasible solution thus obtained can be upper bounded. For a choice of $\delta = (1+\epsilon)((1+\epsilon)L)^{-1/\epsilon}$, where L is the maximum number of edges on any path, one then obtains the following result.

Theorem 5 ([48]). *Algorithm* GKMMF *computes a multicommodity flow of at least* $(1-\epsilon)^2$ *times the optimum, where* $0 < \epsilon < 1$. *The algorithm terminates after at most* $\lceil (|E|/\epsilon) \log_{1+\epsilon} L \rceil$ *iterations.*

Instead of solving the shortest-path subproblem in each iteration exactly, it is possible to use an approximation oracle with approximation factor $\phi > 1$. The approximation guarantee then worsens by this multiplicative factor, i.e., the flow is guaranteed to achieve at least $(1-\epsilon)^2/\phi$ times the optimal value. This technique was first applied by Fleischer [43] to the MMF problem and later by Tsaggouris and Zaroliagis [134] to a problem from transportation planning. See also [134] for modifications to the proofs outlined above.

Besides being an efficient method for computing solutions to packing-covering and more general problems, the GK scheme is often a viable method for dealing with an exponential number of primal variables. It is sometimes the case that the dual problem can be solved easier, or at least approximated to a certain factor. We will apply the same technique in Chapter 8.

Minimum maximum congestion

In communication networks with link capacities, heavy load on some links in the network usually affects the performance negatively and is therefore undesirable. Hence, besides achieving a maximum end-to-end throughput as in problem MMF, one is also interested in balancing the load over the network.

Consider a similar formulation as in problem MMF, where one additionally is given a demand d_n for each source-destination pair (s_n, t_n). We formulate the minimum maximum congestion (MMC) problem as follows.

$$\begin{aligned}
\textbf{MMC:} \quad & \text{minimize} && \lambda && \\
& \text{subject to} && \sum_{p \ni e} x_p \leq \lambda c(e), && \forall e \in E \qquad (3.8)\\
& && \sum_{p \in \mathcal{P}_n} x_p = d_n, && \forall n \qquad (3.9)\\
& && x_p \geq 0, && \forall p \in \mathcal{P}
\end{aligned}$$

In this formulation, the variable λ bounds the *congestion* of an edge e, which is the total flow that passes e divided by its capacity. Minimizing λ therefore corresponds to minimizing the maximum congestion in the network subject to the constraint of satisfying the request for the routable demand for each source-destination pair. A problem that is closely related to MMC is the maximum concurrent flow (MCF) problem, which can be formulated as follows.

$$\begin{aligned}
\textbf{MCF:} \quad & \text{maximize} && \theta && \\
& \text{subject to} && \sum_{p \ni e} x_p \leq c(e), && \forall e \in E \qquad (3.10)\\
& && \sum_{p \in \mathcal{P}_n} x_p \geq \theta d_n, && \forall n \qquad (3.11)\\
& && x_p \geq 0, && \forall p \in \mathcal{P}
\end{aligned}$$

It is easy to see that the two problems MMC and MCF are equivalent. Any feasible flow $\hat{x}_p$ with objective value $\hat{\lambda}$ for MMC can be scaled down by a factor of $\hat{\lambda}$ and results in a feasible solution to MCF. This scaled solution then has objective value $\hat{\theta} = 1/\hat{\lambda}$ and minimizing λ corresponds to maximizing θ.

Although, as for the multicommodity flow problem, the preceding problem can be solved to optimality efficiently using a different LP formulation with edge flows, several fast approximation algorithms have been proposed that exploit the structure of the problem using path flows. An algorithm that is very similar to the Garg-Könemann scheme presented as Algorithm GKMMF, is the algorithm by Young [154], which was originally proposed for general covering and packing problems. Algorithm YMMC is a reformulation of the algorithm applied to the MMC/MCF problem, as summarized by Bienstock in [16], for the case of unit edge capacities, i.e., $c(e) = 1 \ \forall e \in E$. It is easy to see that for the single commodity case with unit demands, the rules for updating the variables in Algorithm YMMC are in fact identical to the updates in Algorithm GKMMF.

The analysis in [16] shows that $I = \lceil \frac{4|E| \log |E|}{\epsilon^2} \rceil$ iterations are sufficient to obtain a solution to problem MMC with congestion $\hat{\lambda}$ that satisfies $\hat{\lambda} \leq (1+\epsilon)\lambda^*$, where λ^* is the minimum maximum congestion. One should note that Algorithm YMMC can also be extended to the case of arbitrary edge capacities.

Algorithm YMMC: Approximation algorithm for the MMC/MCF problem with unit capacities (see [16])

initially:
$\hat{x}_p \leftarrow 0 \quad \forall p \in \mathcal{P}; \quad \hat{y}_e \leftarrow 1 \quad \forall e \in E;$
for I iterations **do**
  $f_p \leftarrow 0 \quad \forall p \in \mathcal{P};$
  for each source-destination pair (s_n, t_n) **do**
    use oracle to compute a shortest path p from s_n to t_n using $\hat{y}_e$;
    $f_p \leftarrow$ path flow of d_n units of commodity over path p;
    $\hat{x}_p \leftarrow \hat{x}_p + f_p$;
  end
  for $e \in E$ **do** $\hat{y}_e \leftarrow (1 + \epsilon \sum_{p \ni e} f_p)\hat{y}_e$;
end
$\hat{x}_p \leftarrow \hat{x}_p / I \quad \forall p \in \mathcal{P}$

3.2.3 Network optimization based on LP relaxation

The technique of linear relaxation was used to establish approximation guarantees for primal-dual algorithms, as demonstrated for the set cover problem. LP relaxation can also be employed for solving an integer linear problem via the branch-and-bound search method. In this section we first describe how to model a routing problem with path constraints as a network flow problem, which is formulated as an ILP where all integer variables take binary values. We then show how this problem can be solved by a branch-and-bound algorithm that uses linear relaxation for pruning the search. Note, however, that the runtime of the algorithm can be exponential in the size of the instance.

The techniques discussed in this section were proposed by Caramia and Sgalambro [24] for solving instances of the *maximum concurrent k-splittable flow problem*. This problem is a variant of the MCF problem discussed previously, where the solution is additionally constrained to contain only a bounded number of non-zero flow paths. The problem was first addressed by Baier et al. [6]. In Chapter 6 we extend the model and method and obtain an algorithm for a routing problem within the context of ad hoc networks.

Maximum multicommodity flow with path constraints

Consider the version of the maximum multicommodity flow problem where each source and destination pair is only allowed to use a constant number K of paths to send flow from the source to the destination. Since any source-destination flow can be decomposed into at most $|E|$ paths and cycles [1], for large enough K, an optimal solution to this problem can be obtained by solving problem MMF (the same problem without path constraints) to optimality. The case with path constraints, however, is already strongly NP-hard for a single commodity and $K = 2$. Constant-factor approximation algorithms were proposed by Baier et al. [7].

The *K-paths constrained maximum multicommodity flow* (KMMF) problem is then formulated as follows. Denote by $P_n \subseteq \mathcal{P}_n$ the variable that contains the set of *selected* paths for the source-destination pair (s_n, t_n). Recall

that $\mathcal{P}_n$ denotes the set of all paths from source s_n to target t_n and that the set of all paths is defined by $\mathcal{P} = \bigcup_n \mathcal{P}_n$.

$$\begin{aligned}
\textbf{KMMF:} \quad \text{maximize} \quad & \sum_n \sum_{p \in P_n} x_p \\
\text{subject to} \quad & \sum_{p \ni e} x_p \le c(e), && \forall e \in E \quad (3.12) \\
& x_p \ge 0, && \forall p \in \mathcal{P} \\
& P_n \subseteq \mathcal{P}_n, && \forall 1 \le n \le N \\
& |P_n| \le K, && \forall 1 \le n \le N
\end{aligned}$$

Note that only flow over selected paths may contribute to the objective function. Further, since the number of path-flow variables can be exponential in the number of nodes, the problem may be intractable to be solved in this representation. However, for any fixed value of K, we use the technique from [24] and formulate an equivalent problem with variables for the flow over each edge.

For the edge-flow formulation of problem KMMF, we introduce additional variables δ_e^{nk}, which determine whether an edge e is available to route flow for the k-th path of pair (s_n, t_n) in a given solution. Let f_e^{nk} denote the amount of flow over edge e that originates from the k-th path from source s_n to destination t_n. Additionally, introduce variables y_n that correspond to the total flow from source s_n to t_n (so in the original problem, $y_n = \sum_{p \in P_n} x_p$). Problem KMMF is then formulated equivalently as the following mixed integer program.

$$\begin{aligned}
\textbf{KMMF-E} \quad \text{maximize} \quad & \sum_{n'=1}^{N} y_{n'} \\
\text{s.t.} \quad & y_n = \sum_{1 \le k' \le K} \sum_{e \in O(s_n)} f_e^{nk'} - \sum_{e \in I(s_n)} f_e^{nk'} && (3.13) \\
& -y_n = \sum_{1 \le k' \le K} \sum_{e \in O(t_n)} f_e^{nk'} - \sum_{e \in I(t_n)} f_e^{nk'} && (3.14) \\
& \sum_{e \in O(v)} f_e^{nk} - \sum_{e \in I(v)} f_e^{nk} = 0 \quad \forall v \in V \setminus \{s_n, t_n\} && (3.15) \\
& \sum_{1 \le n' \le N} \sum_{1 \le k' \le K} f_e^{n'k'} \le c(e) \quad \forall e \in E && (3.16) \\
& f_e^{nk} \le \delta_e^{nk} \cdot c(e) \quad \forall e \in E && (3.17) \\
& \sum_{e \in O(v)} \delta_e^{nk} \le 1, \quad \sum_{e \in I(v)} \delta_e^{nk} \le 1 \quad \forall v \in V && (3.18) \\
& \sum_{e \in O(v)} \delta_e^{nk} - \sum_{e \in I(v)} \delta_e^{nk} = 0 \quad \forall v \in V \setminus \{s_n, t_n\} && (3.19) \\
& \sum_{e \in I(s_n)} \delta_e^{nk} = 0, \quad \sum_{e \in O(t_n)} \delta_e^{nk} = 0 && (3.20) \\
& y_n \ge 0,\ f_e^{nk} \ge 0,\ \delta_e^{nk} \in \{0, 1\} \quad \forall e \in E,
\end{aligned}$$

where $1 \le n \le N$ and $1 \le k \le K$. Problem KMMF-E has several edge-flow variables for a single edge: each commodity corresponding to a single

source-destination pair consists of K subcommodities. For each commodity n and path index k, there is a network flow *layer* f^{nk}. A value of 1 for δ_e^{nk} indicates that edge e lies on the k-th path from s_n to t_n, i.e., that flow of the k-th layer of (s_n, t_n) may be routed over e. Constraint (3.13) requires that the total outgoing flow y_n of s_n equals the sum over all K flow layers, where $I(v)$ and $O(v)$ denote incoming and outgoing edges incident to v, respectively. Equation (3.14) states that the total incoming flow of t_n has to equal y_n, and (3.15) is the *flow-balance* constraint, which has to be satisfied for each flow layer, except at source and destination nodes. Equation (3.16) is the capacity constraint and (3.17) guarantees that only selected edges carry flow.

As the δ_e^{nk} indicate edges that form the k-th path of pair (s_n, t_n), they must satisfy certain constraints. Nodes on a path have at most one incoming or outgoing edge (constraint (3.18)). Also, any node other than the source and destination must have an outgoing edge if and only if it has an incoming edge (constraint (3.19)). Finally, sources must have no incoming edges and destinations no outgoing edges (constraint (3.20)).

It is easy to check that problems KMMF and KMMF-E are equivalent. One can obtain from a solution to KMMF-E a solution to KMMF by following the δ_e^{nk} variables with value 1 starting at the sources and setting the path rates according to the f_e^{nk}. Although a solution to KMMF-E may contain isolated cycles, these do not affect optimality or constraint satisfiability. Conversely, a solution to KMMF-E is obtained from a solution to KMMF by setting the δ_e^{nk} of edges e that are incident to consecutive nodes on the k-th path from s_n to t_n to 1 (ordering the paths in P_n arbitrarily) and setting all others to 0. One should note, however, that KMMF-E has $\mathcal{O}(KN|E|)$ variables, while the number of path-flow variables in KMMF can be exponential in $|V|$.

Branch-and-bound algorithm

The formulation above lends itself to a branch-and-bound (BNB) algorithm that operates on a binary search tree containing values for the δ_e^{nk}. The algorithm employs a bounding heuristic that solves the problem with relaxed integrality constraints for the δ_e^{nk} at intermediate nodes in the search tree, i.e., with $\delta_e^{nk} \in \{0, 1\}$ replaced by $0 \leq \delta_e^{nk} \leq 1$. To reduce the number of calls to the linear programming solver, the algorithm checks constraints (3.18)–(3.20) to prune branches leading to infeasible solutions.

Algorithm BBKMMF, which is essentially a MMF version of the MCF algorithm proposed in [24], shows in pseudocode form a branch-and-bound method for finding the optimal paths and flow allocations. At first, the integrality constraints for the δ_e^{nk} are removed and the corresponding labels l_e^{nk} are set to *variable*. The optimal solution for the linear relaxation is computed, and if all δ_e^{nk} are integral, the solver has found an optimal solution and terminates. Otherwise a δ_e^{nk} variable is chosen to branch on, the corresponding l_e^{nk} are set to *fixed*, and the two subproblems where the δ_e^{nk} chosen for branching is set to 0 and 1, respectively, are solved. If at any point in the search the optimal value for a relaxed problem is worse than the best solution with all δ variables integral seen so far, the search branch is pruned, since introducing additional integrality constraints to the relaxed problem cannot improve the objective value of an optimal solution. Algorithm BBKMMF employs the

Algorithm BBKMMF: Branch-and-Bound for problem KMMF-E (adapted from [24])

```
1  initially
2  |  for 1 ≤ n ≤ N, 1 ≤ k ≤ K, e ∈ E do l_e^nk ← variable;
3  |  best_value ← −∞;
4  |  current_bound ← compute_upper_bound(δ, l);
5  |  if is_integer_solution(δ) then
6  |  |  best_value ← current_bound;
7  |  |  best_solution ← δ;
8  |  else branch_and_bound(δ, l);
9
10 function branch_and_bound(δ, l)
11 |  choose e, n, k such that min{δ_e^nk, 1 − δ_e^nk} is maximized;
12 |  for val ∈ {0, 1} do
13 |  |  δ_e^nk ← val;
14 |  |  l_e^nk ← fixed;
15 |  |  if delta_feasible(δ, l) then
16 |  |  |  current_bound ← compute_upper_bound(δ, l);
17 |  |  |  if current_bound > best_value then
18 |  |  |  |  if is_integer_solution(δ) then
19 |  |  |  |  |  best_value ← current_bound;
20 |  |  |  |  |  best_solution ← δ;
21 |  |  |  |  else branch_and_bound(δ, l);
22 |  |  |  end
23 |  |  end
24 |  |  l_e^nk ← variable;
25 |  end
26
```

standard heuristic of choosing the δ_e^{nk} variable to branch on that is furthest away from being integral.

In Algorithm BBKMMF, delta_feasible(δ, l) checks for satisfiability of constraints (3.18)–(3.20), compute_upper_bound(δ, l) solves the relaxation of problem KMMF-E, and the check for integrality of δ_e^{nk} is performed by is_integer_solution(δ). In an implementation, several practical aspects need to be considered. Firstly, one can safely set the fractional value of δ_e^{nk} to 0, if the solver returns a zero value for f_e^{nk}. This rule can reduce the number of fractional δ_e^{nk} values for a given solution and does not break feasibility. Secondly, the values returned for the δ_e^{nk} are typically only close to optimal, depending on the convergence criterion of the solver. Consequently, one needs to tolerate a certain amount of non-integrality in the δ_e^{nk}.

3.3 CONVEX PROGRAMMING

Methods from convex optimization theory have been used recently to approach a wide range of problems, for example, problems in communica-

tion networks, including ad hoc [71, 108] and sensor networks [68, 114]. We first outline basic definitions and notation, which are close to the textbook by Boyd and Vandenberghe [19]. After the introduction, we briefly discuss some standard iterative methods for solving constrained convex optimization problems. Although other more efficient algorithms exist, the algorithms discussed in this section can be expressed by simple arithmetic operations, which, depending on the structure of the problem, lend themselves to a distributed implementation.

Recall the definition of a convex set and convex function.

Definition 2. *A set $C \subseteq \mathbb{R}^n$ is called* convex *if for any pair of elements x, $y \in C$, we have for any* convex combination

$$\theta x + (1 - \theta)y \in C,$$

where $0 \leq \theta \leq 1$.

Denote by $\mathrm{dom} f$ the *domain* of a function f, i.e., the set of values for which it is defined.

Definition 3. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called* convex *if its domain $\mathrm{dom} f$ is convex and if for all pairs x, $y \in \mathrm{dom} f$, and θ with $0 \leq \theta \leq 1$, we have*

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta) f(y).$$

The function f is called strictly convex *if $x \neq y$ and $0 < \theta < 1$ implies that the strict inequality holds.*

Definition 4. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called* concave *if $-f$ is convex and* strictly concave *if $-f$ is strictly convex.*

In the following we note two necessary and sufficient conditions for convexity, which are used frequently to show convexity of functions.

Proposition 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a partially differentiable function (i.e., the gradient ∇f of f exists at any point in its domain $\mathrm{dom} f$, which is an open set).*

a) f is convex if and only if $\mathrm{dom} f$ is convex and

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{3.21}$$

holds for all x, $y \in \mathrm{dom} f$.

b) f is strictly convex *if and only if the strict inequality holds for all $x, y \in \mathrm{dom} f$ with $x \neq y$.*

Using $\leq$ instead of $\geq$ yields the corresponding result for concave functions.

Note that from (3.21) follows the important property that for differentiable convex functions a local minimum is always a global minimum. This property, however, also holds for general convex functions (see, for example, [19]). Correspondingly, every local maximum of a concave function is always a global maximum.

Proposition 2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function (i.e., its* Hessian $\nabla^2 f$ *exists at each point in dom$f$, which is open).*

a) f is convex if and only if dom$f$ is convex and its Hessian is positive semidefinite:

$$\nabla^2 f(x) \succeq 0$$

holds for all $x \in$ dom$f$.

b) f is strictly convex *if its Hessian is positive definite:*

$$\nabla^2 f(x) \succ 0$$

for all $x \in$ dom$f$. Note that this condition is only sufficient, not necessary.

Choosing $\preceq$ instead of $\succeq$ yields the corresponding conditions for concave functions.

Since the problems in this thesis that fall into the category of convex programming are naturally formulated as maximization problems with a concave objective function, we now use a different presentation compared to the section on linear programming. Consequently, the exposition of the theorems and methods described in this section differs from the standard literature but can be obtained by the simple transformation from the minimization of a convex objective function to the maximization of a concave objective function.

A *convex program* in canonical form is formulated as

$$\begin{array}{lll} \text{maximize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \ldots, m \\ & h_i(x) = 0, & i = 1, \ldots, p, \end{array}$$

with variable $x \in \mathbb{R}^n$, nonempty domain $\mathcal{D} = \bigcap_{i=0}^{m} \text{dom} f_i \cap \bigcap_{i=1}^{p} \text{dom} h_i \subseteq \mathbb{R}^n$. The objective function f_0 is required to be concave, the inequality constraint functions f_i, $i > 0$, are assumed to be convex, and the equality constraint functions are required to be affine, i.e., of the form $a^T x = b$, where a and b are vectors in $\mathbb{R}^n$. By x^* we denote a maximizer of the objective function and OPT denotes the value of the objective at an optimal solution, i.e., OPT=$f_0(x^*)$. We will always assume that there exists a feasible solution and that the problem is bounded, i.e., OPT$< \infty$ holds.

We now briefly discuss *Lagrangian duality* for convex optimization, which can be seen as a generalization of linear programming duality. Lagrangian duality theory has been applied to various problems in communication networks and resulted in distributed algorithms based on a natural problem decomposition.

3.3.1 Lagrangian duality

Consider an optimization problem of the above form, where we do not initially require the f_i and h_i to satisfy the conditions required for a convex optimization problem. Lagrangian duality is based on the approach of relaxing constraints and penalizing infeasibility of solutions by augmenting the

objective function with a weighted sum of the constraint functions. We define the *Lagrangian function* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ associated with the problem as

$$L(x, \lambda, \nu) = f_0(x) - \sum_{i=1}^{m} \lambda_i f_i(x) - \sum_{i=1}^{p} \nu_i h_i(x),$$

with $\text{dom}L = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$. We refer to the vectors λ and ν as the *dual variables*, also called the *Lagrange multiplier vectors* for the given optimization problem. Since we are interested in solving the optimization problem above, we define the *Lagrange dual function* (or just *dual function*) $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ that for any given values of the dual variables gives the supremum of the Lagrangian over x:

$$g(\lambda, \nu) = \sup_{x \in \mathcal{D}} \left\{ f_0(x) - \sum_{i=1}^{m} \lambda_i f_i(x) - \sum_{i=1}^{p} \nu_i h_i(x) \right\},$$

where $\lambda \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^p$. An important observation is that the *dual function is convex*, since it is the point-wise supremum of a family of affine functions of (λ, ν) (parameterized by x), even when the original problem is not convex. Another important result is that the *dual function is differentiable* if the objective function $f_0(x)$ is strictly concave. For an outline of the proof see [14] p. 669 or also [12].

If one considers non-negative Lagrange multipliers, the dual function yields upper bounds on the optimal value OPT$= f_o(x^*)$ of the original maximization problem, i.e., for any $\hat{\lambda} \geq 0$ and any $\hat{\nu}$:

$$g(\hat{\lambda}, \hat{\nu}) \geq \text{OPT}. \tag{3.22}$$

This can be easily seen by noting that for any feasible solution $\hat{x}$ to the original problem, the value of the Lagrangian is an upper bound for $f_0(\hat{x})$. Based on this observation, one may ask about the best upper bound on OPT. This idea is captured by the *(Lagrange) dual problem*, which is formulated as the optimization problem

$$\begin{aligned} &\text{minimize} && g(\lambda, \nu) \\ &\text{subject to} && \lambda_i \geq 0, \qquad\qquad i = 1, \ldots, m. \end{aligned}$$

We call a pair $(\hat{\lambda}, \hat{\nu})$ with $\hat{\lambda} \geq 0$ *dual feasible* and refer to (λ^*, ν^*) as *dual optimal* or *optimal Lagrange multipliers* if they are optimal for the dual problem. Correspondingly, we refer to the original problem as the primal problem. Note that the dual problem is a convex optimization problem, since the objective function and all inequality constraints are convex, even if the primal is possibly not convex. Denote the value of the dual function at the optimum by DOPT, i.e., DOPT $= g(\lambda^*, \nu^*)$.

It is interesting to note that the dual of certain network optimization problems is sometimes easier to solve by a distributed algorithm than the primal. In fact, convex-optimization duality gave rise to an algorithmic technique referred to as *dual decomposition*. This technique has been successfully applied to problems of power assignment and transmission scheduling [129], joint optimization of network flow and resource allocation [151], multipath routing with capacity constraints [113] and distributed coordination of cooperating agents [115], to name just a few.

Weak and strong duality
Since (3.22) holds for any dual feasible solutions, one obtains the following *weak duality theorem*.

Theorem 6 (Weak duality). *DOPT* $\geq$ *OPT.*

The difference DOPT−OPT is referred to as the *duality gap*. Contrary to linear programming, the duality gap does not have to be zero. If the duality gap is zero, we say that *strong duality holds*. In the case of a convex problem, one sufficient condition for strong duality to hold is Slater's condition, as formulated in the following theorem.

Theorem 7 (Slater's theorem). *Assume a convex optimization problem in canonical form*

$$\begin{aligned} &\textit{maximize} && f_0(x) && \\ &\textit{subject to} && f_i(x) \leq 0, && i = 1, \ldots, m \\ &&& h_i(x) = 0, && i = 1, \ldots, p. \end{aligned}$$

If there exists a strictly feasible *solution* $\hat{x}$*, i.e., a solution that satisfies* $f_i(\hat{x}) < 0$*,* $\forall i = 1, \ldots, m$ *and* $h_i(\hat{x}) = 0, \forall i = 1, \ldots, p$*, then strong duality holds:*

$$\text{DOPT} = \text{OPT}.$$

In the case where some inequality constraints are affine, Slater's condition can be modified so that it is sufficient that these affine constraints hold with equality.

Sufficient optimality conditions
Several sufficient conditions for optimality exist within the context of Lagrangian duality. The following condition is very general and in fact does not require the problem to be convex or the objective and constraint functions to be differentiable (see [12, p. 322]).

Proposition 3 (General sufficiency condition). *Consider an optimization problem of the form*

$$\begin{aligned} &\textit{maximize} && f_0(x) && \\ &\textit{subject to} && f_i(x) \leq 0, && i = 1, \ldots, m \\ &&& x \in X, \end{aligned}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ *and* $X \subseteq \mathbb{R}^n$*. Let* x^* *be a feasible solution to the problem and let* $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ *be a vector that satisfies*

$$\lambda_i^* \geq 0, \qquad i = 1, \ldots, m \tag{3.23}$$

$$f_i(x^*) < 0 \Rightarrow \lambda_i^* = 0 \qquad i = 1, \ldots, m \tag{3.24}$$

and let x^* *maximize the Lagrangian* $L(x, \lambda^*)$ *over all* $x \in X$*:*

$$x^* \in \arg\max_{x \in X} L(x, \lambda^*) = \arg\max_{x \in X} \left[f_0(x) - \sum_{i=1}^{m} \lambda_i^* f_i(x) \right],$$

where $\arg\max_{x \in X} L(x, \lambda^*)$ *denotes the set of maximizers. Then* x^* *achieves a global maximum of the problem above.*

Note that there may be more than one primal solution maximizing the Lagrangian for any fixed value for λ. For a pair of primal and dual feasible solutions (3.24) can be also rewritten as the following condition, also called the *complementary slackness condition*.

$$\lambda_i^* f_i(x^*) = 0, \quad i = 1, \ldots, m.$$

In the case that the objective and all constraint functions are differentiable, the f_i are convex and the h_i are affine, it is usually more common to consider the *Karush-Kuhn-Tucker* (KKT) conditions, which are in this case necessary and sufficient for optimality.

Proposition 4 (KKT conditions)**.** *Consider a convex program in canonical form and let $\hat{x}$, $\hat{\lambda}$, $\hat{\nu}$ be any points that satisfy*

$$\begin{aligned} f_i(\hat{x}) &\leq 0, \quad i = 1, \ldots, m \\ h_i(\hat{x}) &= 0, \quad i = 1, \ldots, p \\ \hat{\lambda} &\geq 0, \quad i = 1, \ldots, m \\ \hat{\lambda} f_i(\hat{x}) &= 0, \quad i = 1, \ldots, m \\ \nabla f_0(\hat{x}) + \sum_{i=1}^{m} \hat{\lambda}_i \nabla f_i(\hat{x}) + \sum_{i=1}^{p} \hat{\nu}_i \nabla h_i(\hat{x}) &= 0, \end{aligned}$$

then $\hat{x}$ and $(\hat{\lambda}, \hat{\nu})$ are primal and dual optimal with zero duality gap.

In the case that the conditions for Proposition 4 are satisfied, the KKT conditions may be used to find an optimal solution to the problem. In order to determine a unique primal optimal solution by the condition that the gradient of the Lagrangian vanishes at optimality, it is usually required that the objective function is strictly concave. If this is not the case, then a gradient based approach typically leads to oscillations in the primal solutions [144].

3.3.2 Algorithms

The body of literature on algorithms for nonlinear optimization is vast. Here we list a few methods that share the property that they only require simple operations that sometimes can be performed in a somewhat decentralized manner, although in general other more efficient algorithms exist. For convergence results of the algorithms see the standard literature, for example, [12, 14, 19].

Projected gradient method

Consider the problem of maximizing a differentiable objective function $f : \mathbb{R}^n \to \mathbb{R}$ over a set $X \subseteq \mathbb{R}^n$. Choose a constant stepsize $\alpha > 0$ and an initial feasible solution $x(0) \in X$. The *projected gradient method* with constant stepsize α is defined by the update rule

$$x(k+1) = \left[x(k) + \alpha \nabla f(x(k))\right]^+,$$

where $[\cdot]^+$ denotes the projection onto the set X. It can be shown that if the gradient ∇f is Lipschitz continuous and the stepsize is sufficiently small, the limit points of the sequence of iterates are local maxima of f over X, which are also global maxima if the problem is convex (see Chapter 2.3 in [12]).

Gauss-Seidel method
Consider the problem of maximizing an objective function $f : \mathbb{R}^n \to \mathbb{R}$ over a set $X = X_1 \times X_2 \subseteq \mathbb{R}^{m+(n-m)}$. Suppose that f is continuously differentiable and concave over X. Assume further that the function $f(x_1, \tilde{x}_2)$ is strictly concave over X_1 for any fixed $\tilde{x}_2 \in X_2$ and vice versa for x_2. The *Gauss-Seidel method* is defined by the update rules

$$x_1(k+1) = \arg\max_{x_1 \in X_1} f(x_1, x_2(k)), \quad x_2(k+1) = \arg\max_{x_2 \in X_2} f(x_1(k+1), x_2).$$

Note that the maxima in the update rules are uniquely attained. It can be shown that every limit point of the sequence $\{x(k) = (x_1(k), x_2(k))\}$ maximizes f over X. The Gauss-Seidel method can also be generalized for an arbitrary block-size of X, i.e., when $X = X_1 \times \ldots \times X_B$.

Proximal optimization algorithm
Consider the problem of maximizing a concave objective function $f : \mathbb{R}^n \to \mathbb{R}$ over a convex and nonempty closed set $X \subseteq \mathbb{R}^n$. Since f is not necessarily strictly concave, the dual function may not be differentiable and a gradient-based approach for solving the dual problem may not be applicable. However, by introducing additional variables and a quadratic term it is possible to apply the Gauss-Seidel method and use a gradient method for solving the dual of a subproblem in each iteration.

Consider the following modified problem.

$$\begin{aligned} \text{maximize} \quad & f(x) - \frac{1}{2D}\|x - \bar{x}\|_2^2 \\ & x \in X, \bar{x} \in \mathbb{R}^n, \end{aligned}$$

where D is a positive scalar parameter and $||\cdot||_2$ is the Euclidean norm. It is easy to see that this problem is equivalent to the original problem and $\bar{x}^* = x^*$ for any optimal solution. Moreover, the Gauss-Seidel method is applicable, since the objective function is strictly concave in x for any fixed value for $\bar{x}$ and vice versa. The update rules can be combined into a single rule as follows.

$$x(k+1) = \arg\max_{x \in X} \left[f(x) - \frac{1}{2D}\|x - x(k)\|_2^2 \right]$$

Note that the problem solved in each iteration of the algorithm has a unique optimal solution.

4 DISTRIBUTED ALGORITHMS

This chapter formalizes the model for the computing environment of distributed algorithms presented later in this thesis. We first introduce concepts and basic assumptions and then describe algorithms that are useful for solving common subproblems. Since there exists a vast body of literature on distributed algorithms, this chapter can hardly give a comprehensive overview of the field. The model, definitions and notation of this chapter are mostly based on Santoro's textbook [121].

4.1 MODEL AND ASSUMPTIONS

We start by formulating a model for distributed computation and also note important assumptions to be made in the following chapters.

Computing environment

We define a *distributed computing environment* as a collection of interconnected *nodes* (or processors, or entities). Each node is able to perform computations and has a *state* that comprises *local memory*, a *state register* and an *input register*. The local memory can be used for storing and retrieving data, e.g., results from previous computations, and its storage capacity is unlimited. The state register stores information on the status of the node, and the input register is used for input that is external to the algorithm (e.g., sensor data). Each node also possesses a local clock, which it can use to set a *timer* to expire after a specified delay. The clocks among nodes need not be synchronized.

Besides performing local operations, nodes are able to communicate with their neighbors by sending and receiving *messages*. The interconnection between nodes is modeled as a communication graph, as described in Section 2.1. When there is no risk for ambiguity, we will identify a distributed computing environment by its communication graph $G = (V, E)$. Usually, we assume that the communication graph is directed. In other cases, e.g., when the problem at hand itself requires bidirectional communication, we may consider G to be undirected.

Since the algorithms developed in this thesis approach problems within the context of real-world networks, we assume unique node identifiers for the graph G. Further, we consider each node to be represented in V by its identifier, which in reality may be a network address. We always assume that nodes know their own identifier. Sometimes we may assume that nodes are aware of the identifiers of their neighbors in the graph and that nodes know which edge leads to which neighbor. Note that the two latter assumptions can be satisfied by letting nodes initially send their identifier to all their neighbors.

The transmission of messages over links may fail, as described below. Nodes, however, are assumed not to fail unpredictably. If it is possible that a node fails, e.g., because it is battery powered and its energy is depleted, then this failure has to be part of the description of the environment. Our

algorithms, however, assume the absence of node failures that can not be modeled as transient transmission errors.

Distributed algorithm

We use a *reactive model* [121] to describe distributed algorithms. In this model, any operation performed by a node is a reaction to an *event* that is external to the node. Events include the arrival of a message or the expiration of a timer. Additionally, a special event called *spontaneous impulse* may start computation and communication. The *action* taken by the node as a response to the event may depend on the node's state, as stored in its state register. Possible actions are computations, timer-set and timer-reset operations, state changes, message transmissions and memory access operations. Each action is assumed to be atomic and must terminate within finite time.

A *distributed algorithm* in this model is a set of *transition rules*

$$\text{state} \times \text{event} \mapsto \text{action},$$

where *state* is one of a finite set of node states stored in the node's state register. The algorithms are *homogeneous* in the sense that all nodes obey the same set of rules. The set of rules is required to be complete and non-ambiguous, in the sense that for each pair of state and event there is exactly one resulting action, which may be the special *null action* that does nothing.

It follows from the reactive model that if there are no events, then a node remains in its current state and does not perform any action. We will further use the convention of leaving out rules that have the null action on their right side. In the description of algorithms we group rules according to the states they apply to. In the case of complex actions, we may split an action and write it in the form of several sequential statements, much like statements of a computer programming language. This convention simplifies presentation but does not change the atomicity of actions.

By defining a precedence relationship between events one achieves that the behavior of a node is always well-defined (e.g., spontaneous impulse takes precedence over a timer event, which takes precedence over a message reception). In the case of the simultaneous reception of multiple messages these will be processed in an arbitrary order.

The initialization of local node data (memory, input and state registers) is part of the algorithm, where the initialization of some nodes may occur later than for others. A standard assumption for distributed algorithms is the *unique initiator assumption*, which states that a single node initiates the algorithm. Note that in the reactive model this is the result of a spontaneous impulse event.

Communication

The transmission and the reception of messages over the links in the communication graph are modeled as events. In the absence of failures, messages are subjected to a finite *communication delay*, which models queuing and processing delays. This delay is assumed to be random and non-predictable. Each link can be seen as a queue where messages can be transmitted out-of-

order. The reception of messages at a single node, however, proceeds in the order of their arrival.

Sometimes messages can be lost during their transmission, for example, due to collisions with other messages at the receiver. We do not want to assume that the transmitter can detect the failure. However, by requiring an acknowledgment and using retransmission timers, we may assume that the message is eventually delivered. This retransmission scheme is not part of the algorithm, but its presence may be assumed by the algorithm. In practice, one expects the algorithm to run on top of a MAC protocol that supports the transmission of unicast messages, whose successful reception is acknowledged, as described in Section 2.2.1. We will always assume that the average number of retransmissions of any message is bounded by some constant as the network grows. Note that also retransmissions can cause messages to arrive out-of-order.

A node v may target any neighbor $u \in N_{\text{out}}(v)$ and send a single message to u independent of v's other neighbors. We refer to this as the *unicast model*. Sometimes it is more convenient to use a *broadcast model*, where v may send a single message to all neighbors in $N_{\text{out}}(v)$ at once. However, because of transmission failures it is possible that not all targeted recipients will actually receive the message. Therefore, there is a fundamental difference between broadcast and unicast messages.

The use of acknowledgments for unicast messages requires that bidirectional communication between neighbors is possible. Hence, if the communication graph is directed, we assume it to be symmetric. Since also unicast communication is implemented on top of a wireless channel MAC protocol, nodes may implement a scheme which allows them to *overhear* messages destined for other neighboring nodes. In this sense, nodes may decide to act opportunistically by exploiting the wireless broadcast advantage [39, 148].

Termination

We distinguish between *local* and *global termination*. If a node v can detect local termination, then v is able to determine that it will not have to take further action according to the distributed algorithm. It is possible, however, that the algorithm has not yet terminated at other nodes in the communication graph. If we require that a node is able to detect global termination, then all nodes have to be able to detect local termination and also know about all other nodes being in the state of local termination. In this sense, termination has to be become *common knowledge* [121]. Usually, however, it will be sufficient for our algorithms to detect local termination.

Complexity measures

We will use two main criteria for evaluating distributed algorithms, *message complexity* and *time complexity*. We define the message complexity of a distributed algorithm to be the number of messages transmitted until termination. Since the size of messages can vary, it is sometimes more appropriate to determine the *communication complexity* (also called *bit complexity*) of an algorithm, which is the number of bits transmitted until termination. When

considering communication complexity, we disregard factors that depend logarithmically on the size of the network, e.g., we assume that the number of bits required to represent node identifiers is constant. Similarly, we may assume a fixed precision for representing values, such as node weights.

Note that when determining message complexity, we count transmitted and not received messages. Both coincide in the unicast model but may differ in the broadcast model. Therefore, when expressing message complexity, we will always make clear in the context which of the two models we are referring to. Since we assume that the average number of message retransmissions is bounded, when considering message complexity we can always disregard retransmissions and assume that no messages are permanently lost, e.g., due to message collisions.

In an actual implementation the bandwidth of the channel is limited, which translates into bounds on the size of a message. Hence, we usually limit message size to $\mathcal{O}(\log |V|)$, which results in a constant number of bits under the assumptions above. This model is referred to as the $\mathcal{CONGEST}$ model in the textbook by Peleg [110]. Sometimes we may allow larger messages, for example, when nodes include information about their neighborhood in the message. In this case we account for larger messages by considering these to be split into separate messages of size at most $\mathcal{O}(\log |V|)$.

The second important measure for algorithm performance is time complexity. For determining the time complexity of a distributed algorithm, it is usually assumed that messages experience unit communication delay and that there exists a global clock. In this setting the algorithm operates in rounds and a node v can send one message to each neighbor in $N_{\text{out}}(v)$ and receive one message from each neighbor in $N_{\text{in}}(v)$ in each round. The time complexity then equals the number of rounds required for termination.

When evaluating algorithms by simulations, we instead measure the *execution time*, which is the simulated running time required for a given problem instance. The execution time may give a more realistic impression of the time spent by the execution of an algorithm compared to the unit delay assumption. However, if the algorithm employs timers, then its execution time strongly depends on the timeout values, and special care needs to be taken when comparing the performance of different algorithms.

Recently, the study of so-called *local algorithms* has received considerable attention [110]. These algorithms operate in a synchronous message passing model that assumes the absence of errors. After initially all nodes wake up simultaneously, the time complexity of algorithms in this model is required to be significantly smaller than the network diameter, usually even constant. Hence, algorithms in this model are inherently scalable since nodes can only operate based on local information. Although the model is interesting from a theoretical perspective and properties of local algorithms are highly desirable, it is sometimes too restricted to allow any algorithm to solve relevant problems in wireless networks. For example, Linial [86] showed that there is no distributed constant time algorithm for coloring an n-cycle with three colors. In other cases, the proposed algorithms are only intended to show the existence of local algorithms for a certain problem, though they do not provide for a simple implementation as network protocols. Hence, in this thesis we decided not to restrict the time complexity of algorithms.

4.2 ALGORITHMS

We now demonstrate some of the concepts introduced above by formulating distributed algorithms to be used for solving subproblems in later chapters.

4.2.1 Spanning trees

A standard problem in graph theory is to compute a spanning tree of a given graph. After the termination of the algorithm, each node is required to know which of its incident edges are part of the spanning tree. Note that disregarding trivial cases, this requirement is different from demanding that all nodes will eventually know the complete spanning tree.

In the following we describe Algorithm SHOUT based on the presentation in [121]. The algorithm assumes the presence of a unique initiator, which distinguishes itself from the other nodes in that it has the value of its input register is_initiator set to true and not false. Additionally, the algorithm makes the following three assumptions: the communication graph $G = (V, E)$ is undirected, the transmission of unicast messages is reliable, and all nodes know their neighbors.

Algorithm SHOUT is based on the simple idea of flooding query messages in the network. A node that receives a query for the first time transmits a copy of the query to all its neighbors except the neighbor it received the query from. The edges that were used to transmit these "first time queries" later form the edges in the spanning tree, which can be rooted at the initiator. In addition to sending and receiving queries, a node also collects acknowledgments from its neighbors, which indicate that the incident edge is part of the tree. The first query message is sent by the initiator node that acts upon a spontaneous impulse.

More specifically, when the initiator processes the spontaneous-impulse event, it initializes its set of tree neighbors to the empty set and transmits a query message to all its neighbors. Whenever a node v receives a query message for the first time from a node u, v replies with a "yes" message, confirming the sender u that the edge $\{u, v\}$ is part of the tree to be constructed. Node v then initializes its own set of tree neighbors $N_T(v)$ to $\{u\}$. This occurs when node v is in the IDLE state. After receiving a query for the first time, a node with more than one neighbor changes to the ACTIVE state.

In the ACTIVE state a node v updates $N_T(v)$ by adding nodes from which v receives "yes" messages. Each node also keeps track of the number of queries and acknowledgments it receives. By comparing this number to the number of its neighbors in the communication graph, the node is able to determine when the algorithm has terminated locally.

Theorem 8. *The message complexity of the Shout algorithm satisfies*

$$M[\mathit{Shout}] = 2|E|.$$

Proof. Consider any edge $\{u, v\}$ in the communication graph and the messages that are sent via this edge. In the case that the edge is not in the spanning tree, two query messages, one in either direction, are sent over the edge. If the edge is part of the tree, then either u or v sends a query and the other node replies with a "yes" message. □

Algorithm SHOUT: Spanning tree construction
(see improved version of SHOUT in [121])

```
1  node v with variables root, N_T(v), counter, father, is_initiator;
2  at start
3  |  if is_initiator then enter state INITIATOR;
4  |  else enter state IDLE;
5
6  in state INITIATOR                          // initiator starts algorithm
7  |  if spontaneous impulse then
8  |  |  root ← true; N_T(v) ← ∅;
9  |  |  send (Q) to N(v);
10 |  |  counter ← 0;
11 |  |  enter state ACTIVE;
12 |  end
13
14 in state IDLE
15 |  if message (Q) is received from u then
16 |  |  root ← false; N_T(v) ← {u};
17 |  |  father ← u; counter ← 1;
18 |  |  send (Yes) to {u};
19 |  |  if counter = |N(v)| then enter state DONE;
20 |  |  else
21 |  |  |  send (Q) to N(v) \ {u};
22 |  |  |  enter state ACTIVE;
23 |  |  end
24 |  end
25
26 in state ACTIVE
27 |  if message (Q) is received from u then
28 |  |  counter ← counter + 1;
29 |  |  if counter = |N(v)| then enter state DONE;
30 |  end
31 |  if (Yes) is received from u then
32 |  |  N_T(v) ← N_T(v) ∪ {u};
33 |  |  counter ← counter + 1;
34 |  |  if counter = |N(v)| then enter state DONE;
35 |  end
36
```

Considering time complexity, it is easy to see that the algorithm can require a number of rounds that is at least equal to the diameter of the communication graph, since the queries have to reach each node starting at the initiator. The nodes that were at the furthest distance from the initiator then require an additional round for transmitting their replies. Hence, one obtains the following result.

Theorem 9. *The time complexity of the Shout algorithm satisfies*

$$T[\mathit{Shout}] \leq d(G) + 1,$$

where $d(G)$ is the diameter of G.

If one is interested in computing an MST instead of just any spanning tree, Algorithm SHOUT cannot be applied applied in general. A well-known distributed algorithm for this problem was proposed by Gallager et al. [46]. Although it solves the more general problem, it seems unsuitable for the simple case of unit weights due to its implementation complexity.

Interestingly, Algorithm SHOUT may be modified to handle multiple initiators. Informally, the idea is to let all initiators follow the steps of the single initiator as described above. During the algorithm, a node stops participating in the construction of a spanning tree whenever it receives a query that originated from an initiator with a smaller identifier than the one it participated in previously. The modification requires the inclusion of initiator-node identifiers in query messages. This still relatively simple algorithm may be used also for *leader election*, which is the general problem of selecting a single coordinator in a distributed computing environment. For this problem, however, more efficient algorithms exist. It is interesting to note that for general graphs the MST algorithm of [46] has asymptotically optimal message complexity for the leader election problem. For details we refer to [121].

4.2.2 Shortest paths

Computing the shortest paths from a given initiator node to all other nodes is another standard problem in graph theory. In this section, we assume that the distributed algorithm is required to compute shortest paths on the communication graph $G = (V, E)$, which is edge-weighted with weight function $w_E : E \mapsto \mathbb{R}_{>0}$.

Algorithm BF is the asynchronous version of the Bellman-Ford algorithm (see [90, Sec. 15.4]) that is augmented by sending acknowledgment messages to let the initiator node detect termination of the algorithm. Algorithm BF is based on the same assumptions as Algorithm SHOUT, i.e., the existence of a unique initiator, an undirected communication graph, reliable message transmissions and knowledge of node neighborhoods.

For each node v, the algorithm maintains father pointers to the node which is the next hop on a shortest path to the initiator node among all paths currently known to v. Since messages can arrive out of order, the acknowledgments need to contain information on the length of the path whose message is acknowledged. After termination of the algorithm, the father pointers form a *shortest-path spanning tree* of the communication graph, rooted at the initiator, which can be used, e.g., for routing messages. Note that since the

Algorithm BF: Distributed shortest-path algorithm
(asynchronous Bellman-Ford algorithm, see [90])

```
1  node v with local variables dist, dist[·], father, status[·]
2  at start
3      father ← undefined;
4      if is_initiator then
5          dist ← 0;
6          for u in N(v) do
7              send (w_E(v, u)) to u;
8              dist[u] ← w_E(v, u);
9              status [u] ← wait;
10         end
11         enter state SEARCH;
12     else
13         dist ← ∞;
14         for u ∈ N(v) do
15             dist[u] ← ∞;
16             status [u] ← ready;
17         end
18         enter state IDLE;
19     end
20
21 in state IDLE or SEARCH
22     if (dist′) with dist′ < dist is received from some node u then
23         if father is defined then send NAK(dist) to father;
24         father ← u; dist ← dist′;
25         for w in N(v) \ {u} do
26             send (dist′ + w_E(v, w)) to w;
27             dist[w] ← dist′ + w_E(v, w);
28             status [w] ← wait;
29         end
30         enter state SEARCH
31     end
32     if (dist′) with dist′ ≥ dist is received from some node u then
33         send NAK(dist′) to u;
34     end
35
36 in state SEARCH                    // wait for incoming acknowledgments
37     if status [w]=ready for all w ∈ N(v) \ {father} then
38         send ACK(dist) to father;
39         enter state IDLE
40     end
41     if ACK(dist′) or NAK(dist′) is received from u and dist[u] = dist′
       then
42         status [u] ← ready;
43     end
44
```

graph is assumed to be undirected, the tree can be used to forward messages from the root to the leaf nodes and vice versa.

More precisely, Algorithm BF is initiated when the initiator node sends to each of its neighbors a message that contains the cost of the edge between itself and them (we have omitted the initial spontaneous impulse event from the pseudocode). Whenever a node receives this kind of request for the first time, it stores the identifier of the neighbor that sent the message in its *father* variable and retransmits the message to its other neighbors after updating the total path length contained in the request. Each node also stores the length value of the last request sent to each neighbor. If a node that has already received and retransmitted a request receives a request that indicates a shorter path from the initiator node, it resends the new request to its neighbors. Hence, for each node v, dist is the current estimate of the distance from the initiator node to v and father is the node from which v has received the last request that was accepted by v.

The only node that is able to detect global or even local termination of the algorithm is the initiator node after it has received acknowledgments from all its neighbors. Hence, if it is required that all nodes detect termination, one may extend the algorithm by a network-wide broadcast phase initiated by the initiator node after itself detecting termination. Thus, all other nodes are informed of global termination of the shortest path algorithm. Also, if the task is to compute shortest paths between all pairs of nodes, one can run $|V|$ copies of the algorithm in parallel.

It is well-known that the worst-case message complexity of the asynchronous Bellmann-Ford algorithm is exponential in the size of the graph [90]. In Chapter 7 we modify Algorithm BF and obtain Algorithm MLS, which constructs a minmax-spanner of the communication graph. Our modified algorithm circumvents the problem of an exponential message complexity by pruning the search locally.

Since Algorithm BF constructs a spanning tree of the communication graph, it may be applied to establish a routing tree to disseminate information originating from the initiator node. However, for applications that allow paths to be longer than shortest paths, one may prefer Algorithm SHOUT, which is much simpler and has a better worst-case message complexity. Note also that some information stored by Algorithm BF is redundant, since it would be possible to compute the dist[u] values from dist and the edge costs. We prefer this representation to show similarities to Algorithm MLS. For the same reason we choose to initialize the *status* field for the neighbors of all nodes but the initiator to *ready* in line 16, although it will be set to *wait* as soon as v receives the first request.

Convergecast

Algorithm BF employs a technique that is usually referred to as *convergecast*, which in some sense acts as a reverse network-wide broadcast. After a node v has received the first message, it waits until all its neighbors have sent acknowledgments, positive or negative, by updating their status fields. Only then v sends an ACK to its father. This ensures that when the initiator has received all acknowledgments, it is guaranteed that the algorithm has termi-

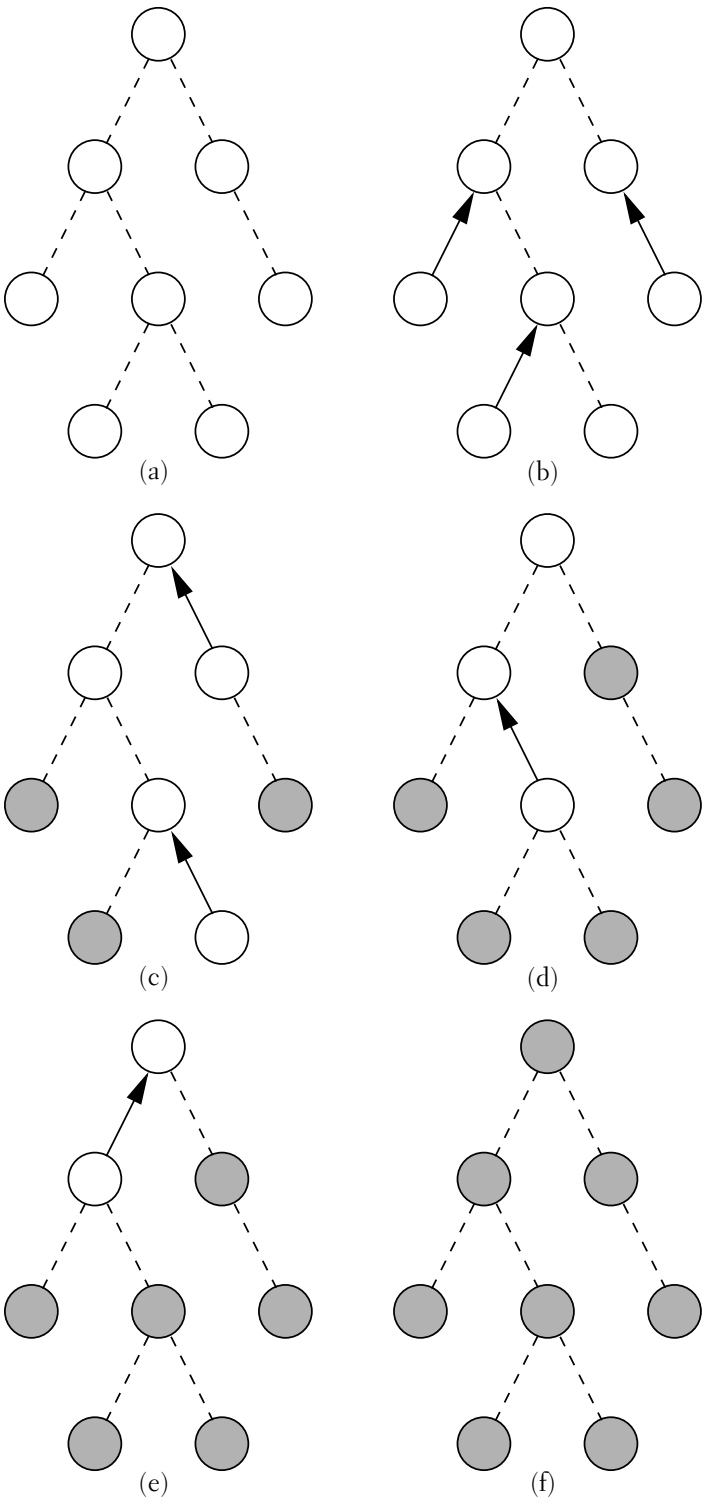


Figure 4.1: Possible execution of a convergecast operation; each node waits for all its children in the spanning tree to submit their message before it sends its own to its father. Nodes at which the convergecast has terminated are shown in gray. The root is the node on top.

nated. However, the same technique can also be applied for other purposes, such as collecting aggregated information, e.g., the minimum or maximum of a certain value present at each node. We use convergecasts at several occasions in the following chapters. Since this elementary technique is so useful, we visualize it in Figure 4.1.

4.2.3 Independent sets

Independent sets are interesting within the context of multihop wireless networks for a variety of applications. Assume that adjacent nodes in a given *interference graph* $G_I = (V, E_I)$ are mutually conflicting in the sense that simultaneous message transmissions lead to collisions. Then finding an independent set in G_I corresponds to finding a mutually non-conflicting set of transmitters in G_I. Since the problem of finding a maximum size independent set in general graphs is computationally hard, several approaches for approximation algorithms have been proposed [57].

A problem that is related to the one mentioned but efficiently solvable in a centralized setting is the problem of finding a maximal independent set. Algorithm MIS is a distributed asynchronous algorithm for this problem proposed by Wan et al. [141]. The algorithm assumes that all nodes know their *level* in a spanning tree rooted at the initiator node, where the level of a node equals its hop-distance in the tree from the root. In addition to its own level, nodes also know the levels and identifiers of their neighbors in the communication graph.[1] Algorithm MIS uses a ranking of the nodes according to the lexicographic order of the (level, identifier) pairs.

The algorithm colors nodes according to their state with respect to the independent set under construction. Initially, all nodes are colored white. When a white node v determines that it has the smallest rank among all the white nodes in its extended neighborhood $N^+(v)$, it decides to add itself to the independent set. Node v notifies its neighbors of its entering the set with a broadcast message, and the neighbors are then colored gray.

In Algorithm MIS, the variable x counts the number of acknowledgments that remain to be received by the children of v in the tree. This counter is decreased whenever a MARK-COMPLETE message is received. A convergecast operation of MARK-COMPLETE messages is used to detect termination at the root node, similar to the operation performed by Algorithm BF. The variable y counts the number of neighbors of v that have a lower rank than v and have not yet been marked gray. Based on the value of y, each node v can determine when it is required to enter the independent set. Since the root node has the lowest rank in the graph, it marks itself black immediately and initiates the algorithm. Note that the algorithm uses both unicast and broadcast messages and assumes that no transmission errors occur.

One should note that due to the ranking of the nodes, the resulting independent set I has the property that the nodes in I can be connected via only a few nodes in $V \setminus I$. More precisely, for any $v \in I$, the length of the shortest path to any node in $I \setminus \{v\}$ is exactly 2 for $|I| > 1$ (see [141]). Since a maximal independent set is also always a dominating set, this property can

[1] This assumption could be satisfied by initially running a slightly modified version of Algorithm SHOUT or defining unit edge costs and using Algorithm BF.

Algorithm MIS: Distributed maximal independent set algorithm (based on [141])

```
1  node v with local variables level[·], rank[·], father, x, y
2  at start
3      for u ∈ N+(v) do rank[u] ← (level[u], u);
4      x ← number of children in tree;
5      y ← number of lower ranked neighbors in N(v);
6      color ← white;
7      blackList ← ∅;
8      if is_initiator then
9          enter state CHECK;
10     else
11         enter state IDLE;
12     end
13
14 in state CHECK          // check condition for entering independent set
15     if y == 0 then
16         color ← black;
17         blackList ← blackList ∪ {v};
18         broadcast BLACK message;
19         enter state WAIT;
20     else
21         enter state IDLE;
22     end
23
24 in state IDLE                                  // process incoming messages
25     if GRAY is received from node u then
26         if u has lower rank than v then
27             y ← y − 1;
28             enter state CHECK;
29         end
30     end
31     if BLACK is received from node u then
32         color ← gray;
33         blackList ← blackList ∪ {u};
34         broadcast GRAY message;
35         enter state WAIT;
36     end
37     if MARK-COMPLETE is received from child u then x ← x − 1;
38
39 in state WAIT
40     if MARK-COMPLETE is received from child u then x ← x − 1;
41     if x == 0 and not is_initiator then
42         send MARK-COMPLETE to father;
43         enter state TERMINATE;
44     end
45
```

be used for computing a connected dominating set that serves as backbone for routing messages in the network.

It is well-known that in a unit disk graph the size of a maximum independent set in any neighborhood $N^+(v)$ is at most five [91]. Hence, Algorithm MIS is in fact a factor $1/5$ approximation algorithm for the maximum independent set and a factor 5 approximation algorithm for the minimum dominating set problem in unit disk graphs. This result can easily be extended to bounded independence graphs, since these, by definition, have a constant number of independent nodes within any one-hop neighborhood.

An algorithm that is very similar to Algorithm MIS was proposed by Wang et al. [145] to solve a subproblem for the minimum weight connected dominating set problem. Instead of defining the node ranks based on the tree level, the algorithm in [145] lets nodes enter the dominating set if they have the minimum weight in their neighborhood. The node-weight function w_V was assumed to be smooth, i.e., the maximum ratio of its values between adjacent nodes, which influences the approximation factor of the algorithm, was assumed to be rather small.

4.2.4 Dominating sets

Dominating sets frequently appear in formulations of problems related to wireless networks, e.g., routing problems, when messages are relayed only using nodes in the potentially small dominating set. For routing one typically requires the dominating sets to be connected. Dominating sets are also a useful model for providing coverage in sensor networks. Consequently, several centralized and distributed algorithms have been proposed. See [17] for an overview of the relevant literature.

The minimum weight dominating set problem, where each node v has a positive weight $w(v)$ and the objective is to minimize the total weight of the dominating set, can be easily formulated as a weighted set-cover problem by choosing $U = V$, $\mathcal{S} = \{N^+(v) \mid v \in V\}$ and $w(N^+(v)) = w(v)$. Therefore, we can apply the (centralized) greedy set-cover algorithm to this problem. Based on the presentation in Section 3.2.2, it is then clear that one obtains a dominating set of size at most $\phi = H_{\Delta^+}$ times the optimum, where H_n is the n-th harmonic number. This follows since Δ^+ is the size of the largest one-hop neighborhood and hence the size of the largest set in the corresponding set-cover instance. The approximation factor is thus $\mathcal{O}(\ln \Delta^+)$.

As argued previously, Algorithm MIS obtains a constant-factor approximation in unit disk graphs with unit weights and for these graphs even polynomial-time approximation schemes are possible [101]. If one considers general communication graphs, then the inapproximability results for the set-cover problem [41] imply that Chvátal's algorithm is essentially the best-possible polynomial time approximation algorithm under standard complexity assumptions. Distributed algorithms based on Chvátal's set-cover algorithm have been proposed earlier for both unit and arbitrary weights. One example is the spine routing algorithm by Sivakumar et al. [128]. Most of the algorithms assume synchronous communication rounds. See Algorithm SynMWDS for the distributed synchronous version of the MWDS greedy algorithm, which is adapted from the algorithm of [128] proposed for the

Algorithm SYNMWDS: Distributed synchronous MWDS algorithm (adapted from [128])

```
1  node v with local variables w(v), span(v), N(v), N_2(v), color
2  at start
3  |  color ← white;
4  |  span(v) ← |N(v)| + 1;
5
6  repeat
7  |  send (v, w(v), span(v)) to all u ∈ N(v);
8  |  wait for incoming (u, w(u), span(u)) from all u ∈ N(v);
9  |  for u ∈ N(v) do send (u, w(u), span(u)) to all w ∈ N(v);
10 |  wait for incoming (u, w(u), span(u)) for all u ∈ N_2(v);
11 |  if (w(v)/span(v), v) < (w(u)/span(u), u)  ∀u ∈ N_2^+(v) and color == white then
12 |  |  color ← black;
13 |  |  send BLACK to all u ∈ N(v);
14 |  end
15 |  if BLACK is received from node u then
16 |  |  if color == white then
17 |  |  |  color ← gray;
18 |  |  |  send GRAY to all u ∈ N(v);
19 |  |  end
20 |  |  span(v) ← span(v) − 1;
21 |  end
22 |  if GRAY is received from node u then span(v) ← span(v) − 1;
23 until for all u holds: span(u) == 0;
```

unweighted case.

The steps in Algorithm SYNMWDS are executed by all nodes in a lock-step fashion. Define the *span* of a node v as the number of nodes within v's extended neighborhood $N^+(v)$ that currently have no neighbor in the dominating set and are also themselves not in the dominating set. In this sense, the span of a node v is the number of *uncovered* nodes it would *cover*. Each node monitors the span of all nodes within its two-hop extended neighborhood $N_2^+(v)$. If a node recognizes that it has the minimum weight-to-span ratio in $N_2^+(v)$, then it adds itself to the dominating set by marking its color to black. Node identifiers are used to break ties.

Whenever a node enters the dominating set, it is colored black and informs its neighbors, which then change their color to gray, i.e., they become covered. After each iteration of the loop all nodes v have the correct span values for $N_2^+(v)$ and at least one node has entered the dominating set in the current iteration. It is easy to see that the dominating set computed by Algorithm SYNMWDS is the same as the solution obtained by Chvátal's greedy algorithm, which in each iterations adds the node with the minimum weight-to-span ratio. This follows from the fact that all nodes that may influence the span of v are within a two-hop distance from v and spans can only decrease.

Although there is the chance that some nodes enter the dominating set

in the same iteration, i.e., if they all achieve the minimum weight-to-span ratio in their neighborhoods, Jia et al. [66] remark that the distributed greedy algorithm has linear time complexity. This follows since there is a family of instances in which nodes enter the dominating set one by one. The authors of [66] propose randomized algorithms with polylogarithmic time complexity and approximation guarantees similar to Chvátal's algorithm. An actual implementation of this algorithm, however, would require careful clock synchronization possibly in a wide-spread network. Although one might be able to work around synchronization issues, using the techniques we describe in the next section, these would require an increase in message overhead.

4.2.5 Network synchronizers

Algorithm SYNMWDS differs from other distributed algorithms in this thesis in its assumption of a synchronous communication model. According to our model of Section 4.1, after their transmission, messages can arrive after a finite but arbitrary delay. However, sometimes simpler algorithms can be obtained if one assumes that nodes operate according to a global clock and send messages only at discrete time points according to ticks of the global clock. This *synchronous distributed computation* model further assumes that message delays are upper bounded by a single clock tick and that every node sends at most one message to every neighbor each time [4,121]. In this sense, one can think of the nodes as operating in rounds, where in each round a node is able to transmit and receive one message over each of the edges in the communication graph.

Although the synchronous model may simplify the design and analysis of algorithms, one cannot avoid to deal with the asynchronous nature of real-world networks. Interestingly, there are general techniques for so-called *synchronizers*, which are methods that transform a synchronous algorithm to an algorithm that can be executed in an asynchronous model, by introducing additional overhead. Awerbuch [4] proposes three types of synchronizers, named α, β and γ, where the third is a combination of the first two.

The synchronizers in [4] are based on the concept of a *safe state* for the nodes. A node is safe with respect to a certain round k, if all messages it has transmitted during round k have already arrived at their destination node. By adding acknowledgments for each message transmitted by the original algorithm, the synchronizer lets nodes detect when they may enter the safe state. Depending on the particular choice of synchronizer, the following steps differ.

Synchronizer α lets a node that has become safe inform its neighbors. Once all neighbors of a node v are in the safe state w.r.t. round k, node v can locally generate the clock tick $k+1$ and enter the next round. The algorithm is guaranteed to work correctly, since it is not possible that v later receives messages originating from round k. The combined algorithm requires $\mathcal{O}(R\ |E|)$ additional messages for the synchronizer, where R is the number of communication rounds needed by the synchronous algorithm.

Synchronizer β is based on the same idea of safe states. Instead of informing all neighbors of becoming safe, however, the network performs a convergecast of safe messages in a spanning tree that is rooted at an initiator

node. This synchronizer requires only $\mathcal{O}(R\,|V|)$ additional messages for the price of increasing the time complexity of an algorithm by a factor proportional to the height of the tree. The third synchronizer γ aims at combining the good features of both methods, low message and time complexity.

Consider now an implementation of Algorithm SYNMWDS on top of a synchronizer. There exist problem instances for which nodes enter the dominating set one at a time, so that the time complexity in terms of loops in Algorithm SYNMWDS is linear [66]. If one disregards the time complexity of the resulting algorithm, one would clearly choose synchronizer β. We obtain the following result:

Theorem 10. *A naive implementation of Algorithm* SYNMWDS *that is based on synchronizer β and suitable for execution in the asynchronous model requires $\Omega(|V|^2\Delta)$ messages in the worst case.*

Proof. In each iteration of the loop in Algorithm SYNMWDS nodes need to first transmit their own and receive weight/span values from all their neighbors, which requires one round. Then, each node v needs to relay the values received from all neighbors u to all other neighbors $N(v) \setminus \{u\}$, which may require $\Delta - 1$ rounds. Since each round requires the transmission of $|V|$ safe-stage messages, the result follows. □

Depending on the spanning tree of synchronizer β, the increase in running time may also be significant. In Chapter 8 we discuss a distributed MWDS approximation algorithm for the asynchronous model which is based on Chvátal's algorithm and has message complexity $\mathcal{O}(|V|\Delta^2)$. It generally holds that $\Delta = \mathcal{O}(|V|)$ but for almost all wireless networks one would expect the maximum degree to be much smaller than $|V|$.

5 LOAD BALANCING VIA MULTIPATH ROUTING

In this chapter we consider the problem of balancing traffic load ideally over a wireless multihop network, which is formulated as an instance of the minimum maximum congestion problem. We propose a multipath routing algorithm that is based on Algorithm YMMC described in Section 3.2.2. The algorithm is implemented in the network simulator NS2 as a modification of the Dynamic Source Routing (DSR) [70] protocol (see Section 2.2.2). Subsequently, the resulting Balanced Multipath Source Routing (BMSR) algorithm is evaluated by simulations, which show that it typically achieves higher network throughput than DSR. We also compare the performance of BMSR to the performance of an idealized shortest-path routing algorithm.

5.1 INTRODUCTION

Load balancing is a general technique that is applied to achieve one or several goals in communication networks. Typically, the task is to utilize the available resources efficiently, thereby maximizing objectives, such as end-to-end throughput. However, load balancing also offers benefits in terms of reliability and may provide redundancy to recover from a limited number of errors, for instance, link breaks [109].

Since wireless network resources, such as battery power or channel bandwidth, are scarce, load-balancing techniques should be applied. We consider a setting where one tries to avoid bottleneck links while satisfying end-to-end traffic demands. This problem is well-known for tethered networks. From an algorithmic point of view, however, it is particularly interesting to develop algorithms that are based on local information available at the nodes, perform only simple operations and can be easily integrated into routing protocols.

A possible application scenario could be an emergency response network that was set up after a major accident which destroyed all information infrastructure. When a sudden demand arises for transmitting a large amount of data between a dedicated pair of nodes, e.g., between a control center and rescue teams, one aims to deliver as much of the critical data as possible. For this purpose one must balance the traffic among the nodes and utilize the network capacity to maximize throughput over source-destination pairs.

A considerable amount of work exists on multipath routing in wireless networks. Multipath extensions to DSR have also been proposed previously. Nasipuri et al. [100] extend the DSR route finding process to consider alternate routes for a given destination. Wu and Harms [150] modify the procedure of forwarding route-reply messages back to the initiator of a route-request to discover alternative routes. The algorithm by Wang et al. [143] distributes traffic over several routes available in DSR's route cache, based on an estimate of round-trip delay. See [98] for a discussion of different approaches to multipath routing in wireless networks. Ganjali and Keshavarzian [47] state that multipath routing *alone* can not improve load balancing: as node density increases, the choice of shortest paths connecting any pair of nodes leads to congestion in the center of the network. They conclude that additional

incentive is needed to push traffic away from the center.

Contrary to the majority of proposed multipath routing algorithms for wireless networks, which evaluate the quality of a path based on some heuristic information, our algorithm chooses paths based on update rules of an approximation algorithm for a linear programming formulation of the underlying problem. More specifically, we consider the MMC problem formulated in Section 3.2.2, which is restated here for convenience.

$$\begin{array}{lll} \textbf{MMC:} & \text{minimize} & \lambda \\ & \text{subject to} & \sum_{p \ni e} x_p \leq \lambda c(e), \qquad \forall e \in E \\ & & \sum_{p \in \mathcal{P}_n} x_p = d_n, \qquad \forall n \\ & & x_p \geq 0, \qquad \forall p \in \mathcal{P} \end{array}$$

Recall that in this problem, there are N source-destination pairs (s_n, t_n), each with a routable demand d_n and a set of paths $\mathcal{P}_n$ from s_n to t_n. To each possible path $p \in \mathcal{P}$, where $\mathcal{P} = \cup_n \mathcal{P}_n$, an optimal solution to the problem assigns a flow x_p, so that the maximum congestion is minimized.

Multipath-based network optimization has been studied extensively for wired networks. Vutukury and Garcia-Luna-Aceves [140] propose to minimize delay by heuristic redirection of flow over multiple paths. Basu, Lin and Ramanathan [9] present a potential-based routing method that forwards packets using steepest gradient search and propose a traffic-aware routing algorithm. This approach relies on a link-state routing algorithm for the dissemination of link information throughout the entire network.

5.2 LOAD BALANCING BY CONGESTION MINIMIZATION

The algorithm presented here is based on Young's version of the approximation algorithm for the MMC problem [16, 154] as discussed in Section 3.2.2 and formulated as Algorithm YMMC. We first describe the algorithm and then discuss some properties and possible extensions.

5.2.1 The BMSR algorithm

Let $G = (V, E)$ be the communication graph of the network and $c : E \mapsto \mathbb{R}^+$ a function that determines the capacity of each edge. These capacities can correspond to limits on the transmission rate of a link, but they may also be derived from other resource constraints. Note that since our BMSR algorithm is based on Algorithm YMMC, we assume unit edge capacities. For simplicity, we also assume that demands are of unit value, which corresponds to all sources having the same amount of routable traffic.

Algorithm BMSR considers G to be directed and edges to be symmetric. Since G is directed, the capacity constraints apply to each pair of edges (v, u) and (u, v) separately. However, the algorithm could be modified for undirected communication graphs. Further, we assume that each node acts only as one source or destination node, respectively. This assumption can be removed by letting each node run several copies of Algorithm BMSR.

Algorithm BMSR: Multipath routing algorithm based on YMMC

```
1  node v with local variables dist[·], ite[·], path, y[·], I, bestpath, ϵ
2  at start
3      for u in N_in(v) do y[u] ← 1;
4      for 1 ≤ n ≤ N do
5          dist[s_n] ← ∞;
6          ite[s_n] ← 0;
7      end
8      if v = s_n for some 1 ≤ n ≤ N then
9          paths ← ∅;    dist[s_n] ← 0;
10         broadcast BREQ(0,0,(s_n), s_n, t_n);
11     end
12     enter state ACTIVE;
13
14 in state ACTIVE
15     if BREQ(ite, dist', path, s_n, t_n) is received from some node u then
16         if (ite = ite[s_n] and dist' + y[u] < dist[s_n]) or ite > ite[s_n] then
17             dist[s_n] ← dist' + y[u];
18             append identifier of v to path;
19             if v ≠ t_n then
20                 broadcast BREQ(ite, dist[s_n], path, s_n, t_n);
21             else
22                 bestpath ← reversed path;
23                 if ite > ite[s_n] then
24                     schedule BREP(ite, bestpath, s_n, t_n) along bestpath
                       after BMSR_ITERATION_DELAY;
25                 end
26             end
27             ite[s_n] ← ite;
28         end
29     end
30     if BREP(ite, path, s_n, t_n) is received then
31         if v ≠ s_n then
32             let u be the next node on path after v;
33             y[u] ← (1 + ϵ)y[u];
34             unicast BREP(ite, path, s_n, t_n) to u;
35         else
36             paths ← paths ∪ {reversed path};
37             if ite[s_n] < I then
38                 ite[s_n] ← ite[s_n] + 1;
39                 broadcast BREQ(ite[s_n], 0, s_n, t_n);
40             end
41         end
42     end
43
```

Recall that each of the I iterations of Algorithm YMMC consists of N shortest path computations that are followed by flow augmentations and edge-length updates along the computed paths. In Algorithm BMSR each node v maintains a variable $y[u]$ for all its incoming neighbors $u \in N_{\text{in}}(v)$, which correspond to the y_e variables of each *outgoing* edge in Algorithm YMMC. After Algorithm BMSR is initiated, each source s_n performs a sequence of I shortest path operations, where I is the number of iterations of Algorithm YMMC. We assume that the sources are loosely synchronized, so that they typically reside in the same iteration of the algorithm, e.g., by using timers. The shortest path computations are implemented as a modification of the DSR route discovery, which essentially leads to the distributed asynchronous Bellman-Ford algorithm of page 48.

When a request reaches the destination node, the destination initiates the propagation of a reply message back to the source, which lets all nodes along the path update the weights of the affected edges. This reverse transmission is facilitated by the source-routing character of DSR. Using distributed weight updates we avoid dissemination of global information.

More precisely, we modify the DSR route discovery, which, in the case of missing routing information, essentially performs a network-wide broadcast of route request (RREQ) messages. The balanced route request in Algorithm BMSR, BREQ, is a modified version of the RREQ of DSR that additionally contains a record of the total weight of the path the BREQ has taken so far. Contrary to DSR, if a node receives a BREQ message of the same iteration and source-destination pair, it may resend it if the second request has a lower weight than the previously seen one.

When a destination node t_n receives a BREQ message destined to itself, it schedules the transmission of a reply, a BREP message. It does not transmit the reply immediately, but waits for a certain amount of time for other incoming BREQ messages with lower path cost. The destination then sends a BREP back to the source by reversing the lowest cost path. Whenever a node v that lies on this path receives a BREP, it updates the value for $y[u]$ that it keeps in its routing table for the next node u on the path towards s_n. Note that the edge (u, v) will be used to route flow along the path from s_n to t_n. When the BREP reaches the source node, s_n records the path from s_n to t_n in its routing table and initiates a new iteration if the total number of iterations I has not been reached. After termination of the algorithm, each source node distributes its traffic evenly over the collected paths, which corresponds to scaling all path flows down by I as required in Algorithm YMMC.

Once the paths are determined, they are used for routing, independently of route failures due to temporarily congested links. This operation deviates from DSR, which initiates route maintenance operations in this case. However, since each source has a balanced collection of routes to the destination, the effect of transient link failure diminishes, as the source distributes the traffic equally among the routes in its the cache.

5.2.2 Discussion

In order to achieve good performance and approach the performance guarantees of the underlying approximation algorithm for the MMC problem,

one needs to choose a large value for the number of iterations I. Based on the analysis in [16], one may choose $I = \lceil \frac{4|E| \log |E|}{\epsilon^2} \rceil$. Our experiments presented later, however, indicate that a reasonably small number of iterations is sufficient for achieving good performance in terms of total throughput. We discuss this aspect further in Section 5.3.2.

Algorithm BMSR updates the weight of incoming edges as BREP messages are passed towards the source node. This update matches the weight update in Algorithm YMMC in all nodes that lie on only one source-destination path but deviates in nodes that lie on several paths. This is due to the fact that in Algorithm YMMC one first determines the increase of flow along the shortest path and then performs the update, which is not possible in Algorithm BMSR due to the nature of the distributed operation. One should note, however, that in the case of small ϵ and a small number of source-destination pairs the difference is small.

A relatively minor modification of Algorithm BMSR, which would let both endpoints of an edge update their notion of the weight of the undirected [*sic*] edge between them, would enable the algorithm to be applied to the MMC problem on the undirected communication graph. Similarly, instead of edge capacities, the problem may be formulated in terms of node capacities. A modified version of the algorithm in this context would maintain node costs rather than edge costs. Since Algorithm BMSR was modeled after Algorithm YMMC for the MMC problem on directed graphs, these modifications are not considered here.

Algorithm YMMC executes the shortest-path computations for all source-destination pairs simultaneously, whereas there is no synchronization between pairs in Algorithm BMSR. This task could be accomplished by adding a termination detection using ACK/NAK messages, as in Algorithm BF, which would also render the waiting timer at destination nodes obsolete. Sources could synchronize and maintain the same iteration index. However, the implementation complexity of the algorithm would increase significantly and simple modifications to DSR would not be sufficient.

Further one should note that since the underlying shortest path algorithm is the asynchronous Bellman-Ford algorithm, the worst-case message complexity is exponential in the number of nodes [90]. However, in practice one rarely observes such bad behavior of the algorithm. In fact, it was shown by Tsitsiklis and Stamoulis [136] that under reasonable assumptions on transmission delays, the expected number of messages for Bellman-Ford type algorithms is polynomial. An alternative implementation of our algorithm may instead rely on a different shortest-path algorithm, for example the distributed algorithm proposed by Haldar [56], which has polynomial message complexity. For the sake of integration with DSR we decided to choose a Bellman-Ford type algorithm and estimate its performance and overhead by network simulations.

Finally, it is clear that when implementing the algorithm as a modification to the DSR implementation of NS2, one needs to respect the limitations of the simulator. One important parameter is the *maximum source-route length*, which limits the hop-count of any path. In our simulations we first selected the default value and then, when network size increased, chose a larger value, which still matched our computational resources.

5.3 SIMULATION AND PERFORMANCE EVALUATION

We now continue to evaluate the BMSR algorithm by network simulations using the NS2 simulator. In all scenarios we consider a stationary network and several source-destination pairs with constant bit rate (CBR) data sources.

5.3.1 Overview of experiments

Recall that the MMC problem is equivalent to the MCF problem, which asks for a flow that maximizes the fraction of routable demand. Although generally this objective is different from maximizing average network throughput, we chose the latter as the main performance criterion for the sake of comparison with DSR. The following questions were addressed when planning the simulation setup.

- Does the BMSR algorithm lead to a performance improvement compared to the standard DSR protocol?
- If so, then how does the choice of routes influence packet collisions and interface queue (IFQ) overflows, which were identified as factors degrading DSR performance in congested networks [61]?
- What is the effect of the parameters ϵ and I on BMSR performance?
- Which role does the number and positioning of source-destination pairs play?

To experimentally validate the BMSR algorithm, we performed simulations for a number of different network topologies. For all simulation trials, we ran BMSR for a chosen value of ϵ and a chosen number of rounds I to select routes for each source-destination pair. Then we let the sources transmit with a previously determined constant rate and packet size. We chose a rather long running time to obtain estimates of the throughput of the network. The same source-destination setup was used to transmit data using the DSR implementation provided in NS2.

For the simulations we chose to separate the execution of Algorithm BMSR from the actual routing of data packets used for evaluation. For this purpose, we introduced an initial *balancing setup phase*, during which the algorithm was run, before the start of the throughput evaluation.

In a first set of experiments we only considered two source-destination pairs in a simple grid topology. We found that BMSR significantly outperforms DSR and investigated the effect of balancing on packet collisions and IFQ overflows. Our results indicate that although multipath balancing does not significantly reduce the effect of packet collisions, bottleneck formation is avoided and a reduction in IFQ overflows is achieved. As a secondary performance criterion we evaluated the average delay of packets that successfully reach their destinations. Due to the small size of the network that was simulated, however, the results on packet delay are inconclusive.

In a second set of experiments we varied the number and location of source-destination pairs and also compared the performance of BMSR to the performance of an idealized shortest-path routing algorithm. For all

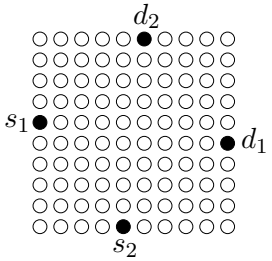


Figure 5.1: Simulation setup: two source-destination pairs (s_1, d_1) and (s_2, d_2) are placed at the boundary of a grid. Sources s_1 and s_2 send data packets at constant rate to their respective destination nodes d_1 and d_2, so that the direct connections between both pairs approximately form a cross. Each node may communicate with the nodes beside, above or below it.

CBR packet size (B)	256, 512, 1024, 2048	MAC bandwidth	1 Mbit
Propagation model	Two-ray ground	Max. IFQ length	50
MAC protocol	802.11 with RTS/ CTS	Max. route length	22
Network size	2160 m × 2160 m	Node count	100
CBR data rate	160 Kbit/s	Balancing setup	500 s
Antenna type	OmniAntenna	Simulation time	1500 s
Carrier sense range:	550 m	Trans. range:	250 m

Table 5.1: The parameters used in the first set of BMSR simulations.

the network setups we considered, BMSR outperforms DSR significantly. BMSR also performs better than the shortest-path algorithm when source-destination pairs are placed densely or the shortest-paths are not disjoint and there are only few sources simultaneously active in the network.

5.3.2 Two-pair cross setup

Consider a simple 10 by 10 square grid topology that places nodes a distance of 240 m apart. Since the default maximum transmission range in NS2 is 250 m, each node may communicate with the nodes beside, above or below it. Nodes that are diagonal neighbors, however, are not able to communicate directly. Regarding packet collisions, one needs to take into account that NS2 uses a pairwise interference model (for details see [64, 93]). It is important to note that if a transmitting node v is outside the carrier sensing range of a receiving node u, then the transmission from v does not interfere with the reception of another packet at u. However, a node may still receive a packet correctly even if another transmitter is close, provided the received signal strength of an incoming packet is sufficiently large compared to the interfering transmission. Figure 5.1 shows the network setup with two source-destination pairs and Table 5.1 gives the parameter values for this first set of simulations.

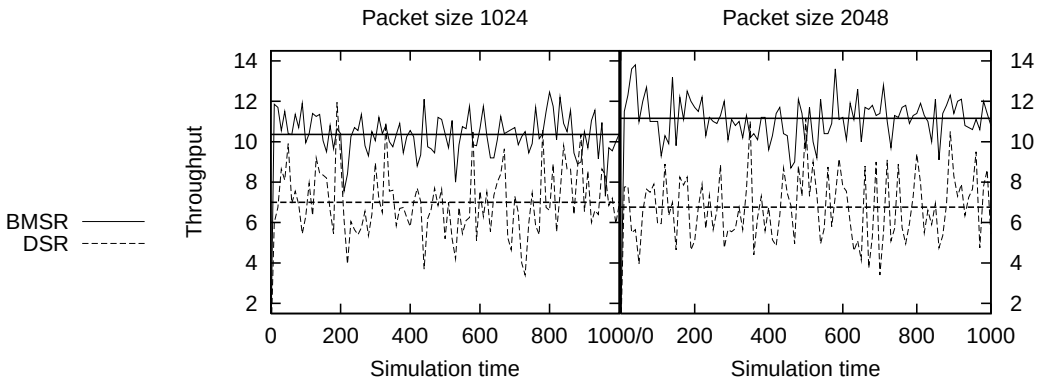


Figure 5.2: Average throughput of both source-destination pairs in KB/s versus simulation time for a single run of BMSR and DSR (setup phase of BMSR is omitted from the plot) for $I = 160$ and $\epsilon = 0.05$. The horizontal lines are averages over the entire simulation.

Throughput comparison

Figure 5.2 shows the performance of both algorithms over time. Comparing throughput for BMSR and DSR, one observes larger fluctuations for DSR. A major reason for the throughput stability of BMSR is that broken links do not cause route invalidation. Therefore, its performance is determined during the initial setup phase of the algorithm. To compensate for the fluctuations of DSR, we considered the throughput over 1000 s from the time when CBR transmissions have been initiated, i.e., after the termination of BMSR's setup phase, to compare both algorithms. For both packet sizes BMSR clearly outperformed DSR.

Distribution of network load

In addition to throughput, we studied the distribution of routes over the nodes by calculating the number of forwarded constant-bit-rate (CBR) packets at each node. For BMSR we expected most packets to be forwarded by nodes located near the center of the network, as these routes are shortest and the algorithm initially prefers shorter routes over longer ones. However, the central nodes should not be loaded much more heavily than those on slightly longer paths. A balanced network load should also reduce collisions and interface queue overflows in the network. Recall that the IFQ contains packets that are scheduled to be transmitted over the network interface. It was previously observed that IFQ drops at congested nodes lead to decreased performance of DSR [61]. Besides queue overflows, collisions of MAC layer control messages and CBR packets degrade performance. Although it is unreasonable to expect BMSR to be unaffected by collisions, BMSR should yield a more uniform distribution over the nodes, which essentially prevents the formation of bottlenecks.

Figure 5.3 shows simulation results for two CBR packet sizes; we use the following measures:

CBR packet load: the number of CBR packets sent by the MAC layer of the node. Note that there are in total 20000 and 10000 packets per source for packet sizes of 1024 and 2048 bytes respectively. Due to

drops and collisions this number does not necessarily correspond to the actual number of successfully forwarded messages. Data from sources was excluded from Figure 5.3 for clarity.

CBR packet collisions: the number of CBR MAC layer collisions caused by interference that occurred at each node, excluding the sources. Since the MAC layer uses a retransmission scheme, these numbers do not necessarily coincide with the number of dropped packets.

IFQ overflows caused by CBR packets: the number of IFQ overflow events that occurred at each node.

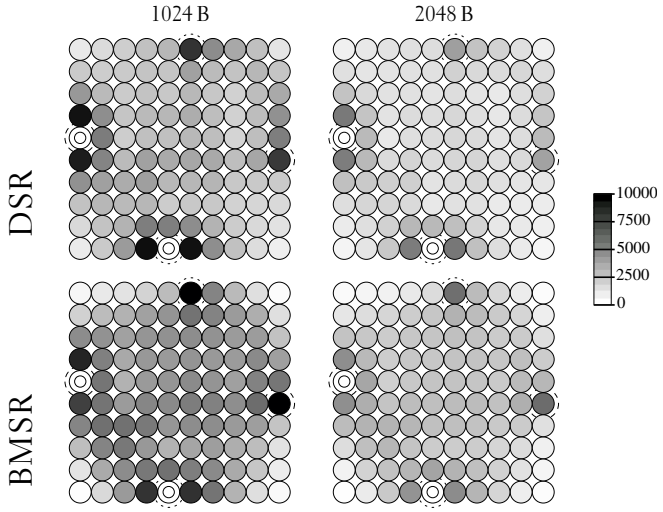
One might expect DSR to favor shorter routes, yielding an increased network load within the center of the grid, resulting in interference and low network throughput. BMSR should recognize areas of higher congestion and after initially selecting shorter routes, select routes that avoid the potentially congested areas. In Figure 5.3, one only observes minor differences for BMSR and DSR. Depending on the averaging of packet load over the rather long simulation run, the load for DSR appears to be well balanced. The reason is that within the congested network, rediscovered routes will typically be different from recently broken routes. There is a slightly higher utilization of boundary nodes by DSR, but the overall network load for BMSR is higher than for DSR, which can be explained by the higher throughput.

Due to overall higher load, BMSR encounters more collisions compared to DSR. A remarkable effect is the concentration in the quadrant of the network formed by the square with the sources on its diagonal. The effect is apparent for both algorithms and packet sizes, but emphasized for BMSR and 1024-byte packets. Nodes within this part of the network may be relaying packets from both sources in roughly opposite directions. Hence they have to transmit packets in more diverse directions than nodes within the vicinity of the destinations.

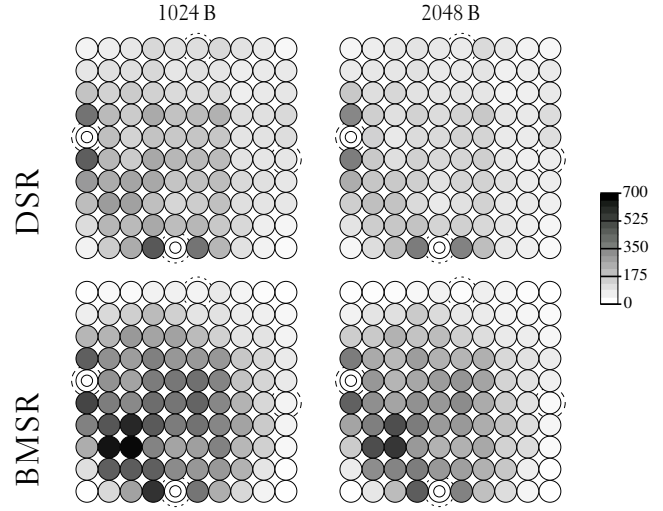
As the MAC layer transmission of a CBR packet includes a *request to send* (RTS)/*clear to send* (CTS) handshake, collisions are more likely to occur when nodes are transmitting in different directions than when the packets travel roughly in the same direction. DSR chooses paths that are close to shortest-hop paths. Therefore, subsequent packets for the same destination are less likely to interfere with each other. Figure 5.4 shows a situation when shortest paths avoid collisions but balancing may lead to an increase in collisions. The distribution of IFQ overflows follows basically the same principle. However, we observed a major difference between BMSR and DSR: the single-path routing of DSR led to the formation of bottleneck nodes due to congestion in the bottom left quadrant of the network. As DSR prefers shorter paths, such overloading of nodes is restricted to the band of nodes between the sources. The effect was stronger for smaller packet sizes, explained by the increased MAC layer overhead. BMSR showed hardly any IFQ overflows at all, except within the vicinity of the sources.

Effect of parameters I and ϵ

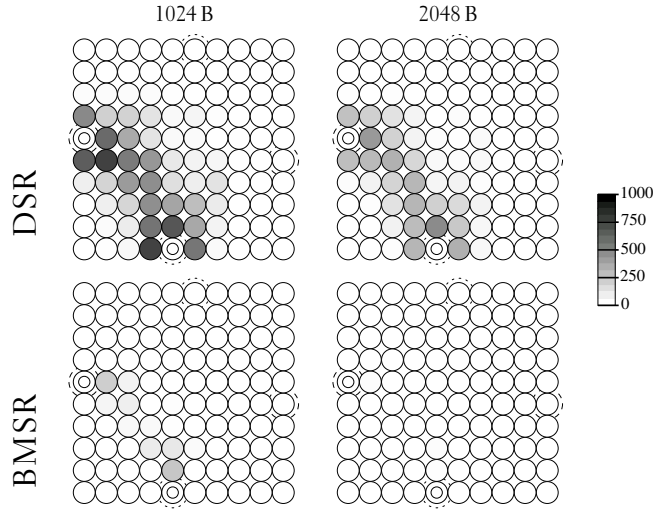
We also studied the effect of the I and ϵ parameters on the performance. The results are summarized in Figure 5.5 and are mostly as expected; already for



(a) CBR packet load



(b) CBR packet collisions



(c) IFQ overflows caused by CBR packets

Figure 5.3: Averages over five runs for the performance measures of DSR and BMSR for $I = 160$ iterations and $\epsilon = 0.05$ for two CBR packet sizes; variations were negligible. Source and destination nodes are indicated by dashed circles, where sources are marked by an additional solid inner circle.

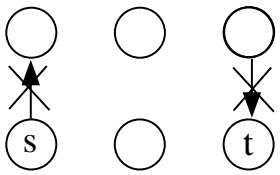


Figure 5.4: Network where nodes are located on a 2×3 grid; interference range is approx. twice the transmission range. The crossed arrows indicate transmissions that may interfere.

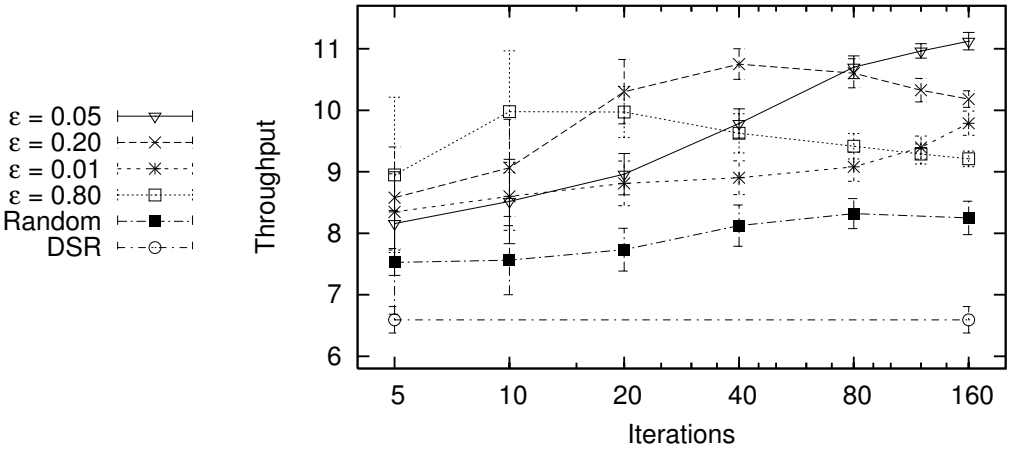


Figure 5.5: Performance in KB/s as a function of I and ϵ for CBR packet size 2048: BMSR, DSR, and random route selections of I routes between source-destination pairs, which are formed by selecting I routes on the basis of a random walk from sources to destinations. All values are averages over at least 15 repetitions (standard deviations shown). The legend ordering corresponds to the throughput value at $I = 160$.

a modest number of iterations we obtained throughput superior to DSR. There is a dependency of the throughput on ϵ and I: for larger values of ϵ, fewer iterations are needed to obtain a good throughput, but running a large number of iterations with a small value of ϵ yields a slightly better throughput.

An interesting observation can be made when considering the results of Figure 5.5. For a given value of ϵ, the throughput first increased rapidly as I increases but after reaching a maximum, the throughput started to decline gradually. We conjecture that this phenomenon is essentially caused by the difference between the two objectives of maximizing total flow versus maximizing the minimum routable demand, which is equivalent to minimizing maximum congestion. As the optimization algorithm progresses, the weights of the most congested edges will come to completely dominate the search for the least cost route from the source to the destination. With a large enough iterations count, the algorithm only seeks to balance the flow on those edges without any regard for the traffic situation in the rest of the network. We also ran the tests for other values of ϵ, but omit these from the figure for clarity; for $\epsilon \leq 0.05$, the peak performance had not yet been reached for $I = 160$.

In our experiments we performed considerably fewer iterations than rec-

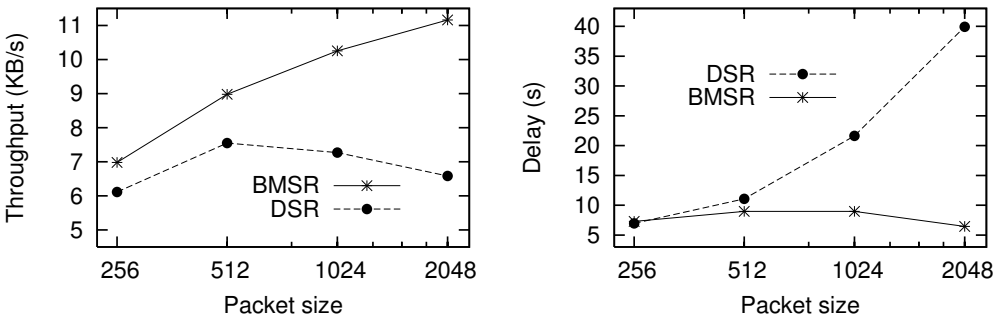


Figure 5.6: On the left, throughput in KB/s versus packet size, and on the right, delay in seconds versus packet size, for parameter values $I = 160$ and $\epsilon = 0.05$. All values are averages over at least 15 repetitions. Note the logarithmic scale for the packet size and that the CBR rate is 160 Kbit/s for all runs.

ommended by Algorithm YMMC. For small values of ϵ, in the first iterations the weight of each edge remains at approximately 1, and thus the paths found by BMSR will be essentially fewest hop paths. It seems plausible that instead of only optimizing the hop count, or only balancing the flow along the most congested edges, good results could be obtained by taking both factors into consideration. It is our conjecture that this is what happens when the number of iterations I is less than recommended by Algorithm YMMC.

Effect of packet size on throughput and delay

Figure 5.3 indicates that both BMSR and DSR perform differently for different packet sizes. Figure 5.6 shows throughput and packet delay for both BMSR and DSR for various packet sizes. The throughput performance of DSR seems to increase until a critical packet size, after which increasing the packet size further decreases DSR's performance. We assume this to be caused by the interdependence of the two main reasons of packet loss: collisions of CBR packets due to interference and IFQ overflows. Increasing packet size for a constant CBR rate reduces the number of packets and therefore the total MAC layer overhead. However, the time frame required for the transmission of a single packet increases correspondingly and retransmissions become more costly. Still the effect of losing larger packets due to IFQ overflows seems to outweigh the impact of collisions. We therefore conclude that increasing the packet size reduces the negative effect of collisions on throughput for BMSR, while DSR suffers from IFQ overflows due to network congestion.

As Figure 5.6 shows, in our experiments DSR packet delay grows nearly linearly with packet size, whereas BMSR shows a clearly lower, approximately constant delay. This is most likely due to the fact that after the initial setup phase BMSR uses a static routing scheme. The considered network size is evidently too small to expose the larger delay that BMSR experiences compared to DSR because of using longer routes on average. Since our main focus is on evaluating the degree of load balancing that can be achieved by the algorithm, we do not consider packet delay further.

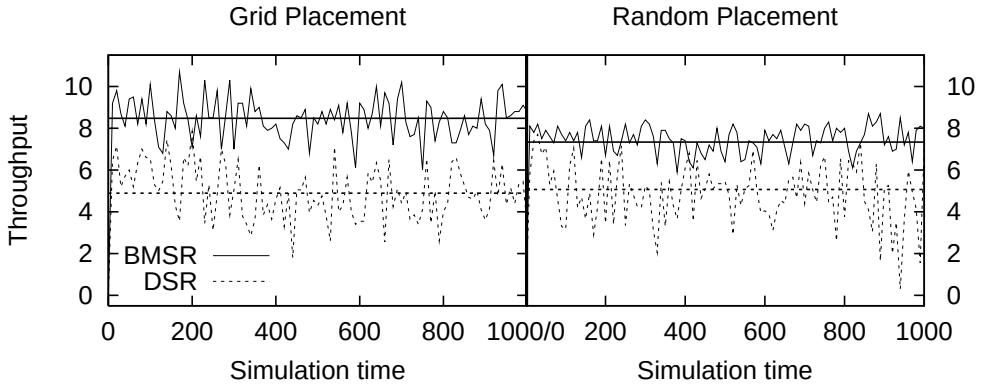


Figure 5.7: Average CBR throughput of both source-destination pairs in KB/s versus simulation time for a single run of DSR and BMSR for $I = 160$, $\epsilon = 0.05$ and packet size of 2048 bytes. The plot shows averages over 15 runs for both the grid and the random placement.

Random node placement

Assuming a random versus a grid deployment is intuitively a more natural assumption, since a grid setup requires the careful placement of nodes prior to the operation of the network. We now proceed to a setup where intermediate nodes are positioned randomly, while the position of source and destination nodes stays the same. In order to maintain connectivity, we roughly doubled the density of nodes within the network by decreasing the network dimensions by a factor of approximately $\sqrt{2}$, yielding a network size of 1530 m $\times$ 1530 m. All nodes different from source and destination nodes were placed uniformly at random in the square area. In order to estimate the effect of a random placement, we compared runs of the algorithms for the random setup to runs on a grid of the same dimensions. The grid setup used for comparison still corresponds to Figure 5.1 but the smaller distances between grid neighbors allow nodes to also communicate with diagonal neighbors.

Figure 5.7 shows throughput results obtained for the grid network and the random geometric graph for both routing algorithms. From the figure one observes that the random node placement did not affect the performance of either routing algorithm significantly. However, a slight performance decrease for BMSR is observable. This may be explained by increased interference caused by the larger density of nodes in the network, as BMSR does not take into account interference between radio links. In this sense, the grid placement is more favorable to BMSR since the node density is constant and relatively low throughout the network.

Due to the only minor decrease and and in order to reduce random effects, we focused further simulations on grid topologies. We also fixed $\epsilon = 0.05$ and $I = 160$, since these values showed the best average throughput over both source-destination pairs.

5.3.3 Multiple source-destination pairs

In order to develop an intuition on how BMSR scales with the number of source-destination pairs and to determine the effect of differing traffic patterns on its performance, we conducted a second series of experiments. We

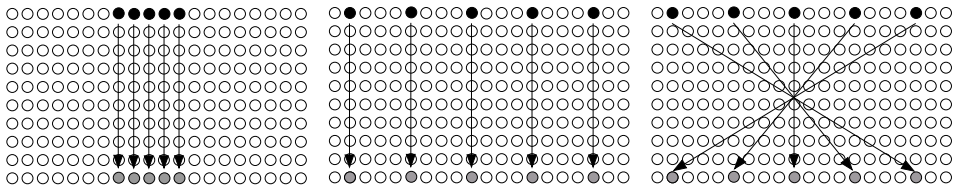


Figure 5.8: Different simulation setups; left: “dense”, middle: “sparse”, right: “twisted”.

Max. route length:	26	Node count:	200
CBR packet size (B)	2048	Network size:	2160 m × 4320 m
Source-destination pairs	1-5	BMSR parameters:	$I = 160,\ \epsilon = 0.05$

Table 5.2: The modified parameters for the second set of BMSR simulations.

increased the number of pairs and also varied their relative location compared to the other pairs in the network. The three different simulation setups are depicted in Figure 5.8. The changes in simulation parameters compared to Table 5.1 are summarized in Table 5.2. Note that the separation between grid neighbors is again 240 m, enabling nodes only to communicate with the nodes that are also their neighbors on the grid. The width of the grid was doubled to determine the spread of routes over the network.

We increased the maximum number of hops in each route to account for the larger network size. This value, which is a constant given in the NS2 implementation of DSR, determines the spreading of routing-control packets, such as RREQ, in the network as well as the connectivity between nodes. However, NS2 resource consumption forced us to choose a rather conservative value of 26 hops. We then explored the performance of DSR, BMSR and SPR, by which we refer to an idealized shortest path algorithm.

The SPR algorithm was initialized to use a route of minimum length from the source to the destination node for all packets during the simulation run, independent of route failures. In the case that there are multiple routes of minimum length, SPR picks any route from these at random. In this sense it behaves similar to BMSR, which determines routes in the setup phase of the algorithm. The algorithm was chosen to evaluate the benefit from choosing multiple routes over a single shortest route for a given network setup.

Densely placed pairs

We first considered the setup shown on the left of Figure 5.8. In this scenario, the sources are located directly opposite to their respective destinations and all sources are located next to each other on the boundary of the grid. We refer to this situation as the “dense setup”.

Because the source and destination nodes are packed closely together, there exists a large number of nodes to the left and to the right, respectively, of the leftmost and rightmost source and destination nodes. In this setup, one can observe from Figure 5.9 that BMSR outperformed both DSR and

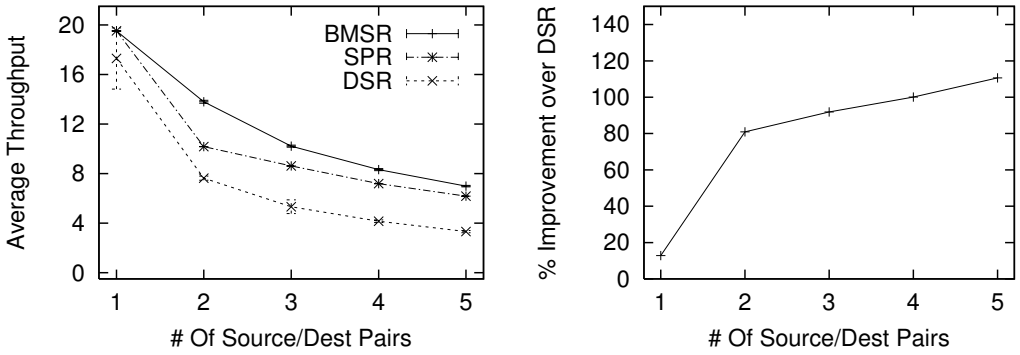


Figure 5.9: Average throughput of multiple densely-placed source-destination pairs in KB/s using routing algorithms DSR, BMSR, and SPR. Errorbars represent the standard deviation over 15 repetitions.

SPR. The latter two routing methods tend to use routes that are close to each other, so that up to three shortest paths can interfere with each other. Route interference, in turn, causes collisions and packet drops due to IFQ overflows.

In the dense setup, as the number of source-destination pairs increases, shortest-path routes can be expected to be the most favorable in terms of causing less interference than any other choice of routes. Thus the comparative advantage of BMSR with respect to SPR decreases. This trend is observable in Figure 5.9. However, as densely packed routes are subject to a higher rate of collisions and retransmissions, SPR also showed a decreasing performance for an increasing number of CBR pairs.

Sparsely placed pairs

The previously considered dense setup does not provide the nodes with non-interfering routes to their respective destination when there is more than one pair. In order to evaluate the benefit of balancing the load over multiple routes, a setting where shortest paths do not interfere was deemed to be more informative. Therefore, we generated a setting where source and destination nodes are separated by three intermediate nodes, thereby eliminating interference among shortest-path routes. The middle picture shown in Figure 5.8 depicts this setup. Not surprisingly, the throughput for SPR remained roughly constant for an increasing number of source-destination pairs, as shown in Figure 5.10.

In this setup, BMSR was able to take advantage of the additional nodes between adjacent shortest-paths and shows an increased throughput compared to the dense placement of source-destination pairs, while DSR did not perform significantly better than for the dense setup. As DSR heavily relies on cached routing information, which is updated by routes overheard from neighbors or taken from forwarded packets, nearby sources tend to share parts of their routes over the long run.

Effect of a large number of route intersections

In order to evaluate the performance of BMSR for a scenario with a large number of route-intersections, we created a worst-case scenario for route in-

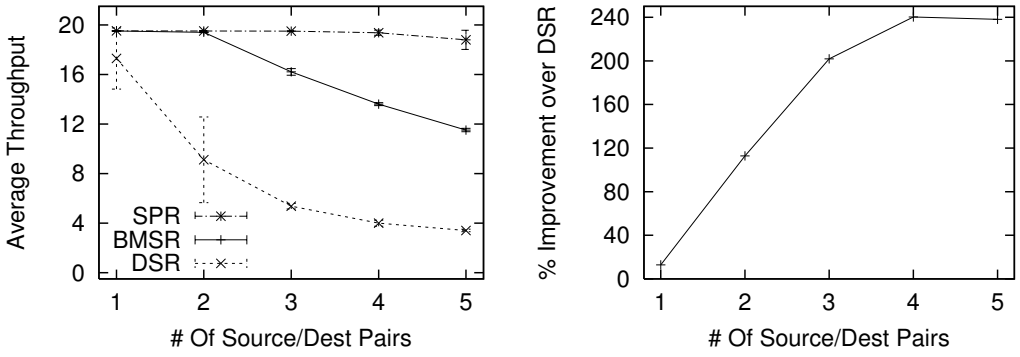


Figure 5.10: Average throughput of multiple sparsely-placed source-destination pairs in KB/s using routing algorithms DSR, BMSR, and SPR. Errorbars represent the standard deviation over 15 repetitions.

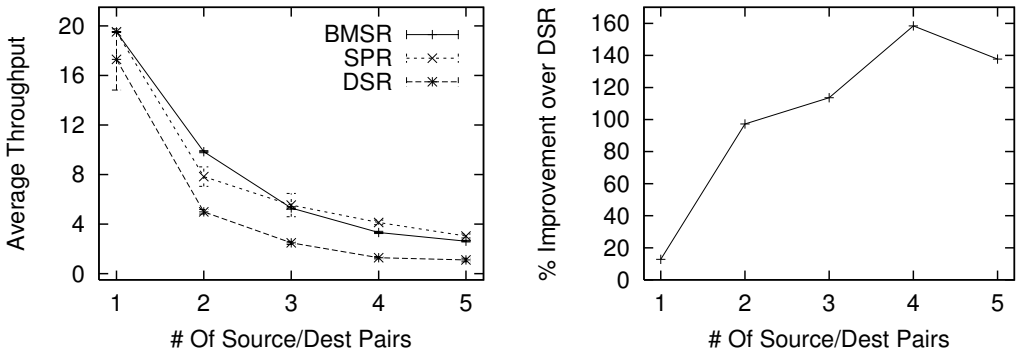


Figure 5.11: Average throughput of multiple sparsely-placed source-destination pairs with twisted destination arrangement in KB/s using routing algorithms DSR, BMSR, and SPR. Errorbars represent the standard deviation over 15 repetitions.

tersections by 'twisting' the aforementioned setup. The right side of Figure 5.8 depicts the resulting network. It is easy to see that the number of pairwise route intersections is $n(n-1)/2$, where n is the number of source-destination pairs.

Figure 5.11 shows performance results obtained for this network setup. It is interesting to see that BMSR performed very similarly to SPR. This can be explained by the fact that each route necessarily crosses any other route, which causes collisions and congestion at intermediate nodes. However, due to the spreading achieved especially for the inner source-destination pairs, BMSR still outperformed SPR for a smaller number of sources. As the maximum source-route length restricts the choice of routes for the balancing algorithm, the degree of freedom for outer pairs was much smaller than for inner pairs. In fact, for five source-destination pairs, the outermost pair always routes over shortest paths, determined by the maximum source-route length of 26.

DSR again showed the least performance for this setup. Congested nodes and collisions due to the heavy load in the center of the network caused routes to fail repeatedly. These had to be rediscovered regularly, causing additional overhead of routing control messages.

6 NETWORK UTILITY MAXIMIZATION WITH PATH CONSTRAINTS

In the previous chapter, we discussed a multipath routing algorithm that is based on a linear optimization problem, which assumes a given demand of routable traffic at each source node in the network. We now consider a situation, when sources have no predetermined traffic demand but optimize a *utility function* that corresponds to their contentment with the current state of the network, independently of other sources. Since network resources are limited, sources still compete for them. We approach the problem in the context of *network utility maximization* [73, 74], which has been applied previously to a large number of related problems.

Besides introducing utility functions to the routing problem, we restrict sources to a maximum number of paths that may carry positive flow. The BMSR algorithm we proposed earlier explores longer paths as it proceeds and may obtain a new path in every iteration of the algorithm. When the number of iterations is small, this approach is feasible. In this chapter, however, we implement the constraint on the number of usable paths by integrating it into the problem formulation. We develop a distributed algorithm that is based on dual decomposition, show the optimality of a stationary solution, and compare the performance of the algorithm with a centralized branch-and-bound method. The branch-and-bound algorithm is based on the techniques described in Section 3.2.3.

A bound on the number of paths is a natural restriction in practice. For instance, in a wireless network connectivity typically changes often, and the amount of state information used for routing should therefore be kept reasonably low.

6.1 INTRODUCTION

The work by Kelly, Maulloo and Tan [73, 74] initiated an active line of research that is based on the idea of treating network protocols, such as TCP, as methods for solving an implicit global optimization problem based on continuous updates of local information. This methodology allows interesting insights into current network protocols but also provides means to develop new distributed algorithms for various network optimization problems.

The work initiated by [74] resulted in new algorithms for power assignment and transmission scheduling in wireless networks [129], joint optimization of network flow and resource allocation [151], multipath routing with capacity constraints [58, 84, 88, 105, 113] and distributed coordination of cooperating agents [115]. The majority of algorithms are derived by the dual-decomposition technique, which can be applied to certain types of problems to derive a decomposition into several subproblems that communicate via dual variables. For an overview of dual decomposition and its applications see, e.g., [29, 69] and the references therein. Further applications and extensions of network utility maximization are discussed in the thesis of Johansson [67]. For recent developments on *stochastic network utility maximization* see the survey by Yi and Chiang [153].

K	Maximum number of paths with non-zero flow in a feasible solution.
U_n	Utility function for pair (s_n, t_n).
$\mathcal{P}_n$	Set of all paths connecting source s_n to sink t_n.
$\mathcal{P}$	Set of all paths between sources and their respective sinks, that is $\mathcal{P} = \bigcup_n \mathcal{P}_n$; note that the $\mathcal{P}_n$ are pairwise disjoint.
S	Selection of paths, where $S = (P_1, \ldots, P_N)$ and each $P_n \subseteq \mathcal{P}_n$.
P_S	Set of selected paths, i.e., $P_S = \bigcup_n P_n$, where $S = (P_1, \ldots, P_N)$.
y_n	Total flow originating from source s_n.
x_p	Flow over path p.
r_p^v	Binary constant whose value 1 indicates that node v is on path p.
T	Set of terminals that includes all source and destination nodes, i.e., $T = \bigcup_n \{s_n, t_n\}$.

Table 6.1: Notation used in this chapter.

While problems similar to the one addressed here have been considered in the literature, network utility maximization problems with path constraints seem not to have received much attention. Lin and Shroff [84] consider a fixed set of paths and briefly outline how to incorporate finding alternate paths into their algorithm. The algorithm by Lestas and Vinnicombe [82] is based on the penalty function approach of [74]. Their algorithm is similar to the one discussed in this chapter, in the sense that it iteratively optimizes path-flow and path prices based on a set of selected paths, which is updated depending on price information. However, as the same authors note, the penalty function approach tends to distribute the flow over a large number of paths, which is undesirable in practice.

A large variety of network utility maximization (NUM) problems exist in the literature. Since there is no established definition of a NUM problem, let us fix the formulation we use here. Similarly to the multicommodity flow problem, we assume a directed graph $G = (V, E)$, which represents the communication graph of a wireless network, and N distinct source-destination pairs $(s_1, t_1), \ldots, (s_N, t_N)$, where s_n is the source and t_n is the destination. For simplicity we assume that edges are symmetric. For each pair (s_n, t_n) there exists a utility function $U_n : \mathbb{R}^+ \mapsto \mathbb{R}$ which is strictly concave, differentiable and increasing.

We are interested in finding a distributed algorithm for a routing problem in wireless ad hoc networks. Hence, instead of the more common edge-capacity constraints, in our model there exist node capacities $c : V \mapsto \mathbb{R}^+$, which may model transmission time, battery power, etc. The objective is to maximize the global network utility, which is the sum of the utilities of the total flow originating from source nodes, such that the total amount of flow received and sent by each node v does not exceed its capacity $c(v)$. We further assume that for each pair (s_n, t_n) there exists at least one path of non-zero capacity from s_n to t_n. Table 6.1 summarizes notation.

Associate a flow commodity with each of the pairs. Denote the set of all paths from source s_n to target t_n by $\mathcal{P}_n$ and let $\mathcal{P}$ be the set of all paths. Let T be the set of all *terminals*, i.e., source and destination nodes. By introducing a variable x_p for the flow on path p for all $p \in \mathcal{P}$, the problem without path

constraints can be formulated as follows:

$$\textbf{NUM} \quad \text{maximize} \quad \sum_{n=1}^{N} U_n(y_n)$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_n} x_p = y_n \quad \forall 1 \le n \le N \tag{6.1}$$

$$\sum_{p \in \mathcal{P}} 2\, r_p^v x_p \le c(v) \quad \forall v \in V \setminus T \tag{6.2}$$

$$\sum_{p \in \mathcal{P} \setminus \mathcal{P}_n} 2\, r_p^v x_p + \sum_{p \in \mathcal{P}_n} x_p \le c(v) \quad \forall 1 \le n \le N, v \in \{s_n, t_n\} \tag{6.3}$$

$$y_n \ge 0 \quad \forall 1 \le n \le N, \quad x_p \ge 0 \quad \forall p \in \mathcal{P}.$$

Constraint (6.1) requires that for the n-th commodity, the total flow equals the sum of the path flows. Equations (6.2) and (6.3) ensure that the total flow received and transmitted by any node does not exceed its capacity. The binary constants r_p^v equal 1 if node v lies on path p and $r_p^v = 0$ otherwise. The constraints take into account that a node v receives and transmits traffic if it lies on a path p, unless $p \in \mathcal{P}_n$ and $v \in \{s_n, t_n\}$, in which case v either receives or transmits.

The NUM problem as formulated above is a convex optimization problem with linear constraints and a potentially exponential number of variables. This formulation using path flows lends itself to a solution by a distributed algorithm, as we show in Section 6.2. However, a more common formulation employs variables for the flow of each commodity over a particular edge, leading to a polynomial number of variables. This edge-flow formulation of the problem can be solved efficiently by standard centralized convex optimization algorithms. We use an edge-flow representation later on to compare against our distributed algorithm.

Note that our model does not explicitly take interference into account. However, instead of node-capacity constraints as formulated by (6.2) and (6.3), which apply to each node separately, one could instead consider capacities for node neighborhoods $N_k^+(v)$ for $k \ge 1$. This approach is based on the assumption of a simple interference model, according to which nodes that are located within a distance of k hops from each other may interfere. A single-path routing algorithm based on this interference model using $k = 2$ was proposed by Vannier and Lassous [138].

In the context of NUM, one particular choice of utility functions leads to a flow allocation y that is *proportionally fair*. A feasible solution (x, y) is said to be proportionally fair if for any other feasible solution (x', y') the aggregated proportional change is either zero or negative, i.e.,

$$\sum_{n=1}^{N} \frac{y'_n - y_n}{y_n} \le 0.$$

For the choice of $U_n = \log(y_n)$ it is known that optimal solutions to the network utility maximization problem are exactly those that satisfy the proportional fairness condition [74].

As argued earlier, in practice one would like to restrict the number of nonzero flow paths used in a solution. We call the problem with additional path

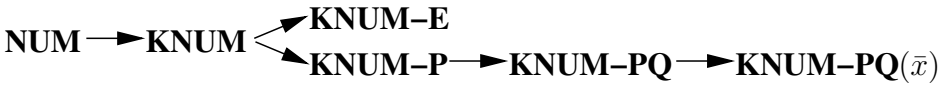


Figure 6.1: Transformations between problems applied in this chapter.

constraints KNUM and formulate it as follows.

$$\textbf{KNUM} \quad \text{maximize} \quad \sum_{n=1}^{N} U_n\left(y_n\right)$$

$$\text{s.t.} \quad \sum_{p \in P_n} x_p = y_n \quad \forall 1 \le n \le N \tag{6.4}$$

$$\sum_{p \in P_S} 2\, r_p^v x_p \le c(v) \quad \forall v \in V \setminus T \tag{6.5}$$

$$\sum_{p \in P_S \setminus P_n} 2\, r_p^v x_p + \sum_{p \in P_n} x_p \le c(v) \quad \forall 1 \le n \le N, v \in \{s_n, t_n\} \tag{6.6}$$

$$|P_n| \le K, \quad P_n \subseteq \mathcal{P}_n \quad \forall 1 \le n \le N \tag{6.7}$$

$$y_n \ge 0 \quad \forall 1 \le n \le N, \quad x_p \ge 0 \quad \forall p \in P_S,$$

In problem KNUM each source s_n maintains a selection of paths $P_n \subseteq \mathcal{P}_n$ and path rates x_p for all paths $p \in P_n$. We denote the complete path selection by S, where $S = (P_1, \ldots, P_N)$. Equation (6.7) limits the number of selected paths to at most K for any source-destination pair. The relation between NUM and KNUM is analogous to the relation between MMF and KMMF in Section 3.2.3. Figure 6.1 depicts the high-level transformations of problem NUM that we apply to obtain a distributed algorithm. The algorithm actually solves an instance of problem KNUM by iteratively solving KNUM-P, which uses a fixed selection of paths but is otherwise identical to KNUM. To prevent oscillations in the path rates, as also observed earlier in [144], we apply proximal minimization techniques to KNUM-P and obtain problem KNUM-PQ, which has the same optimum. Recall that proximal optimization was discussed in Section 3.3.2. Finally, we keep some of the variables in KNUM-PQ fixed while others remain variable. The resulting problem with $\bar{x}$ fixed is then denoted as KNUM-PQ($\bar{x}$).

When K is small, solving problem KNUM becomes computationally complex. Wang et al. [142] prove that NUM problems that consider a variable path selection and optimize the selection of a single path for each communication pair are NP-hard. However, because any source-destination flow can be decomposed into at most $|E|$ paths and cycles [1], if one allows a large number of paths, an optimal solution to KNUM is also an optimal solution to NUM. Hence, since an edge-flow based formulation of problem NUM can be solved efficiently, an optimal solution to KNUM for large K can be obtained by solving NUM and decomposing the resulting flow.

In practice the minimum number of paths required to decompose a given source-destination flow may be much smaller than $|E|$, but finding the minimum K is NP-hard [5]. In the rest of this chapter, we solve problem NUM by solving a corresponding KNUM instance while assuming that K is sufficiently large. Later we compare solutions obtained by our algorithm to solutions obtained by a centralized branch-and-bound algorithm. This algorithm

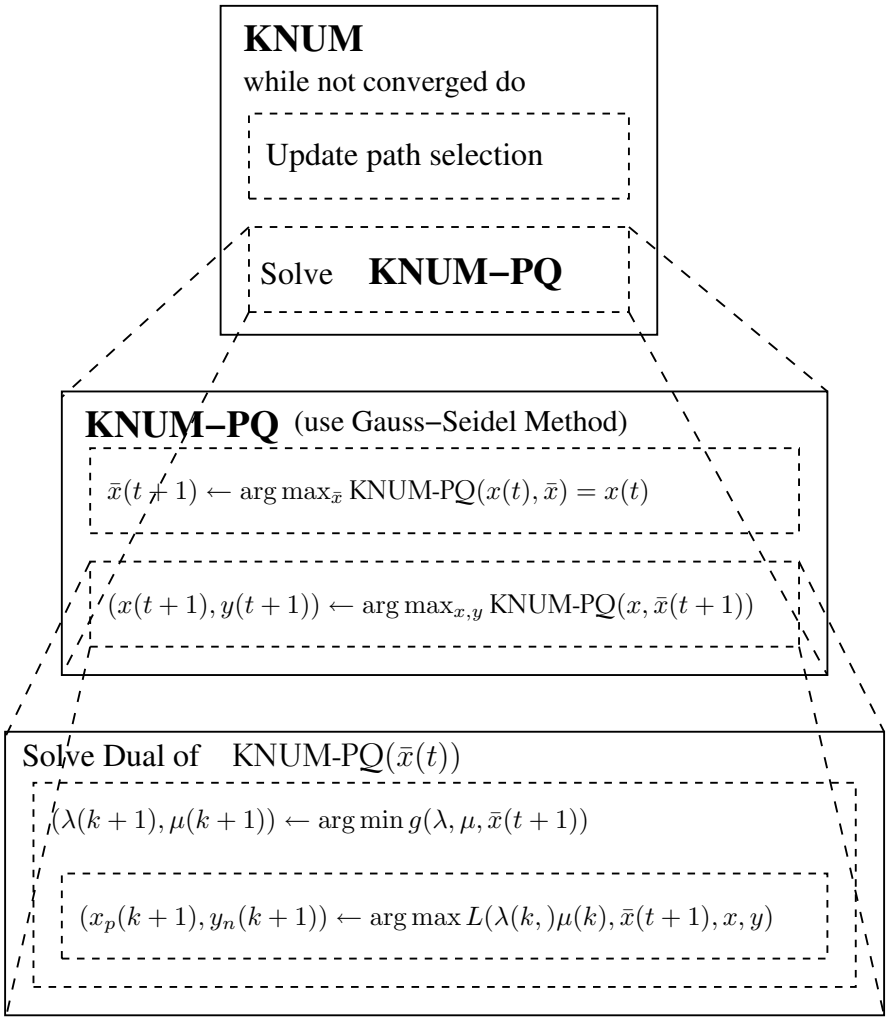


Figure 6.2: High-level outline of the proposed method.

solves an edge-flow representation of problem KNUM for a fixed value of K, which is an instance of problem KNUM-E to be defined later. By varying K we can thus determine the minimum number of paths required to obtain an optimal solution to problem KNUM with the same objective value as an optimal solution to NUM. Our results indicate that for smaller networks our algorithm does not require many more paths to reach convergence and solve NUM compared to the centralized branch-and-bound algorithm.

6.2 DISTRIBUTED ALGORITHM

We now decompose problem KNUM into a series of subproblems that are obtained by keeping some variables fixed while optimizing over the remaining free variables. Using the technique of dual decomposition we thus obtain a distributed algorithm for the original problem. Figure 6.2 depicts the high-level structure of the decomposition.

6.2.1 Fixed selection of paths

Using the notation in Table 6.1 we first consider a fixed selection of paths $S = (P_1, \ldots, P_N)$. When only optimizing the path-rates for the fixed selection, the problem becomes

$$\textbf{KNUM-P} \quad \text{maximize} \quad \sum_{n=1}^{N} U_n(y_n)$$

$$\text{s.t.} \quad \sum_{p \in P_n} x_p = y_n \quad \forall 1 \le n \le N \tag{6.8}$$

$$\sum_{p \in P_S} 2\, r_p^v x_p \le c(v) \quad \forall v \in V \setminus T \tag{6.9}$$

$$\sum_{p \in P_S \setminus P_n} 2\, r_p^v x_p + \sum_{p \in P_n} x_p \le c(v) \quad \forall 1 \le n \le N, v \in \{s_n, t_n\} \tag{6.10}$$

$$y_n \ge 0 \quad \forall 1 \le n \le N, \quad x_p \ge 0 \quad \forall p \in P_S.$$

Here the difference between KNUM and KNUM-P is that in the latter the selection of paths is fixed. One observes that the objective function of KNUM-P is strictly concave in y_n but not in x_p. Hence, the dual function may not be differentiable and applying a subgradient method may result in oscillations [144]. Instead of a subgradient method, we apply proximal optimization described in Section 3.3.2 and introduce additional variables $\bar{x}$ and a quadratic term to the objective function. The same technique has been applied previously to NUM problems for fixed multipath routing, e.g., by Lin and Shroff [85] and Wang et al. [144].

Introducing the $\bar{x}$ results in an objective function that is strictly concave in x and y for fixed $\bar{x}$ and leads to a differentiable dual function [14, p. 669]. As a minor technicality, we also substitute each equality constraint by two inequality constraints. The modified problem then becomes

$$\textbf{KNUM-PQ} \quad \text{maximize} \quad \sum_{n=1}^{N} U_n(y_n) - \frac{1}{2D} \sum_{p \in P_S} (x_p - \bar{x}_p)^2 \tag{6.11}$$

$$\text{s.t.} \quad \sum_{p \in P_n} x_p - y_n \le 0, \quad y_n - \sum_{p \in P_n} x_p \le 0 \quad \forall 1 \le n \le N \tag{6.12}$$

$$\sum_{p \in P_S} 2\, r_p^v x_p \le c(v) \quad \forall v \in V \setminus T \tag{6.13}$$

$$\sum_{p \in P_S \setminus P_n} 2\, r_p^v x_p + \sum_{p \in P_n} x_p \le c(v) \quad \forall 1 \le n \le N, v \in \{s_n, t_n\} \tag{6.14}$$

$$y_n \ge 0 \quad \forall 1 \le n \le N, \quad x_p \ge 0, \quad \bar{x}_p \ge 0 \quad \forall p \in P_S, \tag{6.15}$$

where D is a positive scaling constant. Problem KNUM-PQ then matches KNUM-P except for the quadratic term, which equals zero for any optimal solution. That is, if (x^*, y^*) is an optimal solution for KNUM-P, then $(x^*, \bar{x}^*, y^*)$ with $\bar{x}^* = x^*$ is an optimal solution for KNUM-PQ.

We now apply the Gauss-Seidel method described in Section 3.3.2 to KNUM-PQ. Consider two parameterized versions of KNUM-PQ, one with the value of $\bar{x}$ fixed and (x, y) variable and one with $\bar{x}$ variable and

(x, y) fixed. In every iteration of the Gauss-Seidel method, we first optimize KNUM-PQ with x, y variable and $\bar{x}$ fixed. Then, we solve KNUM-PQ with $\bar{x}$ variable and x and y fixed to their values determined before. The second step is simple, as the objective function (6.11) is maximized at $\bar{x} = x$ for fixed x and y, so we only need to consider the first step.

Denote the problem with $\bar{x}$ fixed by KNUM-PQ($\bar{x}$). Since the objective function in KNUM-PQ($\bar{x}$) is strictly concave and because all constraints are linear, the problem has a unique solution. It follows that the corresponding dual function is differentiable, and we can apply a gradient method to the dual problem. Based on the Lagrange multipliers and their utility functions U_n, sources can then determine their path rates, which is the essential idea of dual decomposition. We formulate the (partial) Lagrangian for KNUM-PQ($\bar{x}$) by introducing multipliers λ_v for the node capacity and μ_n^+ and μ_n^- for the total-flow constraint of source s_n.

$$\begin{aligned} L(x, \bar{x}, y, \lambda, \mu^+, \mu^-) &= \sum_{n=1}^{N} U_n(y_n) - \frac{1}{2D} \sum_{p \in P_S} (x_p - \bar{x}_p)^2 \\ &- \sum_{v \in V \setminus T} \lambda_v \Big(\sum_{p \in P_S} 2\, r_p^v x_p - c(v) \Big) - \sum_{n=1}^{N} (\mu_n^+ - \mu_n^-) \Big(y_n - \sum_{p \in P_n} x_p \Big) \\ &- \sum_{1 \le n \le N} \sum_{v \in \{s_n, t_n\}} \lambda_v \Big(\sum_{p \in P_S \setminus P_n} 2\, r_p^v x_p + \sum_{p \in P_n} x_p - c(v) \Big) \end{aligned} \tag{6.16}$$

Let us denote by $\rho_v(x)$ the *load* of a node v:

$$\rho_v(x) = \begin{cases} \sum_{p \in P_S \setminus P_n} 2\, r_p^v x_p + \sum_{p \in P_n} x_p & \forall 1 \le n \le N, v \in \{s_n, t_n\} \\ \sum_{p \in P_S} 2\, r_p^v x_p & \forall v \in V \setminus T \end{cases}$$

Based on this definition we can rewrite (6.16) as

$$\begin{aligned} L(x, \bar{x}, y, \lambda, \mu^+, \mu^-) &= \sum_{n=1}^{N} U_n(y_n) - \frac{1}{2D} \sum_{p \in P_S} (x_p - \bar{x}_p)^2 \\ &- \sum_{v \in V} \lambda_v \Big(\rho_v(x) - c(v) \Big) - \sum_{n=1}^{N} (\mu_n^+ - \mu_n^-) \Big(y_n - \sum_{p \in P_n} x_p \Big). \end{aligned} \tag{6.17}$$

The value of the dual function $g(\lambda, \mu^+, \mu^-, \bar{x})$ to problem KNUM-PQ($\bar{x}$), i.e., KNUM-PQ with fixed $\bar{x}$, can then be obtained by maximizing the Lagrangian over x and y for the given dual values and $\bar{x}$. One obtains

$$g(\lambda, \mu^+, \mu^-, \bar{x}) = \sup_{x \ge 0, y \ge 0} \big\{ L(x, \bar{x}, y, \lambda, \mu^+, \mu^-) \big\} \tag{6.18}$$

and the dual problem becomes

$$\text{minimize} \quad g(\lambda, \mu^+, \mu^-, \bar{x}) \tag{6.19}$$

$$\text{subject to} \quad \lambda \ge 0, \quad \mu^+ \ge 0, \quad \mu^- \ge 0. \tag{6.20}$$

Recall that $\bar{x}$ is constant. Since the dual function is differentiable, we can form the following partial derivatives for the dual function $g(\lambda, \mu^+, \mu^-, \bar{x})$.

$$\frac{\partial g}{\partial \lambda_v} = c(v) - \rho_v(\tilde{x}), \ \frac{\partial g}{\partial \mu_n^+} = \sum_{p \in P_n} \tilde{x}^p - \tilde{y}_n, \ \frac{\partial g}{\partial \mu_n^-} = \tilde{y}_n - \sum_{p \in P_n} \tilde{x}^p, \tag{6.21}$$

$$\begin{aligned} \text{where} \quad \tilde{x}^p &= \arg\max_{x \geq 0} L(x, \bar{x}, y, \lambda, \mu^+, \mu^-) \\ \tilde{y}_n &= \arg\max_{y \geq 0} L(x, \bar{x}, y, \lambda, \mu^+, \mu^-). \end{aligned} \tag{6.22}$$

We would like to use the projected gradient method of Section 3.3.2 for finding a maximizer of the dual function. However, in each iteration we need to solve the primal subproblem given by (6.22). Fortunately, in this case the maximizers $\tilde{x}$ and $\tilde{y}$ can be determined in a single step, by making some mild assumptions on the utility functions U_n. Denote by $\gamma_p(\lambda)$ the *cost* of a path $p \in P_n$ as seen by the source-destination pair (s_n, t_n). More precisely, let

$$\gamma_p(\lambda) = \lambda_{s_n} + \lambda_{t_n} + \sum_{v \in V \setminus \{s_n, t_n\}} 2\, \lambda_v r_p^v.$$

Then we can once more rewrite (6.16), this time by grouping together the terms related to each source-destination pair and each path. We obtain

$$\begin{aligned} L(x, \bar{x}, y, \lambda, \mu^+, \mu^-) = \sum_{n=1}^{N} \Bigg[& U_n(y_n) - (\mu_n^+ - \mu_n^-) y_n \\ & - \sum_{p \in P_n} \left(\frac{1}{2D} (x_p - \bar{x}_p)^2 - \big(\mu_n^+ - \mu_n^- - \gamma_p(\lambda)\big) x_p \right) \Bigg] + \sum_{v \in V} \lambda_v c(v). \end{aligned} \tag{6.23}$$

From (6.23) it becomes clear that the primal subproblem decomposes into decisions for each path and each source-destination pair. For simplicity let us assume that the derivative U_n' of U_n is invertible. Then by elementary differentiation we can rewrite (6.22) as

$$\begin{aligned} \tilde{x}^p &= \bar{x}_p + D\big(\mu_n^+ - \mu_n^- - \gamma_p(\lambda)\big) \\ \tilde{y}_n &= {U_n'}^{-1}(\mu_n^+ - \mu_n^-) \end{aligned} \tag{6.24}$$

and thus obtain the following update rules for all $1 \leq n \leq N$ and $v \in V$ for the projected gradient method applied to the dual problem (6.19):

$$x_p(k) = \left[\bar{x}_p + D\left(\mu_n^+(k) - \mu_n^-(k) - \gamma_p\big(\lambda(k)\big) \right) \right]^+ \quad \forall p \in P_n \tag{6.25}$$

$$y_n(k) = [{U_n'}^{-1}(\mu_n^+(k) - \mu_n^-(k))]^+ \tag{6.26}$$

$$\lambda_v(k+1) = [\lambda_v(k) + \alpha(\rho_v(x(k)) - c(v))]^+ \tag{6.27}$$

$$\mu_n^+(k+1) = \left[\mu_n^+(k) + \alpha \left(y_n(k) - \sum_{p \in P_n} x_p(k) \right) \right]^+ \tag{6.28}$$

$$\mu_n^-(k+1) = \left[\mu_n^-(k) - \alpha \left(y_n(k) - \sum_{p \in P_n} x_p(k) \right) \right]^+, \tag{6.29}$$

where k is the iteration index, α is a sufficiently small step size, $[\cdot]^+$ is the projection on the non-negative orthant, and ${U_n'}^{-1}$ is the inverse function of the derivative of U_n. Note that the update rules contain an explicit solution for the primal variables given the corresponding dual variables and $\bar{x}_p$. Assuming that each source s_n is aware of the cost $\gamma_p(\lambda(k))$ for each path $p \in P_n$, these

update equations are suitable for a distributed implementation. The remaining dual variables, $\mu_n^+(k)$ and $\mu_n^-(k)$ can be maintained at the sources.

Since we assume that for any source-destination pair there exists a path with positive capacity, it can easily be seen that Slater's condition holds, and thus also strong duality holds. The results for the projected gradient method then imply that for a sufficiently small step size α the solutions updated according to (6.25)–(6.29) converge to an optimal solution of KNUM-PQ($\bar{x}$) for any given $\bar{x}$ and choice of paths. It follows from the results of the Gauss-Seidel method that a stationary point of the update equations for variable $\bar{x}$ is optimal for problem KNUM-PQ for the given selection of paths. We now turn to the case when path selections are updated iteratively.

6.2.2 Adaptive path selection

Previously we assumed that the problem is solved for a fixed path selection. Now we want to let the sources choose paths based on the Lagrangian multipliers for the capacity constraints. Suppose one had chosen the path selection $S_c = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$, which consists of the set of all paths from each source to its designated destination node, prior to solving KNUM-PQ. It is easy to see that KNUM-PQ with selection S_c is equivalent to NUM. Although in general $|\mathcal{P}_n|$ can be exponential in $|V|$, in an optimal solution only minimum cost paths may carry positive flow. This follows directly from the Lagrangian (6.23) and the dual function (6.18).

Based on this observation, we add an additional layer on top of the Gauss-Seidel algorithm that iteratively updates a fixed selection of paths. Each source regularly initiates a shortest-path computation and adds the resulting path to its collection. If the addition causes the current number of paths to exceed the limit K, then the source removes a maximum cost path from its selection. After the update has been performed, new values for the primal and dual variables are determined by solving KNUM-PQ for the modified path selection. We first describe the algorithm in the centralized setting. The implementation details of the distributed algorithm are discussed below.

More precisely, denote the path selection in iteration i of the algorithm by $S^i = (P_1^i, \ldots, P_N^i)$ and let $(\tilde{x}, \tilde{\tilde{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$ be an optimal solution to KNUM-PQ with selection S^i. Let γ_p^i be the cost of path p and let ρ_v^i be the load of node v for the optimal solution. We update the selection according to the following rule.

Let $p^{\min}$ be a path of minimum cost from a source s_n to its destination t_n resulting from the current values for the dual variables. For the same source-destination pair, let $p^{\max}$ be a path of maximum cost among all paths in P_n^i maintained by s_n. If the maximum cost path in P_n^i is not unique, ties among these are broken by choosing any path that has the lowest amount of flow. Then P_n^{i+1} is obtained from P_n^i by adding $p^{\min}$, and if that would increase the number of paths in P_n^{i+1} beyond K, then one removes $p^{\max}$:

$$P_n^{i+1} = \begin{cases} (P_n^i \setminus \{p^{\max}\}) \cup \{p^{\min}\} & \text{if } |P_n^i \cup \{p^{\min}\}| > K \\ P_n^i \cup \{p^{\min}\} & \text{otherwise.} \end{cases} \tag{6.30}$$

Algorithm CENDD describes the complete method outlined above. In order to prove that a stationary point $(\tilde{x}, \tilde{\tilde{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$, which satisfies (6.12)–

Algorithm CENDD: Centralized dual-decomposition for KNUM

initially:
for $1 \leq n \leq N$ **do** $P_n \leftarrow \{p_n\}$, where p_n is a shortest hop-count path;
pick some initial solution $(x(0), \bar{x}(0), y(0), \lambda(0), \mu(0))$;
while true **do**
  repeat
    repeat
      for $1 \leq n \leq N$ **do**
        for $p \in P_n^i$ **do**
          $x_p(k) \leftarrow [\bar{x}_p(t) + D(\mu_n^+(k) - \mu_n^-(k) - \gamma_p^i(\lambda(k)))]^+$;
        end
        $y_n(k) \leftarrow [{U_n'}^{-1}(\mu_n^+(k) - \mu_n^-(k))]^+$;
        $\mu_n^+(k+1) \leftarrow [\mu_n^+(k) + \alpha(y_n(k) - \sum_{p \in P_n} x_p(k))]^+$;
        $\mu_n^-(k+1) \leftarrow [\mu_n^-(k) - \alpha(y_n(k) - \sum_{p \in P_n} x_p(k))]^+$;
      end
      for $v \in V$ **do**
        $\lambda_v(k+1) \leftarrow [\lambda_v(k) + \alpha(\rho_v^i(x(k)) - c(v))]^+$;
      end
    until convergence of $(\lambda(k), \mu(k))$;
    let $(x(t), y(t), \lambda(t), \mu(t))$ be a stationary point;
    $\bar{x}(t+1) \leftarrow x(t)$;
  until convergence of $(x(t), \bar{x}(t), \lambda(t), \mu(t))$;
  let $(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$ be a stationary point;
  for $1 \leq n \leq N$ **do**
    let $p^{\min}$ be a minimum cost path from s_n to t_n;
    let $p^{\max}$ be a maximum cost path of minimum flow in P_n^i;
    if $|P_n^i \cup \{p^{\min}\}| > K$ **then** $P_n^{i+1} \leftarrow (P_n^i \setminus \{p^{\max}\}) \cup \{p^{\min}\}$;
    else $P_n^{i+1} \leftarrow P_n^i \cup \{p^{\min}\}$;
  end
end

(6.15), is an optimal solution for KNUM-PQ with variable paths, we use the general sufficiency conditions of Proposition 3 in Section 3.3.1. Note that these conditions need to be satisfied by all paths, including paths that were not part of the selection. Applied to the problem the conditions then read as follows.

$$\tilde{\lambda} \geq 0, \quad \tilde{\mu}^+ \geq 0, \quad \tilde{\mu}^- \geq 0 \tag{6.31}$$

$$\tilde{\mu}_n^+(\sum_{p \in P_n} \tilde{x}^p - \tilde{y}_n) = 0, \quad \tilde{\mu}_n^-(\tilde{y}_n - \sum_{p \in P_n} \tilde{x}^p) = 0, \quad \forall 1 \leq n \leq N \tag{6.32}$$

$$\tilde{\lambda}_v(\rho_v(\tilde{x}) - c(v)) = 0 \quad \forall v \in V \tag{6.33}$$

$$\tilde{y} = \arg\max_{y \geq 0} L(x, \bar{x}, y, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-) \tag{6.34}$$

$$(\tilde{x}, \tilde{\bar{x}}) = \arg\max_{(x,\bar{x}) \in \mathbb{R}_{\geq 0}^N \times \mathbb{R}_{\geq 0}^N} L(x, \bar{x}, y, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-), \tag{6.35}$$

Lemma 1. *For any given fixed selection of paths, any point* $(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$

that is a stationary point of the method given in Section 6.2.1 is an optimal solution for problem KNUM-PQ.

Proof. We first need to establish primal feasibility. It is easy to see that any stationary point necessarily satisfies constraints (6.12)–(6.15), since otherwise $(\tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$ would not be stationary due to the update rules (6.27)–(6.28). The solution is also dual feasible, since we use the projection rule and dual solutions only need to be non-negative. From (6.27) follows

$$\rho_v(\tilde{x}(k)) - c(v) \leq 0, \quad \rho_v(\tilde{x}(k)) < c(v) \Rightarrow \tilde{\lambda}_v = 0 \quad \forall v \in V,$$

so that the complementary slackness condition (6.33) holds true and analogously for $\tilde{\mu}^+$ and $\tilde{\mu}^-$ because of (6.28) and (6.29), respectively. We still need to show that $\tilde{x}$, $\tilde{\bar{x}}$, and $\tilde{y}$ maximize the Lagrangian with respect to $\tilde{\lambda}$, $\tilde{\mu}^+$, and $\tilde{\mu}^-$. However, since the Lagrangian (6.23) is strictly concave in y, we obtain

$$\frac{\partial L(x, \bar{x}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)}{\partial y_n} = 0$$

only for the unique optimal solution; this is equivalent to equality holding in (6.26). Further, at convergence one necessarily needs to have $\tilde{x} = \tilde{\bar{x}}$. For $\tilde{x}^p > 0$ we obtain from (6.25)

$$\tilde{\mu}_n^+ - \tilde{\mu}_n^- - \gamma_p(\tilde{\lambda}) = 0 \tag{6.36}$$

and hence we have

$$\frac{\partial L(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)}{\partial x_p} = 0, \qquad \frac{\partial L(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)}{\partial \bar{x}_p} = 0.$$

If $\tilde{x}^p = 0$, then from (6.25) one obtains $\tilde{\mu}_n^+ - \tilde{\mu}_n^- - \gamma_p(\tilde{\lambda}) \leq 0$ and hence we have

$$\frac{\partial L(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)}{\partial x_p} \leq 0, \quad \frac{\partial L(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)}{\partial \bar{x}_p} = 0.$$

Therefore, the conditions (6.34) and (6.35) also hold. □

Let us now consider the general case where paths are updated according to (6.30) and the algorithm performs as described in Algorithm CENDD.

Lemma 2. *When the algorithm reaches convergence, for each pair (s_n, t_n), all paths $p \in P_n$ with positive flow have cost $\gamma_p(\tilde{\lambda}) = \tilde{\mu}_n^+ - \tilde{\mu}_n^-$, which is smallest among all paths from s_n to t_n.*

Proof. The claim follows directly from (6.30) and (6.36). □

Proposition 5. *If Algorithm* CENDD *reaches convergence, the path selection and rate allocation forms an optimal solution to* KNUM. *Moreover, the path rates correspond to an optimal multicommodity flow for problem* NUM.

Proof. Consider two path selections S_A and S_B, where $S_A = (P_1, \ldots, P_N)$ is the selection obtained by Algorithm CENDD and $S_B = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ is the selection that consists of all paths between sources and destinations. Recall that KNUM-PQ using selection S_B is equivalent to NUM. Denote the solution for the primal and dual variables obtained by the algorithm by $(\tilde{x}, \tilde{\bar{x}}, \tilde{y}, \tilde{\lambda}, \tilde{\mu}^+, \tilde{\mu}^-)$. We can extend this solution to a solution for KNUM-PQ for paths in S_B by introducing new variables. For each path in $S_B \setminus S_A$, we choose $\tilde{x}_p = \tilde{\bar{x}}^p = 0$. We now prove that this new solution is stationary for the selection S_B, so that from Lemma 1 follows that it is optimal for problem NUM. Since we have not introduced additional paths with positive flow rate, it follows that the solution is also feasible and hence optimal for KNUM.

Fix any pair (s_n, t_n) and consider a path p from s_n to t_n, which was not previously in S_A. Hence, we have that $\tilde{x}_p = \tilde{\bar{x}}^p = 0$. Since p was not added to the selection, it follows from Lemma 2 that its cost is lower bounded by the cost of the paths in P_n, i.e., $\gamma_p(\tilde{\lambda}) \geq \tilde{\mu}_n^+ - \tilde{\mu}_n^-$ for all $p \in S_B \setminus S_A$. From the path-update rule (6.25) then results that $\tilde{x}^p = \tilde{\bar{x}}^p = 0$ is stationary. Furthermore, adding new path variables x_p and $\bar{x}_p$ with value zero does not affect the update equations for the other variables. Hence, any solution obtained by the algorithm is optimal for KNUM-PQ with selection S_B and, equivalently, for problem NUM. □

Corollary 1. *If the limit on the number of positive-flow paths K is too small to admit a solution to problem* KNUM*, which has the same objective value as an optimal solution for* NUM*, then Algorithm* CENDD *does not achieve convergence.*

6.2.3 Distributed implementation

We now outline how the operations in Algorithm CENDD can be performed locally. We assume that sources can monitor their total outgoing path flow and all nodes v, including non-terminals, can estimate their load ρ_v. If the traffic injected into the network by the sources can be approximated by a continuous network flow, then explicit message passing can be replaced by passive measurements for the estimation of ρ_v. The latter is clearly preferable, since it avoids the dissemination of additional control messages. For the implementation of the path update rule (6.30) one could basically use any distributed shortest-path algorithm, for example the algorithm proposed by Haldar [56], which has polynomial message complexity.

Although the data dependency of the variable updates is local, due to the decomposition of the dual, there are issues related to the timing of variable updates. More precisely, in Algorithm CENDD the primal variables x and y and the dual variables λ, μ^+, and μ^- have to converge before one can update $\bar{x}$. Therefore, a distributed implementation would either require synchronization between all source-destination pairs or a separation of time scales, which is usually assumed for distributed algorithms based on dual decomposition techniques. Here, to prevent this problem, we modify our algorithm as follows. We let the sources update the $\bar{x}$ at the same time scale as λ, μ^+, and μ^- but using a different step size β. In the limit of $\beta/\alpha \to 0$, the two versions of the algorithm are then equivalent. For a very similar algorithm in [84] it was shown that for a fixed set of paths convergence is guaranteed for

Algorithm DISTDD: Distributed dual-decomposition for KNUM

1 **while** true **do**
2 **if** is_source_node(s_n) **then**
3 **for** $p \in P_n$ **do**
4 $x_p(t) = [\bar{x}_p(t) + D(\mu_n^+(t) - \mu_n^-(t) - \gamma_p(\lambda(t)))]^+$;
5 $\bar{x}_p(t+1) = [(1 - \frac{\beta}{D})\bar{x}_p(t) + \frac{\beta}{D}x_p(t)]^+$;
6 **end**
7 $y_n(t) = [U_n'^{-1}(\mu_n^+(t) - \mu_n^-(t))]^+$;
8 $\mu_n^+(t+1) = [\mu_n^+(t) + \alpha(y_n(t) - \sum_{p \in P_n} x_p(t))]^+$;
9 $\mu_n^-(t+1) = [\mu_n^-(t) - \alpha(y_n(t) - \sum_{p \in P_n} x_p(t))]^+$;
10 **if** prim_local_conv($x_n(t), \bar{x}_n(t), y_n(t)$) and
11 dual_local_conv($\mu_n^+(t), \mu_n^-(t)$) and lambda_conv($\lambda(t)$) and
12 it_since_path_update $\geq$ min_it_before_path_update) **then**
13 let $p^{\min}$ be a minimum cost path from s_n to t_n;
14 let $p^{\max}$ be a maximum cost path of minimum flow in P_n^i;
15 **if** $|P_n^i \cup \{p^{\min}\}| > K$ **then**
16 $P_n^{i+1} \leftarrow (P_n^i \setminus \{p^{\max}\}) \cup \{p^{\min}\}$;
17 **else**
18 $P_n^{i+1} \leftarrow P_n^i \cup \{p^{\min}\}$;
19 **end**
20 **end**
21 **end**
22 $\lambda_v(t+1) = \lambda_v(t+1) = [\lambda_v(t) + \alpha(\rho_v(x(t)) - c(v))]^+$;
23 **end**

sufficiently small β. Note that our proof of optimality of a stationary solution, as presented in this section, still holds for the modified algorithm.

Similarly, there is a timing dependency between the updates of the path selection and the node prices λ. Since the cost of a path depends on λ, the path selection would only be modified after the prices have converged. In principle, we may add a convergence test to the distributed shortest-path algorithm. In practice, however, it may be more desirable to rely on the separation of time scales, choose a low update frequency and give the λ time to converge. In order to handle the potentially large number of nodes with zero prices, we add a small constant to each λ_v so that generally shorter paths are preferred over longer paths with equal cost.

Algorithm DISTDD shows the steps that are executed by each node continuously, where lines 3 to 20 are only executed by each source s_n. The pseudocode contains the functions prim_local_conv and dual_local_conv, which evaluate the convergence criterion for the primal and dual variables of source s_n, respectively. The function lambda_conv checks whether convergence of the node prices has been obtained, but this check may also be integrated in the shortest-path algorithm, as described above. Letting some sources perform path updates too frequently, however, could lead to unfairness among sources. Hence, we require a minimum number of iterations to pass between consecutive path updates.

Although our model is restricted to a fixed set of source-destination pairs, Algorithm DISTDD could be made adaptive to arrival and departure of pairs. Based on the update equations of the link prices, one may conjecture that the algorithm would be able to recover from these changes, provided they occur after a stationary point has been reached.

6.3 CENTRALIZED ALGORITHM

In order to evaluate our algorithm we now reformulate problem KNUM so that an optimal solution can be obtained by a centralized algorithm for any fixed value of K. A reformulation is necessary, since an explicit solution does not scale well due to the possibly large number of primal variables.

We consider an edge-flow representation of problem KNUM that results from a transformation similar to the one that was used in Section 3.2.3 to obtain problem KMMF-E from problem KMMF. Hence, we introduce binary variables δ_e^{nk}, for each edge e, that when set to 1 indicate that an edge e is available to route flow for the k-th path of pair (s_n, t_n). By $f_{(u,v)}^{nk}$ we denote the amount of flow over edge (u, v) that is part of the k-th path from s_n to t_n. Problem KNUM-E can then be formulated as the following convex mixed-integer nonlinear program.

$$\textbf{KNUM-E} \quad \text{maximize} \quad \sum_{n=1}^{N} U_n(y_n)$$

$$\text{s.t.} \quad y_n = \sum_{1\le k\le K} \sum_{e\in O(s_n)} f_e^{nk} - \sum_{e\in I(s_n)} f_e^{nk} \quad \forall 1 \le n \le N$$

$$-y_n = \sum_{1\le k\le K} \sum_{e\in O(t_n)} f_e^{nk} - \sum_{e\in I(t_n)} f_e^{nk} \quad \forall 1 \le n \le N$$

$$\sum_{e\in O(v)} f_e^{nk} - \sum_{e\in I(v)} f_e^{nk} = 0 \quad \forall 1 \le n \le N, 1 \le k \le K, v \in V \setminus \{s_n, t_n\}$$

$$\sum_{1\le n\le N} \sum_{1\le k\le K} \sum_{e\in I(v)\cup O(v)} f_e^{nk} \le c(v) \quad \forall v \in V$$

$$f_e^{nk} \le \delta_e^{nk} \cdot c(v) \quad \forall 1 \le n \le N, 1 \le k \le K, v \in V, \forall e \in O(v)$$

$$\sum_{e\in O(v)} \delta_e^{nk} \le 1, \quad \sum_{e\in I(v)} \delta_e^{nk} \le 1 \quad \forall 1 \le n \le N, 1 \le k \le K, v \in V$$

$$\sum_{e\in O(v)} \delta_e^{nk} - \sum_{e\in I(v)} \delta_e^{nk} = 0 \quad \forall 1 \le n \le N, 1 \le k \le K, v \in V \setminus \{s_n, t_n\}$$

$$\sum_{e\in I(s_n)} \delta_e^{nk} = 0, \quad \sum_{e\in O(t_n)} \delta_e^{nk} = 0 \quad \forall 1 \le n \le N, 1 \le k \le K$$

$$y_n \ge 0,\ f_e^{nk} \ge 0,\ \delta_e^{nk} \in \{0, 1\} \quad \forall 1 \le n \le N, 1 \le k \le K, e \in E,$$

The reformulation that results in problem KNUM-E is essentially the same that was applied in Section 3.2.3, where we replaced path flows by flow levels that can be expressed using flow-balance constraints. Since problems KNUM and KNUM-E are equivalent, we can thus obtain a solution to the original problem for any fixed K. Note that KNUM-E has $\mathcal{O}(KN|E|)$ variables, some of which are continuous while others are integer variables taking

binary values. By a simple modification to Algorithm BBKMMF, we can apply the same branch-and-bound method to KNUM-E. Here, the only difference is the subproblem to be solved at each point in the search, which is now a non-linear convex optimization problem.

6.4 NUMERICAL EXPERIMENTS

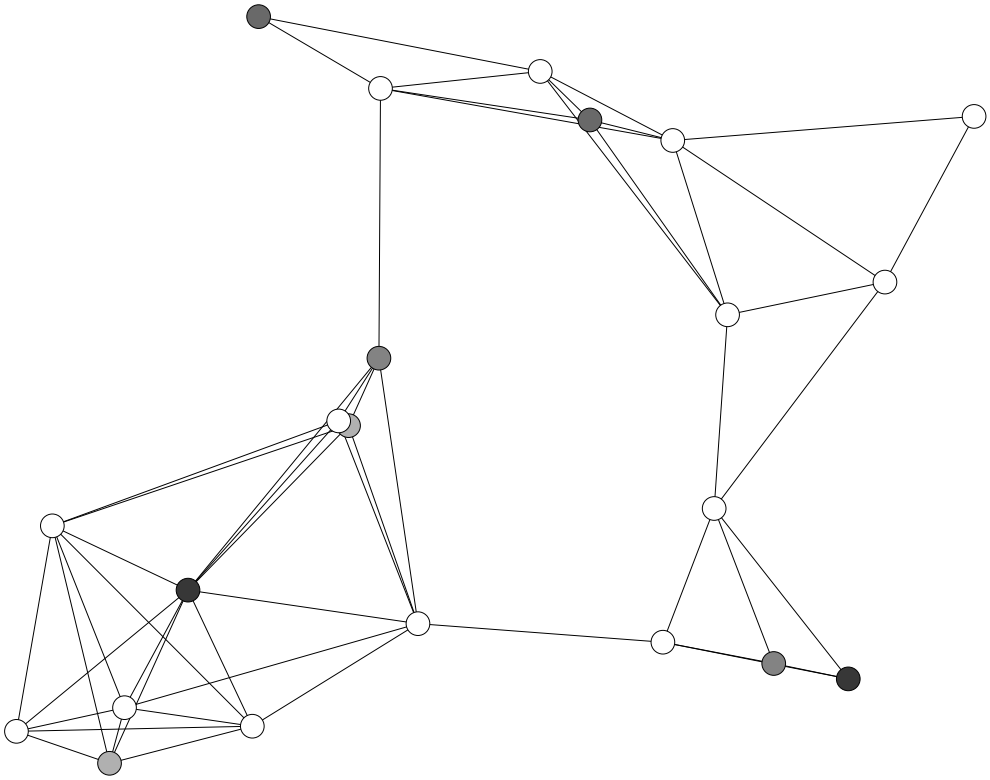
In Section 6.2.2 we showed that a stationary point of Algorithm CENDD is an optimal solution to the network utility maximization problem. When developing a distributed implementation, as given by Algorithm DISTDD, we needed to make some assumptions on the separation of timescales for variable updates. In order to decompose the original problem, we also had to assume that the parameter K in KNUM is sufficiently large to allow for a solution to be optimal for the corresponding NUM instance, while only using at most K paths with positive flow. Hence, the value of K and the number of iterations required for convergence, if convergence is achieved, were in the focus of attention when planning simulations for evaluating our proposed algorithm.

As a first step towards validating convergence and assessing the effect of parameter K on Algorithm DISTDD, we implemented a centralized version using Matlab. A future implementation using a network simulator, such as NS2, would be interesting to estimate the impact of concurrent path changes by sources and related effects. We would expect the resulting implementation to be somewhat similar to the BMSR algorithm described in the previous chapter.

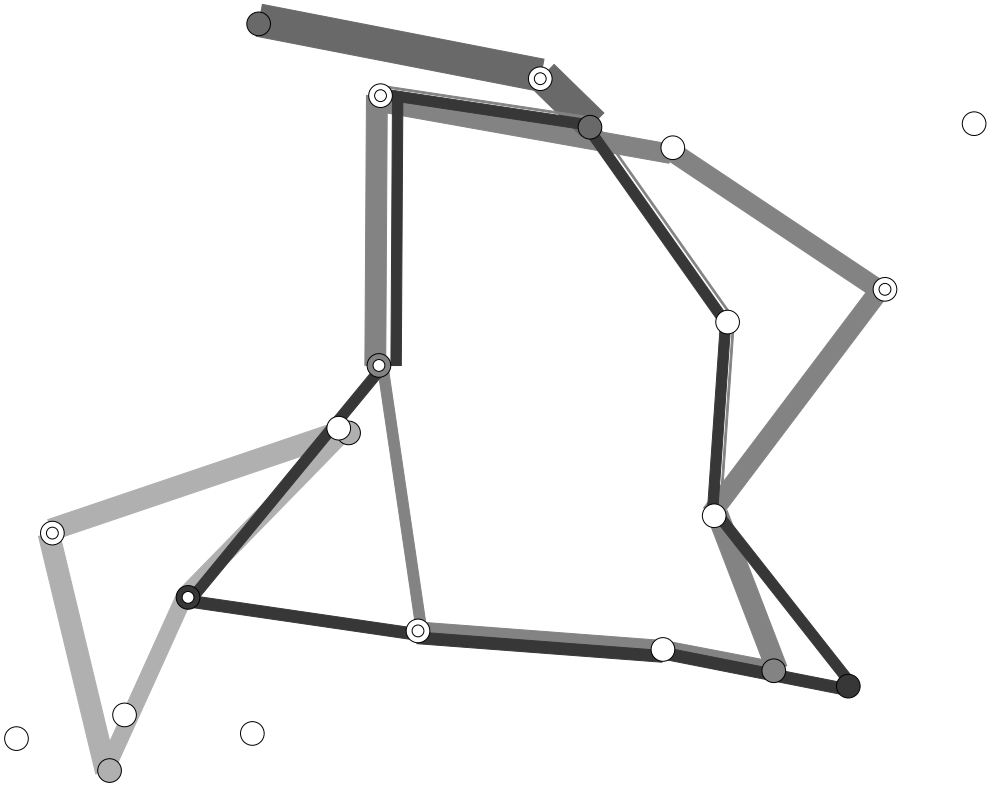
The problem instances of our simulations were generated as follows. We constructed a random disk graph $G(V, E)$ by scattering $|V|$ nodes, including source and destination nodes, in the square of unit dimensions using a uniform node distribution and setting $V = \{1, \ldots, |V|\}$. For any pair of nodes u and v there is an edge $(u, v) \in E$, if and only if their distance is not larger than the transmission range, which is chosen to be close to 0.31. Values for the node capacities $c(v)$ were drawn independently and uniformly at random from the interval $[5, 10]$. This process was repeated to generate a set of problem instances. We chose $N = 4$ and $U_n = \log(y_n)$, so that the problem involves finding a proportionally fair allocation of flow to the four sources.

For each instance and various choices for K, we first ran the branch-and-bound algorithm to obtain an optimal solution for the given value of K. In the case of an unconnected graph or if the branch-and-bound algorithm failed to terminate within a runtime limit of four hours on a 2GHz computer, we discarded the graph and generated a new one. We then ran our algorithm and observed how the path rates x_p evolve for each path $p \in P_n$ and pair (s_n, t_n). Based on experimentation, we chose the parameter values $\alpha = 10^{-3}$, $\beta = 10^{-2}$, and $D = 2^{-1}$.

Figure 6.3 depicts an example instance for a network with 22 nodes. Each pair is colored in a different shade of gray. Figure 6.4 shows a plot of the path rates for $K = 2$ and $K = 3$ for all pairs in the same instance. For $K = 2$ one can see that the path rates did not achieve convergence within $5 \cdot 10^4$ iterations and in fact oscillate. For parameter values $K \geq 3$, however, both x_p

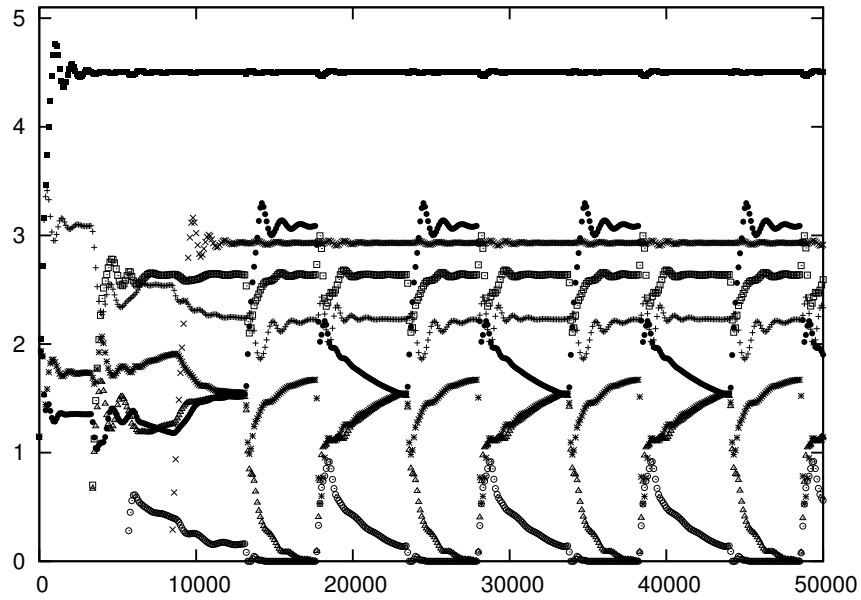


(a) Input Graph; source and destination nodes are drawn in the same color; edges are shown without direction.

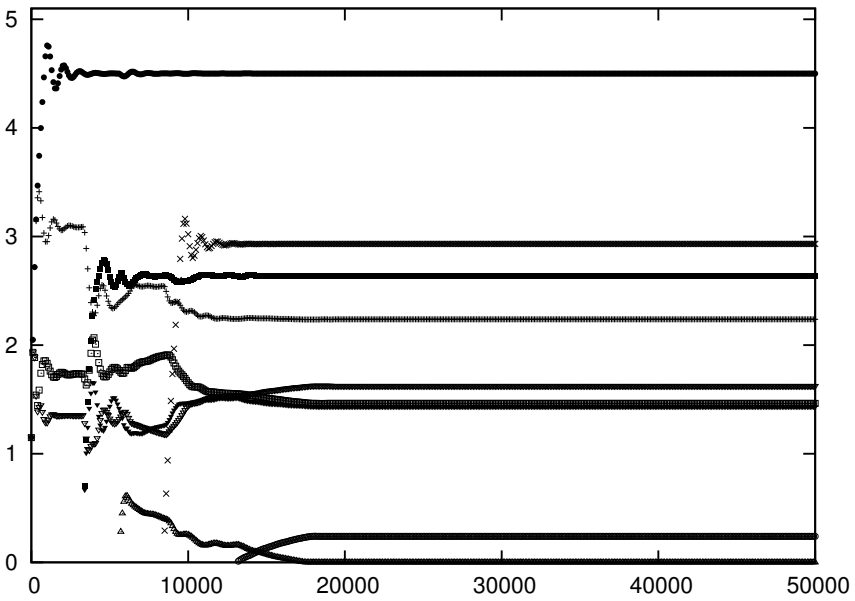


(b) Final solution obtained for $K = 3$; saturated nodes are labeled using inner circles; the width of a path is proportional to its flow rate and the path is colored according to its pair.

Figure 6.3: Problem instance with 22 nodes and 4 source-destination pairs



(a) $K = 2$



(b) $K = 3$

Figure 6.4: Evolution of path rates for each source-destination pair for the network in Figure 6.3.

and y_n converged to an optimal solution. The lower part of Figure 6.3 shows the solution for $K = 3$. Interestingly, not all of the $K = 3$ potential paths for each pair actually carry positive flow. One source uses one, two sources use two, and one source uses three paths to route flow to its destination.

If the network instance is small enough, we can solve a family of corresponding KNUM-E instances for increasing values for K, starting from 1, using the branch-and-bound algorithm. In this way, one can determine the smallest value $K = K_m$, such that an optimal solution to KNUM-E has the same objective value as an optimal solution to NUM and uses only K_m positive-flow paths for each pair. Similarly, we can solve a family of instances

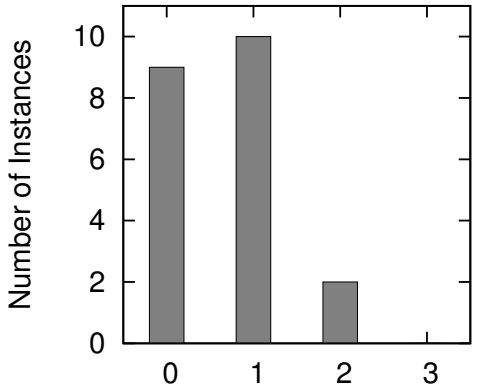


Figure 6.5: Histogram for the value $K_c - K_m$ for a set of 21 random graphs with 22 nodes.

of KNUM to find the smallest value $K = K_c$, which is sufficient so that our algorithm achieves convergence for the given network instance. The difference $K_c - K_m$ then equals the number by which K would need to be increased to achieve convergence to an optimal solution for NUM and thus, in some sense, corresponds to the overhead incurred by the distributed solution of that particular instance. Note that $K_c \geq K_m$ for any fixed instance because of Corollary 1.

Figure 6.5 shows a histogram for the results obtained for a set of 21 networks instances generated by the process described earlier. For all instances we observed $K_c \leq 5$. Figure 6.5 shows that for this set of instances the number of additional paths required is rather small and strictly less than 3 for all instances. Although one would need to perform more extensive experiments before concluding the feasibility of an implementation, it seems that the algorithm generally stabilizes using only a few more paths than the minimum number required for an optimal solution.

7 TRANSMISSION POWER ASSIGNMENT IN SENSOR NETWORKS

Previously, we addressed routing problems in ad hoc networks. We now turn to lifetime maximization in sensor networks. The methods proposed in this chapter compute energy-efficient communication graphs that induce transmission power assignments which achieve maximum lifetime. Our notion of lifetime assumes the constraint of maintaining network connectivity. We consider lifetime maximization by sleep scheduling in Chapter 8.

More precisely, we discuss two distributed algorithms that perform topology control by computing a connected communication graph with small maximum edge cost. At termination all nodes know which of their incident edges are in the graph and can then set their power levels to a value sufficient to transmit via these edges. We also propose a setup procedure for network initialization that lets nodes discover their neighbors and estimate transmission costs. We evaluate all algorithms by network simulations and compare their performance to a related algorithm proposed by Guo at al [52].

7.1 INTRODUCTION

Assume that a group of sensors has been deployed in an area to collect environmental data. Since networks are expected to self-organize, upon powering-up for the first time, nodes need to determine their neighborhood and decide where to forward collected data, at which intervals, transmission power levels, etc. An important objective during this self-configuration process is to initialize data-gathering and transmission protocols so that network lifetime is maximized [2, 25]. We make the following assumptions.

Battery: the initial battery capacities have the same value.

Data: data is aggregated on the way to the sink so that every node roughly transmits the same amount of data.

Deployment: the network is operating in a data-gathering scenario as depicted in Figure 2.1, so that there is a single sink node which needs to receive the data via multihop communication.

Energy: energy consumption due to transmitting dominates energy spent by receiving, data processing, etc.

Lifetime: all nodes are equally important and the notion of lifetime is the time until the first node dies (n-of-n lifetime).

Scheduling: the nodes operate a sleep-scheduling scheme so that idle time energy consumption is negligible.

Static: once the transmission power levels are set, they remain unchanged over the operating time of the network.

Transmission: transmission power is adjustable and transmission costs are time-invariant and symmetric.

The assumptions above describe a network consisting of sensor nodes that provide an approximately uniform, low-intensity stream of data to the sink. Consider for example a setting in which the task is to monitor some environmental parameters (e.g., humidity, temperature, concentration of chemical substances) in the deployment area. An application that falls into this category is a forest-fire detection system, where the traffic consists almost entirely of periodic "status OK" messages. For a different example consider a fence-monitoring system for intrusion detection. In the system proposed by Wittenburg et al. [149], nodes are able to save energy by choosing a lower sampling rate during normal operation and collaboratively detect intrusion events based on a relatively simple threshold model.

Besides assuming that transmission costs are symmetric and do not vary over time, we initially do not make any assumptions on them. This is contrary to the majority of related papers that usually assume that edge costs result from some specific path-loss model. We only assume that nodes know the cost of transmitting to a node within their transmission range, e.g., by monitoring the power level of their radio. In Section 7.4 we discuss an initial setup phase for determining edge costs that is based on reasonable assumptions on the propagation model. It is desirable to avoid extensive assumptions on the path loss that signals experience, since in practice it it hard to identify conditions that are satisfied by all possible network deployments. Our assumptions are satisfied, for example, in the case that costs represent signal attenuation that is derived from a deterministic path-loss model, which only depends on the pairwise distance of nodes.

Note that the assumption of time independent transmission costs may not always be justified. In wireless network deployments channel conditions are influenced by environmental factors, such as humidity or obstacles that may pass through the network [155]. However, in cases where channel conditions change on a slow timescale a static transmission-power setting may still suffice. In this case one may want to recompute the power assignment at regular intervals, while realizing that there may be a tradeoff between the power spent by running the algorithm versus the gain in lifetime obtained by the recomputation. For a brief overview of lifetime maximization see Section 2.3.2.

Problem formulation

Denote the set of nodes by V and assume we are given a transmission cost function $c : V \times V \mapsto [0, \infty]$ that captures the minimum power required to transmit from one node to another. Here, a node v can reach node u if the power level $\tau(v)$ satisfies $\tau(v) \geq c(v, u)$. Since we assume transmission costs to be symmetric, we have $c(u, v) = c(v, u)$. We include ∞ to model the fact that some nodes may not be within transmission range of each other. The communication graph $G = (V, E(\tau))$ is induced by the transmission power assignment τ, where $E(\tau)$ contains the edges that are supported by the power assignment, i.e., $E(\tau) = \{(u, v) \mid (u, v) \in V \times V,\ u \neq v,\ \tau(u) \geq c(u, v)\}$.

The *lifetime maximization problem* is formulated as follows: compute a transmission power assignment $\tau : V \mapsto [0, p^{\max}]$, where $p^{\max}$ is the maximum available power, such that under uniform load and uniform battery

capacities, the network stays (strongly) connected for a maximum amount of time until the first node runs out of energy. We assume that the graph $G(\tau^{\max})$ with $\tau^{\max}(v) = p^{\max}$ for all v is connected. Note that our notion of *network lifetime* is the n-of-n lifetime discussed in Section 2.3.2.

Although for the data-gathering scenario it would be sufficient if all nodes could reach the sink node via edges in $E(\tau)$, for the sake of a reliable MAC layer it is usually required to have bidirectional links. Hence, we restrict our search to power assignments that induce symmetric graphs. In order to simplify the exposition, we treat all graphs as being undirected.

Based on the assumptions above, the problem is essentially equivalent to the problem of finding a spanning subgraph of the graph $G(\tau^{\max})$ that has minimum maximum edge cost, i.e., a minmax spanner. After a minmax spanner $G^* = (V, E^*)$ has been computed, the nodes can locally choose their transmission powers by selecting the minimum level required to reach all nodes that are their neighbors in G^*. Note that although for any network there may exist several minmax spanners, the resulting power assignments all achieve the same lifetime.

The problem of finding a minmax spanner that is a tree is also referred to as the *bottleneck spanning tree* problem [49, 72]. It is easy to see that the maximum edge cost in a bottleneck spanning tree and in a minimum spanning tree of the same graph coincide [72]. Thus, the problem could essentially be solved by a distributed minimum spanning tree algorithm. However, the distributed computation of an MST, as proposed in [46], is much more involved than the distributed search for a minmax spanning tree. In this chapter, we present two simple and efficient distributed algorithms that are based on exploiting different properties of the lifetime maximization problem. We formulate the first algorithm in the unicast communication model, while the second algorithm uses only broadcast messages. Hence, the algorithms are expected to differ in the number of control messages and running time required. For differences between the two models see Section 4.1.

Related work

Existing work in the literature on lifetime maximization problems can be broadly distinguished by the objective to be maximized. We first look at work that aims at maximizing the shortest lifetime of any node and then briefly review relevant papers that minimize total energy consumption.

Minmax power optimization

Under our assumptions for the data-gathering scenario, the lifetime maximization problem is very similar to the problem of maximizing the lifetime of a single session broadcast. It was shown by Papadimitriou and Georgiadis [107] that the problem of finding the lexicographically smallest power assignment is computationally more complex than finding any minmax spanner. Another related problem is the *multicast lifetime maximization problem*, which was addressed by Floréen et al. [44]. It turns out that if the problem involves dynamic, i.e., time-varying, power assignments then one may not be able to solve it in polynomial time even in a centralized setting, i.e., the problem becomes NP-hard. Therefore, we only consider the static case.

Probably the earliest paper that addresses our version of the lifetime maximization problem is the work of Ramanathan and Rosales-Hain [117], who model it as a broadcasting problem. Ramanathan and Rosales-Hain propose a centralized algorithm for computing an optimal power assignment, as well as two simple distributed heuristics that may result in suboptimal solutions. Their first algorithm does not necessarily guarantee connectedness, while the second algorithm relies on an expensive link-state routing algorithm which is likely unsuitable for sensor networks.

Narayanaswamy et al. [99] consider discrete power levels, as we also do in Section 7.3. Their distributed algorithm is similar to ours in that it establishes a minmax spanner of $G(\tau^{\max})$. However, similar to [117], their algorithm relies on a proactive routing protocol and runs one instance for each power level. This algorithm may be applicable to ad hoc networks but the resulting control overhead renders it unsuitable for sensor networks.

Kang and Poovendran [72] study several aspects of dynamic lifetime maximization, i.e., the case where the power assignment may vary over time. They allow non-uniform initial battery power and further emphasize that the minmax energy metric is preferable over the more often addressed minimum total energy metric for the purpose of maximizing network lifetime. Kang and Poovendran refer to distributed methods for constructing minimum spanning trees for an implementation, such as the one proposed by Gallager, Humblet and Spira [46]. These techniques are, however, rather involved, and we complement this work by proposing two efficient and much simpler algorithms for minmax spanners in this chapter.

The problem of finding the *critical transmission range* in ad hoc networks has been addressed by Sanchez et al. [119]. This problem is equivalent to finding a minmax spanner of $G(\tau^{\max})$. The authors of [119] refer to the classical MST algorithms of Prim and Kruskal for computing the spanner.

The algorithm that is most similar to our proposed methods is the algorithm by Guo et al. [52], which is formulated in terms of multicast trees. Their algorithm roughly follows Prim's algorithm for constructing MST and can easily be adapted to the lifetime maximization problem. We therefore decided to use it as a baseline for comparisons with our algorithms.

Total power minimization

The lifetime maximization problem considered here is also related to the problem of minimizing the total network transmission power required for connectivity, which has been studied extensively over the past few years (e.g., see [87] and the references therein). This problem is related to the range assignment problem described in Section 2.3.1. Since it is not expected that there exist even efficient centralized algorithms that provide optimal solutions to all instances, algorithms proposed in the literature typically make special additional assumptions in order to guarantee optimality.

Rodoplu and Meng [118] propose a distributed algorithm to compute a topology that contains all minimum total energy paths from each node to a designated master node. Their approach relies on the geometric concept of *relay regions*: each node knows its own geographic location and the location of neighboring nodes. Based on a specific path-loss model each node can locally decide which neighbor it should forward a message to so that total

energy consumption is minimized.

Wattenhofer et al. [147] study a geometric cone-based forwarding scheme for the same problem. Their distributed algorithm requires that each node can measure the precise angle of arrival of radio transmissions it receives, which raises the demands on the available radio hardware.

Wieselthier et al. [148] propose a simple centralized heuristic that also exploits properties of the underlying path-loss model. They provide experimental evidence that savings in total energy can be achieved using their *broadcast incremental power* heuristic compared to other topologies, such as MST, although there is no optimality guarantee for the resulting solutions.

7.2 MAXIMUM LIFETIME SPANNER ALGORITHM

We now discuss our first distributed algorithm for computing a minmax spanner. Here, we assume that nodes know their neighbors in the graph $G(\tau^{\max})$, i.e., the communication graph that results from letting all nodes transmit at maximum power. We further assume that nodes know the cost for transmitting to their neighbors. Both requirements may be satisfied by performing an initial setup stage, which is discussed further in Section 7.4. In the following, we compute a minmax spanner for an arbitrary undirected and connected graph G, although for the implementation we choose $G = G(\tau^{\max})$.

7.2.1 Minmax spanner computation

Recall that a minmax spanner G^* of a connected graph $G = (V, E)$ is a subgraph of G with the same set of nodes V that is connected and has some maximum edge cost α^*. In addition, there are no α'-spanners with $\alpha' < \alpha^*$. Our maximum lifetime spanner (MLS) algorithm is based on the following simple construction of G^*. Consider all paths from a fixed node s to all remaining nodes. For each such node $v \in V \setminus \{s\}$, choose a path in G from s to v, which achieves minimum maximum edge cost, i.e., minimum bottleneck cost. Form the union of all these paths and denote the resulting edge set by E^*.

Lemma 3. *The resulting graph $G^* = (V, E^*)$ is a minmax spanner of G.*

Proof. The graph G^* is clearly a spanning subgraph of G and it is connected. We need to show that that there are no α'-spanners with $\alpha' < \alpha^*$. Suppose that there would be one, say G' with maximum edge cost α'. Choose an edge e in E^* that achieves the cost α^* and let p be the path that resulted in adding e to the set E^* in the construction above. The path p was chosen because it achieved minimum bottleneck cost among all the paths in G between its endpoints. Since G' is an α'-spanner and $\alpha' < \alpha^*$, there must exist a path in G' between the same endpoints with maximum edge cost strictly smaller than α^*. But G' is also a subgraph of G, so that is impossible. □

The same type of construction is used by Georgiadis [49] to compute bottleneck multicast trees by a centralized algorithm in linear time. Interestingly, the algorithm in [49] is derived by modifying Dijkstra's algorithm

for the search of minimum bottleneck paths, while our first algorithm results from a modification of the Bellmann-Ford algorithm. In this sense our approach is also similar to the algorithm by Gupta and Srimani [53], who propose a distributed self-stabilizing algorithm that computes shortest-paths multicast trees in ad hoc networks. However, since we do not require the construction of an MST, we are able to significantly simplify the algorithm.

Besides resulting in a distributed algorithm, the approach of computing minmax-cost paths allows for a simple and efficient reduction of the set of candidate edges prior to the execution of the algorithm. By computing a relative neighborhood graph and using this RNG instead of the original graph as input, we are able to drastically reduce our bound on the message complexity. In Section 7.4.3 we present a distributed method for obtaining the RNG of a given communication graph.

Our distributed MLS algorithm computes minimum bottleneck paths from the sink s to all other nodes. For this purpose, we modify algorithm BF to compute paths that achieve minimum bottleneck cost, rather than shortest paths. The pseudocode is shown on page 97. Since Algorithm BF was covered in Section 4.2.2, we restrict ourselves to the discussion of the differences between the two algorithms and refer to Section 4.2.2 and the pseudocode for details.

We assign the role of the initiator in Algorithm MLS to the sink node. If one compares Algorithm MLS to the Bellman-Ford algorithm on page 48, one notices that the only difference lies in lines 26 to 30 of Algorithm MLS. This difference, however, is crucial, since it may be used to show a lower message complexity for MLS. The condition in line 26 can be seen as a test to prune the search at a given node, if a new potential candidate for a minimum bottleneck cost cannot improve over a previously tested one at the subtree rooted at node v. This pruning is possible due to the properties of the minmax cost objective function. A sample run of Algorithm MLS on a small instance is given in Figure 7.1.

Theorem 11. *Algorithm* MLS *computes a minmax spanner of* G.

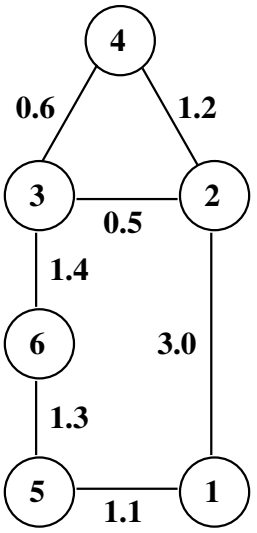
Proof. We only need to show that the algorithm terminates and that the α values correspond to the minimum maximum distance to the sink at termination. Since the father pointers are set to the next node on a path to the sink with bottleneck cost α, the result then follows.

Let us first show that the algorithm terminates. Denote the set of distinct edge costs by $\mathcal{C} = \{\, c(u,v) \mid (u,v) \in V \times V, u \neq v\}$. Obviously, no node can learn of a route with better bottleneck cost more than $|\mathcal{C}|$ times. Also, it is not possible that a node remains in the state SEARCH indefinitely. This follows since a node replies with an acknowledgment, positive or negative, latest when it has received acknowledgments from all its children in the spanning tree constructed. Hence, the algorithm terminates.

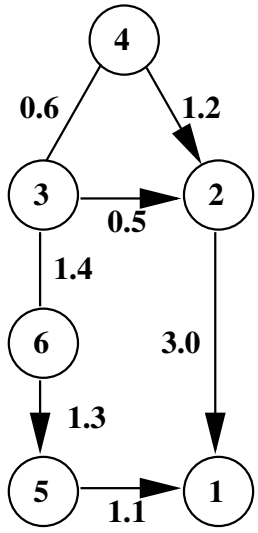
Assume that at termination, node v would have an α value larger than the minimum bottleneck path length from the sink to v. Then there would exist a path with bottleneck cost $\alpha' < \alpha$ from the sink to v via a node u, whose incident edges include an edge of cost α'. The node on the other side of the edge would have a larger estimate. This is not possible, since node u would send a request along that edge. □

Algorithm MLS: Algorithm MLS for finding a minmax spanner

```
1  node v with local variables α, α[·], father, status[·]
2  at start
3      father ← undefined;
4      if is_sink then
5          α ← 0;
6          for u in N(v) do
7              send (c(v, u)) to u;
8              α[u] ← c(v, u);
9              status [u] ← wait;
10         end
11         enter state SEARCH;
12     else
13         α ← ∞;
14         for u ∈ N(v) do
15             α[u] ← ∞;
16             status [u] ← ready;
17         end
18         enter state IDLE;
19     end
20
21 in state IDLE or SEARCH
22     if (α′) with α′ < α is received from some node u then
23         if father is defined then send NAK(α) to father;
24         father ← u; α ← α′;
25         for w in N(v) \ {u} do
26             if max(α′, c(v, w)) < α[w] then
27                 send (max(α′, c(v, w))) to w;
28                 α[w] ← max(α′, c(v, w));
29                 status [w] ← wait;
30             end
31         end
32         enter state SEARCH
33     end
34     if (α′) with α′ ≥ α is received from some node u then
35         send NAK(α′) to u;
36     end
37
38 in state SEARCH                    // wait for incoming acknowledgments
39     if status [w]=ready for all w ∈ N(v) \ {father} then
40         send ACK(α) to father;
41         enter state IDLE
42     end
43     if ACK(α′) or NAK(α′) is received from u and α[u] = α′ then
44         status [u] ← ready;
45     end
46
```



$1 \rightarrow 2$:(3.0),
$1 \rightarrow 5$:(1.1),
$2 \rightarrow 3$:(3.0),
$2 \rightarrow 4$:(3.0),
$5 \rightarrow 6$:(1.3),
$3 \rightarrow 4$:(3.0),
$4 \rightarrow 3$:NAK,
$4 \rightarrow 3$:(3.0),
$3 \rightarrow 4$:NAK,
$3 \rightarrow 6$:(3.0),
$6 \rightarrow 3$:NAK,
$3 \rightarrow 2$:ACK,
$4 \rightarrow 2$:ACK,
$2 \rightarrow 1$:ACK,
$6 \rightarrow 3$:(1.4),



$3 \rightarrow 2$:NAK,
$3 \rightarrow 2$:(1.4),
$2 \rightarrow 1$:NAK,
$3 \rightarrow 4$:(1.4),
$4 \rightarrow 2$:NAK,
$2 \rightarrow 4$:(1.4),
$4 \rightarrow 2$:NAK,
$4 \rightarrow 2$:(1.4),
$2 \rightarrow 4$:NAK,
$2 \rightarrow 1$:(1.4),
$1 \rightarrow 2$:NAK,
$2 \rightarrow 3$:ACK,
$4 \rightarrow 3$:ACK,
$3 \rightarrow 6$:ACK,
$6 \rightarrow 5$:ACK,
$5 \rightarrow 1$:ACK

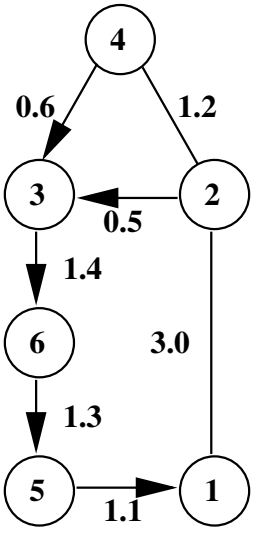


Figure 7.1: Sample execution of Algorithm MLS from reference node 1; messages listed as *source→destination:message*. The three graphs show the initial, intermediate and final state of the algorithm with messages listed on the right side of states in the order of their transmission.

Let us consider the message complexity of MLS. Here, as before, we disregard message retransmission due to transient link errors.

Theorem 12. *Algorithm* MLS *requires at most* $\mathcal{O}(|\mathcal{C}|\,|E|)$ *messages.*

Proof. We first observe that the number of distinct edge costs is bounded by $|\mathcal{C}| \leq |E|$.[1] When a node v receives a request with a lower α value, it sends a message to each neighbor u. This can occur at most $|\mathcal{C}|$ times. Each neighbor u will eventually reply with an ACK, a NAK or first an ACK and later a NAK when u learns about a better path later. Note that only the reception of lower α value can cause v to send any messages. Hence, the claim follows. □

In this sense, the problem of finding a minmax spanner is fundamentally different from the problem of finding shortest paths, where the number of paths with different total cost between a pair of nodes can be exponential in the number of nodes [90, Sec. 15.4].

Notification stage

The only node that is able to detect termination is the sink. In order to update their transmission power, however, all nodes need to be informed of termination. Hence, we require that after detecting termination, the sink initiates a network-wide broadcast using the edges of the computed spanning tree. When a node v receives this broadcast message, v decreases its power level $\tau(v)$ to the minimum that is required to reach its father and the neighbors that have chosen v to be their father.

7.2.2 Improvement for Euclidean space

In Algorithm MLS nodes exchange unicast messages with all neighboring nodes when a request with a better α was received. For dense sensor networks, where the number of neighbors of a node is relatively large, it may be beneficial to reduce the number of neighbors that have to be contacted. One needs to ensure, however, that the resulting graph is still a minmax spanner of the original graph.

Instead of modifying the algorithm, we change the communication graph that Algorithm MLS runs on by replacing G by its relative neighborhood graph. Recall that a relative neighborhood graph is obtained from the original graph by removing the longest edge of each triangle of edges. We are not the first to employ an RNG for topology control, see [15, 18] also for other applications. Usually, however, RNGs have been used in a geometric context.

Our model only requires symmetric path loss, which means that $c(u, v) = c(v, u)$ for all nodes u and v, although our algorithm provides better worst-case guarantees when nodes are located in the Euclidean plane. More precisely, when the nodes are placed in the plane and path loss is an increasing function of distance, the RNG is a subgraph of the Delaunay triangulation of G and contains $\mathcal{O}(|V|)$ edges [133]. It follows that when run on the RNG,

[1] We can disregard edge costs $c(u, v) = \infty$ since it is not possible to transmit messages via non-existing edges.

algorithm MLS has a message complexity of $\mathcal{O}(|\mathcal{C}||V|)$. In Section 7.4.3 we describe a distributed algorithm to obtain the RNG by a beaconing method run initially to setup the network. The method for constructing the RNG is efficient, since it has a message complexity of $\mathcal{O}(|V|)$ broadcast messages and a communication complexity of $\mathcal{O}(|E|)$ bits in total. To conclude, we obtain the following theorem.

Theorem 13. *When run on the* RNG *of the communication graph, which is a geometric graph whose nodes are placed in the plane, and if edge costs are non-decreasing in distance, then Algorithm* MLS *requires* $\mathcal{O}(|\mathcal{C}|\,|V|)$ *messages in the worst case.*

Note that depending on node density, the difference in the bounds of Theorems 12 and 13 can be significant.

7.3 BINARY SEARCH FOR MINMAX POWER SPANNERS

In the previous section we described an algorithm that is based on the search for minimum bottleneck paths from the sink to all other nodes in the graph. For computing the power assignment it would be sufficient only to know the actual bottleneck cost of such a minmax spanner. Since in practice the number of available different power levels can be rather small, for large networks probably much smaller than $|E|$, one could instead search over these and find the smallest one that is sufficient to obtain network connectivity. This is the basic idea of the algorithm presented in this section, which implements a binary search over the available power levels. Note that the same idea of performing a binary search over power levels was proposed by Lloyd et al. [87]. They, however, perform the search under the assumption that complete information on network connectivity and edge costs is available at a central location.

Besides modifying the search, we also want to reduce the number of transmissions by using broadcast instead of unicast messages. Particularly in dense networks we expect the savings to be significant. Changing to the broadcast communication model, however, comes at a cost, since transmissions become unreliable, as discussed in Section 2.2.1. We propose measures to mitigate this problem below. It turns out that the problem in this case has a simple solution based on retransmission timers, once the nodes have determined their neighborhood.

We make two assumptions in addition to the ones formulated above. Let the available transmission power levels be given by a set $P = \{p_1, p_2, \ldots, p_{|P|}\}$, where $p_1 < p_2 < \ldots < p_{|P|} = p^{\max}$, which is the same for all nodes. A solution to the lifetime maximization problem is then a power-level assignment of the type $\tau : V \mapsto P$. We further assume that the sink node is aware of the total number of nodes in the network. Knowledge of the network size as well as the local node neighborhoods will be gathered by performing the setup procedure we propose in the next section. As the MLS algorithm before, the Binary Search for Min-Max Power Spanner (BSPAN) algorithm relies on the presence of a single initiator, which in our case is the sink node.

Algorithm BSPAN performs a distributed binary search that is coordinated

Algorithm BSPAN: Algorithm BSPAN for finding a minmax spanner

```
1  node v with variables α, lower, curr, upper, child_candidates, father,
   is_sink, N(v), node_count, status;
2  at start
3      if is_sink then
4          lower ← 0; upper ← |P|;
5          curr ← ⌊(lower + upper)/2⌋; α ← p_curr;
6      for u ∈ N(v) do status[u] ← processed;
7      enter state RESET;
8
9  in state RESET
10     father ← none; node_count ← 0;
11     if is_sink then
12         if lower + 1 < upper then enter state SEND_REQUEST;
13         else enter state SEARCH_FINISHED;
14     else enter state IDLE;
15
16 in state IDLE                                  // wait for incoming requests
17     if request(u, α′, f′) with α′ ≥ c(u, v) is received then
18         α ← α′; father ← u;
19         enter state SEND_REQUEST;
20     end
21
22 in state SEND_REQUEST                  // broadcast request to neighbors
23     child_candidates ← {w ∈ N(v) \ {father} | c(v, w) ≤ α};
24     for w ∈ child_candidates do status[w] ← wait;
25     if child_candidates ≠ ∅ then                          // (see text on p. 102)
26         broadcast request(v, α, father);
27     enter state PROCESSING;
28
29 in state PROCESSING                  // process requests, wait for replies
30     if request(u, α′, f′) is received then
31         if f′ = v then status[u] ← child;  // u acknowledged v as father
           else status[u] ← processed;    // u has father f′ different from v
32     end
33     if reply(u, nodes) is received then
34         status[u] ← processed; node_count ← node_count + nodes;
35     end
36     if status[w] = processed for all w ∈ N(v) \ {father} then
37         if is_sink then
38             if total_nodes = node_count then upper ← curr;
39             else lower ← curr;
40             curr ← ⌊(lower + upper)/2⌋; α ← p_curr;
41         else unicast reply(v, node_count + 1) to father;  // report count
           enter state RESET;
42     end
43
```

by the sink node. Each iteration of the algorithm corresponds to a trial that counts the number of nodes reachable from the sink via edges of cost at most a threshold cost. Since the sink knows the total number of nodes in the network, it can establish whether a particular candidate cost α suffices to connect the network. We give details on the operation below.

The algorithm operates in iterations, which consist of two phases. The first phase of an iteration is similar to Algorithm SHOUT for constructing spanning trees. Here, however, we need to ensure that the cost of an edge which was used to receive a request does not exceed the current candidate cost α (see line 17). This is required since we run the algorithm on the graph $G = G(\tau^{\max})$. Hence, the spanning tree formed by the algorithm will contain only edges of cost at most α.

In the second phase of each iteration, a convergecast operation counts the number of nodes that are contained in the tree previously constructed. Once the sink has received reply messages from all its children in the tree, it can compare the number of nodes reached to the total number of nodes and update its bounds on the minmax edge cost.

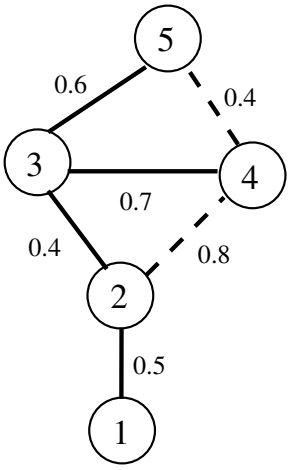
Request phase

A request message is of form (u, α, f), where u is the node identifier of the sender, α is the bottleneck edge cost and f is the father node of u in the current iteration. Suppose that node v receives a request from u via the edge (u, v). If this is the first request that v received in the current iteration and if $c(u, v) \leq \alpha$, then u becomes the *father* of v. Node v then determines whether it needs to send a request to its neighbors. It does so only if it just received the first request in the current iteration, if $c(u, v) \leq \alpha$, and if additionally there exist adjacent nodes w different from u such that the edge (v, w) has cost $c(v, w) \leq \alpha$. Note that the last condition corresponds to the check in line 25 of Algorithm BSPAN shown on page 101.

Reply phase

After a node v has sent a request (v, α, u), v waits until it has received requests from every neighbor in G that meets the conditions given above, i.e., from every node that is a potential child of v in the tree constructed in the current iteration. If v receives a request (w, α, v') from node w, node v labels w as *child* if $v' = v$, and as *processed* otherwise. A neighboring node w that was labeled as *child* corresponds to w being v's child in the tree constructed. A label *processed* implies that v does not expect a reply from w, which at this stage corresponds to w being a member of the tree with a different father.

It is possible that a node v has no child nodes. This can be the case either because v does not have adjacent nodes with a sufficiently low edge cost or if all of these have different father nodes. In both cases v can determine that it is a leaf in the current tree. Subsequently, node v sends a reply message $(v, 1)$ to its father, which includes a node count of 1. If v has at least one child w, v waits for reply messages from all children before it sends a reply to its own father. Each node keeps track of which children have sent a reply by marking them as *processed*.



(a) Spanning Tree

1: transmit request$(1, 0.75, -)$,
2: mark 1 as father,
2: transmit request$(2, 0.75, 1)$,
4: drop request from 2,
1: mark 2 as child,
3: mark 2 as father,
3: transmit request$(3, 0.75, 2)$,
2: mark 3 as child,
4, 5: mark 3 as father,
5: transmit request$(5, 0.75, 3)$,
4: transmit request$(4, 0.75, 3)$,
4: drop request from 5,
5: drop request from 4,

(b) Request phase

$4 \rightarrow 3$: reply$(4, 1)$,
$5 \rightarrow 3$: reply$(5, 1)$,
$3 \rightarrow 2$: reply$(3, 3)$,
$2 \rightarrow 1$: reply$(2, 4)$

(c) Reply phase

Figure 7.2: Single iteration of Algorithm BSPAN with $\alpha = 0.75$, initiated by sink with identifier 1 (a) spanning tree defined by father records at the end of the iteration; edges outside the tree are shown dashed (b) messages and actions of nodes during request phase (c) replies sent along spanning tree edges during reply phase; note that requests are sent via broadcast, while replies are sent from a child to its father by unicast; set of power levels $P = \{0.05, 0.1, \ldots, 0.95, 1.0\}$.

After v received the last outstanding reply, i.e., when all neighbors except the father are labeled *processed*, v sums up the node counts of all replies received from its children, increments the sum by one and sends a reply message containing the count to its father. Thus, at the end of this convergecast operation the sink can determine the number of nodes in the network reachable by edges with cost at most the current candidate α. Since the sink also knows the total number of nodes in the network, it is able to determine whether α is an upper or lower bound of the minimum bottleneck cost and update α correspondingly. Figure 7.2 shows a single iteration of the algorithm for a small instance.

Notification stage

Here, as for Algorithm MLS, only the sink can detect termination. Hence, after the reply phase of the last iteration has terminated, the sink informs the remaining nodes about the termination and the final value for α, which then equals the minmax edge cost required for connectivity. The notification stage consists of broadcasting messages over edges of cost at most α and thus constructs a minmax spanner. After local termination of the notification stage, all nodes set their transmission-power levels to the minimum level required to reach the father and all child nodes in the tree.

Message complexity

The number of messages sent by the BSPAN algorithm can be easily bounded.

Theorem 14. *Algorithm* BSPAN *requires* $\mathcal{O}(\log(|P|)\ |V|)$ *messages in the*

worst case.

Proof. First, one observes that the binary search over the set of power levels P requires exactly $\lceil \log(|P|) \rceil$ iterations, where the logarithm is taken in base 2. In each iteration of the algorithm every node that has received a request sends at most two messages, one request and one reply message. Additionally, the sink sends a request but no reply message. Hence, the number of messages transmitted in one iteration is upper bounded by $2|V| - 1$. Due to the explicit expression on the number of iterations we obtain that the algorithm has message complexity $\mathcal{O}(\log(|P|)\ |V|)$. □

As a further note, consider a modification of algorithm BSPAN where instead of searching over the value range of power levels one searches over an ordered set $\mathcal{C}$ of real numbers. This simple change may be desirable for some situations where power levels are not necessarily equidistant and the number of available levels is small. Instead of including the value α, messages would then encode the rank of the power level in each iteration. Note that this modification relies on the assumption that all nodes share the same set of power levels.

Implementation remarks

Recall that the reception of broadcast messages is not acknowledged by the MAC layer, which complicates the detection of collisions. The following two situations with a collision of a request message cause problems for BSPAN. Assume that v broadcasts a request where node u is one of the intended recipients, i.e., $u \in$ child_candidates in line 23 of Algorithm BSPAN. Now if either the request sent by v collides with another message at u, or the later request sent by u collides with another message at v, then v will stay forever in state PROCESSING. This follows since v waits for u to send a request, potentially indicating v to be the father of u, or unicast a reply message if u is in fact a leaf in the tree and its father is v.

However, both situations can be easily handled by retransmission timers with a small timeout value. This follows, since *nodes know whom to wait for.* The timer ensures that v retransmits a copy of the last request until v receives any message from u. Retransmitted requests are sent using unicast communication, which is sufficient since not necessarily all neighbors of v are required to react. Whenever a node u receives a retransmitted request, it will either unicast its last transmitted request back to v, which is a signal for v that v is not u's father in the tree, or process it as usual if it has not processed any request in the current iteration.

7.4 ALGORITHM INITIALIZATION AND TERMINATION

Both our proposed algorithms for computing minmax spanners, MLS and BSPAN, rely on some restricted knowledge of the network topology, such as network size and neighbor lists. Further, they require the notification of nodes after termination was detected by the sink, so that nodes can set their power level accordingly. We now discuss a simple setup method for

collecting the required neighborhood information and network-wide termination notification. These methods employ standard distributed-algorithm techniques upon which we expand by integrating the computation of the edge-cost function c.

7.4.1 Setup stage

Algorithm BEACSET is a simple modification of Algorithm SHOUT (see p. 46) that uses broadcast instead of unicast messages. It computes a spanning tree of the communication graph by letting nodes repeatedly transmit beacon messages at their maximum available power level p^{max}, where random delays are injected into the beacon sequence to desynchronize neighboring transmissions. A node starts to transmit beacons once it has entered the tree, where initially the tree only contains the sink. The reception of a beacon message lets nodes estimate the cost of their incident edges in the communication graph, discover their neighbors and determine whether they are leaf nodes in the spanning tree. By transmitting beacons not once but several times the probability of undiscovered edges due to packet collisions is reduced significantly.

Algorithm BEACSET is also very similar to a single iteration of Algorithm BSPAN, since it counts the number of nodes reached via a convergecast of reply messages towards the sink node. This convergecast also allows the sink to determine termination of the setup stage and then to either initiate MLS or BSPAN to compute a minmax spanner of the communication graph that was constructed.

A node v that receives a beacon message processes the beacon by estimating the transmission power required to reach the node that originated the beacon. More formally, we consider a message that v receives from u with signal strength s^{recv} arrived successfully if $s^{\text{recv}} \geq s^{\text{thresh}}$, where s^{thresh} is the threshold signal strength necessary for a successful transmission. Here we disregard any effects caused by mutual interference. For the simulations we performed to evaluate our algorithms, the signal strength s^{recv} for each reception is determined by the propagation model of the simulator. The value for s^{thresh} is a constant, which in practice may depend on the sensitivity of the receiver. Here we use the default value of the NS2 simulator.

We assume that the received signal strength depends linearly on the sending power, i.e., $s^{\text{recv}} = X_{u,v}\, p^{\text{send}}$, where $X_{u,v}$ is the attenuation coefficient [39, 64]. Note that $X_{v,u} = X_{u,v}$ since we assume path loss is symmetric. It follows that if the receiver v knows the power that u used for sending, then v can compute the minimum transmission power p^{min} it has to use to transmit to u. Note that a transmission power p^{min} by definition corresponds to a received signal strength of s^{thresh}. Hence, all v needs to do is to solve $s^{\text{thresh}} = X_{v,u}\, p^{\text{min}}$, which translates into

$$p^{\text{min}} = \frac{s^{\text{thresh}}\, p^{\text{send}}}{s^{\text{recv}}} = \frac{s^{\text{thresh}}}{s^{\text{recv}}} p^{\text{max}},$$

where p^{send} is the power that was used by u for transmitting and the right-hand equality holds since in Algorithm BEACSET we use $p^{\text{send}} = p^{\text{max}}$ for the beacons. If our assumptions on the path loss do not hold, for example

Algorithm BEACSET: Beaconing algorithm for network setup

```
1  node v with variables beacon_count, beacon_delay, expecting_reply,
   father, neighbor_list, node_count, timer;
2  at start
3      node_count ← 0; expecting_reply ← ∅;
4      enter state IDLE;
5
6  in state IDLE                                  // wait for incoming beacons
7      if beacon(u, f′) is received with strength s^recv then
8          father ← u; c(v, u) ← (s^thresh / s^recv) p^max;        // estimate cost
           neighbor_list ← neighbor_list ∪ (u, c(v, u));
9          broadcast beacon(v, father) at power p^max;
10         timer ← beacon event after rand(0,beacon_delay);
11         enter state ACTIVE;
12     end
13
14 in state ACTIVE                                // send beacons with random delay
15     if beacon(u, f′) is received with strength s^recv then
16         c(v, u) ← (s^thresh / s^recv) p^max;                    // estimate cost
           neighbor_list ← neighbor_list ∪ (u, c(v, u));
17         if f′ = v then
18             expecting_reply ← expecting_reply ∪ {u};
19
20     end
21     if reply(u, count) is received with strength s^recv then
22         expecting_reply ← expecting_reply \ {u};
23         if expecting_reply = ∅ and beacon_count = beacon_repetitions
           then
24             unicast reply(v, node_count + count + 1) at power p^max to
25               father;
26             enter state SETUP_FINISHED;                       // end of stage
27         else
28             node_count ← node_count + count;
29         end
30     end
31     if timer triggers beacon event then
32         broadcast beacon(v, father) at power p^max;
33         if beacon_count < beacon_repetitions then
34             beacon_count ← beacon_count + 1;
35             timer ← beacon event after rand(0,beacon_delay);
36         else if expecting_reply = ∅ then
37             unicast reply(v, node_count + 1) at power p^max to father;
38             enter state SETUP_FINISHED;                       // end of stage
39         end
40     end
41
```

because the measured received signal strength shows random variations, then beacon messages sent with varying transmission power could be used for the same purpose. Similar techniques were proposed by Kohvakka et al. [77].

For the lifetime maximization problem of this chapter we choose the costs $c(v,u)$ of edges (v,u) in the communication graph to be equal to the minimum power required for node v to send to u. Hence, by computing $p^{\min}$ node v can estimate the cost for the edge to u as $c(v,u) = p^{\min} = (s^{\text{thresh}}/s^{\text{recv}})p^{\max}$. For details on the path-loss model and how it relates to other models we refer to the work by Iyer et al. [64].

The most frequently used method for measuring the strength of arriving radio signals is probably Received Signal Strength Indication (RSSI) [131]. However, one could also employ alternative methods such as the ones proposed in [77]. In order to obtain correct local neighborhood information at all nodes, for a reasonable node density we expect the number of beacon retransmissions to be fairly small. Due to our assumptions on the communication model in Section 4.1, it follows that the message complexity of Algorithm BEACSET is $\mathcal{O}(|V|)$.

Implementation remarks

In line 37 of Algorithm BEACSET a node v sends a reply to its father as soon as it has finished its beacon sequence if all neighbors that have v as their father have sent a reply to v. However, in the presence of message collisions, it is possible that a node u has chosen v as father but u has either not started its beacon sequence yet, or all beacons sent by u so far have collided at v. For this reason we implemented an additional delay: when a node finishes its beacon sequence, even if it is not aware of any nodes it has to wait for in its neighborhood, it waits for some time before it sends a reply. This delay gives u the chance to receive and either retransmit or reply to any of the beacons from v before v sends its reply to its father. The delay can be rather small in practice, since it only depends on the length of the beacon sequence and the *propagation delay*, which is the maximum time difference between sending and receiving a message transmitted between neighboring nodes.

7.4.2 Notification stage

For the nodes to be able to set their transmission powers to values that induce the minmax spanner computed by MLS or BSPAN, they need to be informed about termination. The notification stage performs a simple network-wide broadcast initiated by the sink and its message complexity is therefore $\mathcal{O}(|V|)$. The implementation depends on whether MLS or BSPAN was used.

Since all nodes can infer which of their incident edges are part of the minmax spanner computed, after the local termination of the notification stage each node sets its transmission power $\tau(v)$ to a value that suffices to maintain the most expensive edge to its neighbors in the minmax spanner, i.e., $\tau(v) = \max_{(v,u)\in E^*} c(v,u)$, where E^* are the edges contained in the minmax spanner. Note that in general this value may be lower than the bottleneck cost.

7.4.3 Distributed algorithm for RNG computation

Recall that we are able to obtain an improved bound on the message complexity for MLS when run on a relative neighborhood graph of the communication graph instead of the graph itself, assuming the nodes are located in the plane. Now we discuss a simple modification of Algorithm BEACSET to obtain a distributed method for constructing RNGs that does not require initial knowledge of the node neighborhoods or the cost function c.

In the modified beaconing method nodes not only beacon their own identifier, but also information about their one-hop neighborhood, so that each node can build up a view of its two-hop neighborhood in the communication graph. More precisely, a node v includes the list of identifiers and the corresponding edge costs c for those nodes in its one-hop neighborhood $N^+(v)$, which it has discovered so far. After having learned about their two-hop neighborhood $N_2(v)$, the nodes then prune unnecessary edges from the communication graph, based on the defining property of an RNG.

The edge pruning operates as follows. After the setup has locally terminated, each node v considers all triangles that can be formed by choosing v and two of its neighbors. It does so by iterating over all pairs of neighbors that also share an edge between them. Let $u, w \in N_1(v)$ be two such neighbors. Whenever v sees that $c(v,w) < c(v,u)$ and $c(w,u) < c(v,u)$, then v determines that the edge (v,u) is not part of the RNG, which follows from Definition 1. However, v initially only marks the edge (v,u) as redundant, which corresponds to marking u to be removed from its neighborhood. Only when v has considered all possible triangles and found all redundant neighbors it may permanently remove them from its neighborhood.

In order to let neighboring nodes make the same decision on whether or not to prune an edge between them, the nodes need to break ties between edge costs deterministically, e.g., based on node identifiers. We include the pruning of non-RNG edges into Algorithm BEACSET when node v sends the reply message to its father node.

Since the number of beacon repetitions is constant, the nodes can prune their neighborhoods so that only the RNG remains using $\mathcal{O}(|V|)$ broadcast messages. The size of each beacon message is proportional to the number of neighbors of the particular node. Thus, in our model of Section 4.1, we obtain a message complexity of $\mathcal{O}(|E|)$, assuming each value of the cost function c can be represented by a constant number of bits.

7.5 EXPERIMENTAL EVALUATION

We implemented the two algorithms for computing minmax spanners, MLS and BSPAN, together with the setup and RNG algorithm of Section 7.4 in NS2. From the algorithms proposed in the literature, we found the Distributed Min-Max Tree (DMMT) algorithm by Guo et al. [52] to be the approach closest to ours. Hence, we implemented DMMT as close to the description in [52] as we found feasible and used it as a benchmark algorithm. We set out to answer the following questions by the simulations.

- How do MLS and BSPAN perform compared to DMMT when con-

NS2 version	2.31	Node density	1 node per 130 m×130 m
Transmission range	250 m	Number of nodes	50-500
Max jitter (MLS, BSPAN)	0.5 s	Propagation model	Two-ray ground
Message timeout	2.1 s	$\|P\|$	128
Beacon delay (max)	1.5 s	Beacon repetitions	10

Table 7.1: Parameter values used in the simulations.

sidering observed message and time complexity?

- Does running MLS on the RNG improve performance? If yes, then to what extent?
- Is it possible to observe structural differences between the graphs computed by the different algorithms?

For our experiments we generated disk graphs by distributing nodes over a square area uniformly at random. Connectivity was determined by the NS2 default maximum transmission range. The propagation model was set to the two-ray ground model [21, 93], which meets the conditions given in Section 7.4.1. Since the problem formulation assumes the input graph to be connected, we discarded disconnected graphs.

Table 7.1 shows the values of simulation parameters. For initial trials we used 4 beacon transmissions per node in the setup stage and observed a failure rate of 1 to 2% of the instances in which some edges present in the communication graph were not detected by some nodes or for which the total node count obtained was incorrect. Note that the latter may be the case if a node incorrectly determines to be a leaf during the setup stage because of message collisions, which thus leads to a failure of the convergecast. Because RNG pruning requires correct information about neighbors to be broadcast, the modifications to Algorithm BEACSET for RNG computation make it more susceptible to message collisions.

Since the main focus of the experiments was on analyzing MLS and BSPAN performance, we chose to increase the number of beacons until the setup stage terminated successfully for all instances. Hence, we settled on a rather large value of 10 beacons per node. Alternatively, it would also be possible to vary the time interval between beaconing messages. See Table 7.1 for other parameter values.

Distributed minmax tree algorithm

The Distributed Min-Max Tree (DMMT) algorithm was proposed in [52] for computing multicast spanning trees, which connect a given subset $M \subseteq V$ of nodes with minmax edge cost. The same algorithm can be readily applied to solve the lifetime maximization problem by simply choosing $M = V$. Two versions of DMMT were proposed in [52], for directional and for omnidirectional antennas. Since NS2 nodes have omnidirectional antennas, we used the latter version of the algorithm. The algorithm borrows ideas from the well-known Prim's MST algorithm (see for example [34, pp. 570-573]).

Prim's algorithm obtains a minimum spanning tree by growing a subtree of the original graph starting from an initial node, such that in each step an edge is added that connects one node belonging to the tree and another node not yet in the tree. Among all potential edges, one that achieves minimum cost is chosen. After the tree spans all nodes, Prim's algorithm terminates and the resulting tree forms an MST.

A centralized version of DMMT is very similar to Prim's algorithm but adds an additional step to each iteration. The algorithm starts from a fixed initiator node, which is the source of the multicast session. After the minimum cost of any edge that connects a non-tree node to the tree is found, say α, DMMT grows the current tree by adding nodes that can be reached via edges of cost at most the current threshold α. This process is called *growth phase* and the process of finding the minimum cost of any outgoing edge is called *search phase*.

The search phase in the distributed version of DMMT is implemented via a convergecast of edge costs to the initiator node. This phase requires that all nodes keep state information of their neighbors, i.e., they need to know which neighbors are in the tree. After the convergecast terminates at the initiator, the cost value α is propagated to all tree nodes via a *join request* message. In the following growth phase each tree node v then forwards the request to each neighbor u that v believes is not yet in the tree if $c(v, u) \leq \alpha$. Whenever a node receives a join request for the first time, the sender of the request becomes the father of the recipient in the spanning tree to be constructed. All following join request that are received are used to update the tree status of neighboring nodes.

However, the distributed version of DMMT does not necessarily always find an outgoing edge during the search phase. This is due to the fact that a node v only learns about a neighbor u having joined the tree via a father $w \neq v$, if u later forwards a join request to v. This situation can result in costly non-progress iterations of the algorithm.

The DMMT algorithm as formulated in [52] employs timers at each node for estimating the local termination of the growth phase. In our comparison we implemented a more synchronized method of a single timer used at the sink node. After this timer expires, the sink notifies all nodes to switch from the growth to the search phase. This modification was deemed necessary for DMMT to become more resilient against packet drops at the MAC level. We did not take into account the additional control messages resulting from this modification in the comparisons described below.

Network simulations

In the simulations of MLS and BSPAN, the sink initiates Algorithm BEACSET to obtain neighbor lists, edge costs and a count of the nodes at the sink. The DMMT algorithm also requires neighborhood and edge cost information, which was loaded into the neighbor table of each node prior to execution. However, the same setup stage could have been used for this purpose. For BSPAN we chose P as a set of 128 equidistant power levels, whose maximum value was selected according to the propagation model of NS2.

Figure 7.3 shows one communication graph with 100 nodes, its MST,

RNG, and the minmax spanners constructed by DMMT, MLS, and BSPAN. One can observe that the shortest-hop distances between nodes in the tree constructed by MLS run on the RNG are generally slightly longer than in the tree resulting from MLS and the original graph. Although pruning a graph to its RNG generally reduces the number of edges, and hence the number of messages required by MLS, it also removes paths with a small hop count.

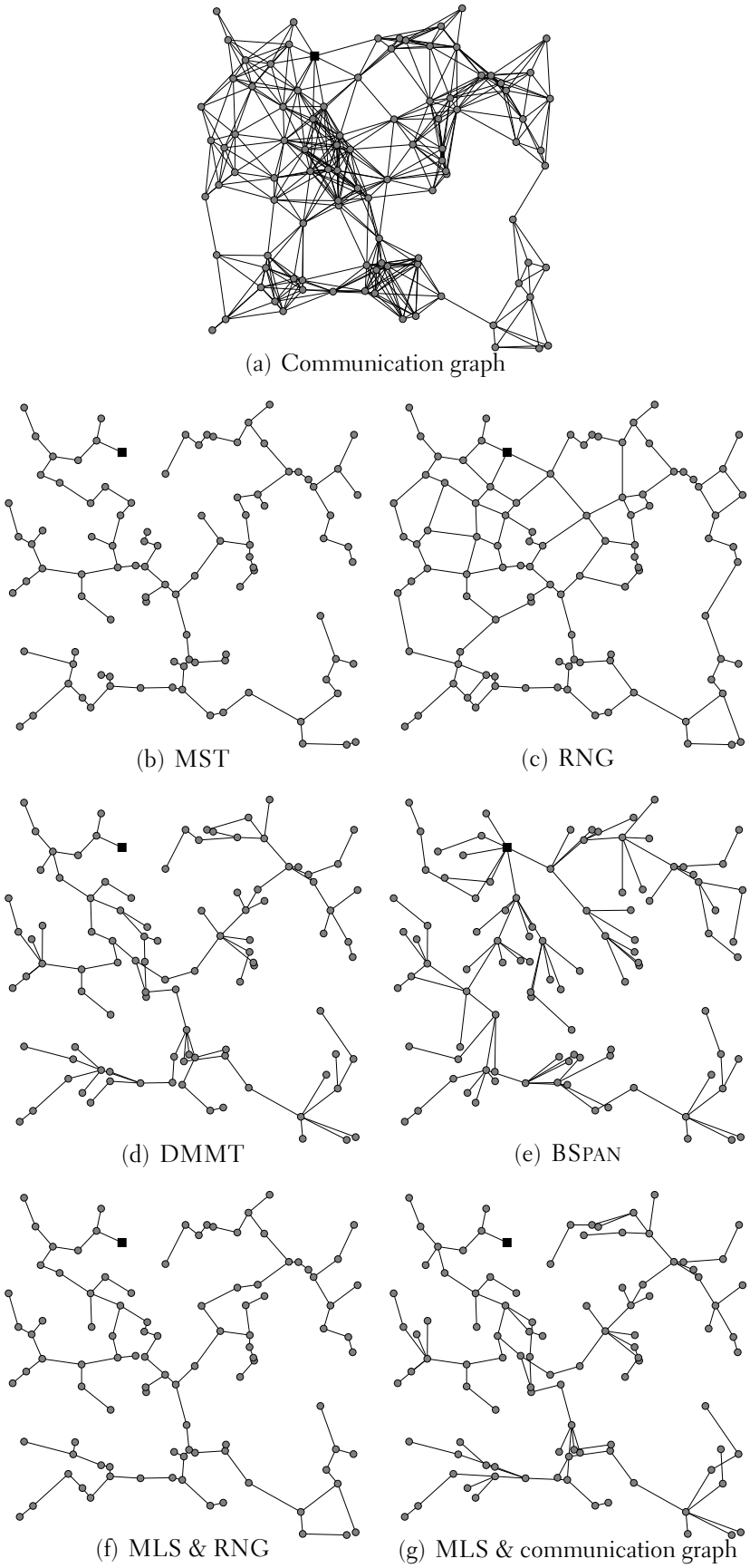
Figure 7.4 shows the total message count for the different algorithms averaged over a set of problem instances. Since the message complexities of DMMT and BSPAN appear insensitive to the choice of the input graph, we only show data for the two algorithms run on the communication graph. Note that DMMT considers only the single least-cost outgoing edge in each iteration, while BSPAN uses broadcast messages which are sent to all neighbors simultaneously. Recall that for MLS and BSPAN message counts include the messages transmitted by Algorithm BEACSET. Despite this advantage of DMMT, both algorithms outperformed DMMT significantly when comparing the number of control messages required.

More precisely, from Figure 7.4(a) one can see that in our simulations DMMT requires between 2 and 6 times more control messages compared to MLS run on the communication graph and between 6 and more than 30 times more messages than BSPAN. Hence, it appears that DMMT does not scale well with the number of nodes in the network. When comparing the performance of MLS and BSPAN, one observes from Figure 7.4(b) that BSPAN outperformed MLS by a factor of 2.5 to 4 when MLS was run on the communication graph. However, when we used the modified setup algorithm that constructs the RNG, MLS outperformed BSPAN by a factor of 1.5 to 1.2, depending on the size of the network. It is worth mentioning that BSPAN seems to significantly benefit from using broadcast messages as implicit acknowledgments.

When considering the message complexity of BSPAN as a function of the network size and number of power levels, one observes the following. For a fixed set of power levels P the number of messages required by BSPAN is linear in the network size $|V|$, whereas for a fixed number of nodes the message count is linear in $\log |P|$. This effect is illustrated by Figure 7.5. To conclude, the simulations indicate that in contrast to DMMT and MLS when run on the communication graph, MLS run on the RNG and BSPAN scale well with the size of the network.

We now turn to execution time. When evaluating execution time by network simulations, one needs to take into account the effect of timers on algorithm performance. If one assumes a collision free network that does not require retransmissions, both MLS and BSPAN would only require one timer in the setup stage as described in algorithm BEACSET. This is due to the fact that a node v determines to be a leaf in the spanning tree constructed, if no other node u has transmitted a beacon indicating v to be the father of u. Therefore, a timer is required to let nodes discover their children in the tree.

The DMMT algorithm uses timers extensively, which leads to a strong dependency of its execution time on timeout values. In our simulations, the execution times shown in Figure 7.6 indicate that BSPAN is slightly slower than MLS, and that both significantly outperform DMMT. One should



(a) Communication graph

(b) MST

(c) RNG

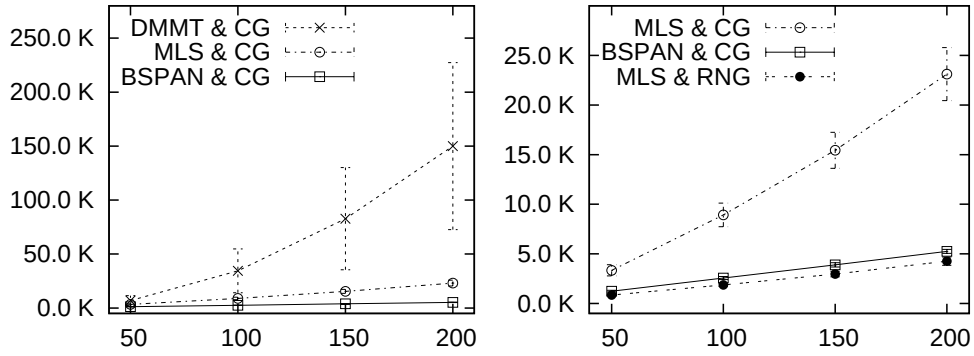
(d) DMMT

(e) BSPAN

(f) MLS & RNG

(g) MLS & communication graph

Figure 7.3: Sink is marked by a black square; bottleneck edge located in the bottom right part of picture.



(a) DMMT, MLS, and BSPAN run on communication graph

(b) MLS and BSPAN run on communication graph and MLS run on RNG

Figure 7.4: Message counts versus $|V|$, including setup but excluding notification stage. Error bars show standard deviations over 200 repetitions. For BSPAN $|P| = 128$. Data is shown for MLS run on the communication graph CG and its RNG. Note the different y-axis scale of (a) and (b).

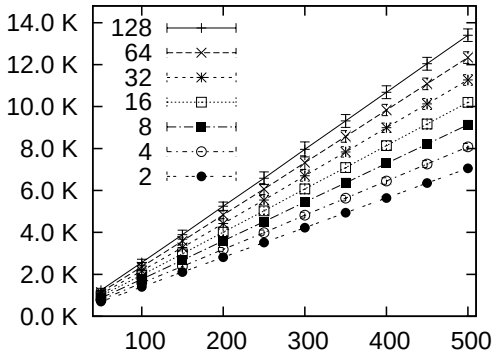


Figure 7.5: Message count for BSPAN versus number of nodes for various $|P|$, including setup stage but excluding notification stage; error bars show standard deviations for 200 repetitions.

note, however, that by choosing timeout values more carefully, one could probably achieve some improvements in the execution time of DMMT.

As we already argued above, when MLS is run on the RNG instead of the communication graph, the number of messages sent is usually reduced drastically, but it also removes some paths with small hop count. This effect is also observable by the data shown in Figure 7.6, since propagating acknowledgments along the edges of the tree takes more time.

To conclude, the experiments that we conducted for our proposed algorithms for finding minmax spanners show promising results. Although both algorithms solve the lifetime maximization problem to optimality, one sometimes may prefer one algorithm over the other. Depending on environmental conditions, transmission of broadcast messages may not be feasible. If also the RNG of the communication graph is readily available, for example if nodes know their own and the locations of their neighbors and the edge costs are increasing in distance, then one may prefer MLS over BSPAN. In other cases, for example, if the number of available power levels is quite small

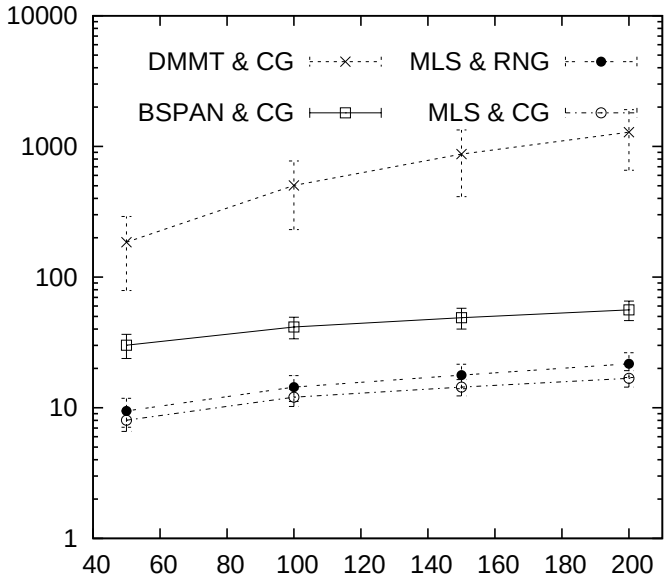


Figure 7.6: Execution time in seconds; error bars show standard deviations over 200 repetitions, including setup stage but excluding notification stage; note the logarithmic scale. For BSPAN the value of $|P|$ is 128. The plot shows data for MLS run on the communication graph CG and its RNG.

and the transmission of broadcast messages is viable, then BSPAN would be preferable. Our simulation results further indicate that BSPAN also provides a tighter bound on its message complexity, so in this sense its runtime behavior is more predictable.

8 SLEEP-SCHEDULING IN SENSOR NETWORKS

In this chapter we present a distributed approximation algorithm for the sleep scheduling problem, which was outlined in Section 2.3.3. We formulate the problem as a linear program that fractionally packs dominating sets of the redundancy graph. Then we apply the Garg-Könemann scheme of Section 3.2.2 to obtain a distributed approximation algorithm. This algorithm requires an oracle for solving an instance of the minimum weight dominating set problem, which is the second topic of this chapter. We consider our novel distributed approximation algorithm for MWDS, which is based on Chvátal's set-cover algorithm, to be the main contribution presented here. The chapter discusses first general algorithmic ideas and then simulation results.

Note that we are not the first to apply the GK scheme to sleep scheduling. The same technique was proposed by Suomela [132] for sleep scheduling in so-called *local graphs*, which allow for an efficient solution to the minimum weight dominating set problem. Berman et al. [11] propose an application of the GK algorithm to obtain a centralized sleep-scheduling algorithm. They also use the centralized version of Chvátal's greedy algorithm for computing the MWDS subproblem in each iteration. Hence, although their approach is similar to ours, it is inherently centralized.

8.1 INTRODUCTION

Recall that in sleep-scheduling nodes can switch between active and inactive states, where the amount of energy consumed while being inactive can be several orders of magnitude smaller than while being active [127]. Hence, sleep scheduling is a viable method for extending the lifetime of sensor networks. For a brief overview of sleep scheduling see Section 2.3.3. Here, we assume that the pairwise redundancy relationship between sensor nodes is captured by a given redundancy graph [45]. Since we want to guarantee complete network coverage at all times, the problem reduces to finding dominating sets of the redundancy graph and assigning active times to them, so that the total active time is maximized.

We assume that the communication graph of the network $G(V, E)$ is connected and undirected. In order to simplify the exposition, we further assume that the redundancy and the communication graphs are identical, i.e., nodes that are neighbors in the communication graph are also mutually redundant and vice versa. However, this assumption is not a real restriction, as long as nodes are aware of their neighbors in the redundancy graph and can communicate with them via short routes in G, which one would expect to be the case in most settings. It is important to note that our approach does not require any specific geometric properties of either graph.

For each dominating set $D \subseteq V$, let us introduce a variable x_D, whose value corresponds to the total activation time of dominating set D. Note that there is possibly an exponential number of variables, although in an optimal solution many dominating sets may have zero activation time. We formulate the *maximum length sleep-scheduling problem* as a linear programming

problem as follows.

$$\begin{array}{lll} \textbf{MSS:} \quad \text{maximize} & \displaystyle\sum_{D} x_D & \\ \text{subject to} & \displaystyle\sum_{D \ni v} x_D \leq 1, & \forall v \in V \qquad (8.1) \\ & x_D \geq 0, & \forall D \end{array}$$

Here, the objective function $\sum_D x_D$ computes the length of the sleep schedule and (8.1) is the battery-capacity constraint for node v. Note that the sum in (8.1) runs over all dominating sets that contain node v. For simplicity, we assume that all nodes have unit capacity. This assumption may be removed as we discuss below.

It is clear that an optimal sleep schedule may have non-integral activation times (see, for example, Figure 2.4 on page 18). If one requires the variables x_D to be integral, then the inapproximability results for the domatic number problem due to Feige et al. [42] imply that the problem can be approximated in polynomial time within a factor of $\mathcal{O}(\log |V|)$, but that it is hard to approximate it within a factor of $(1-\epsilon)\ln|V|$ for any $\epsilon > 0$. The results for the domatic number problem were extended by Moscibroda and Wattenhofer [96], who obtained a distributed, randomized algorithm for the same problem. Their algorithm is very simple and only requires the exchange of information among neighboring nodes, after which nodes randomly decide to join a particular subset. There is a small probability, however, that a solution contains sets that are not dominating sets, so that in applications where certainty about the coverage of the network at all times is required the scheme may not be applicable. Here we do not impose integrality constraints on the x_D, which may also lead to an improved lifetime in some cases, e.g., for the instance of Figure 2.4.

The MSS problem as described above has only been rarely addressed in the literature. Suomela [132] showed that the same hardness of approximation result of the domatic number problem also holds in the case of MSS. Floréen et al. [45] propose an algorithm that only requires constant-size neighborhood information and achieves a constant approximation factor in so-called *marked* graphs. Since here we consider general graphs, we can therefore only aim at a logarithmic approximation factor.

Although we may not be able to compute the length of an optimal schedule efficiently, it is easy to obtain an upper bound on the length of any feasible schedule. From the capacity constraints (8.1) and from the fact that there exists a node in G that can be dominated by at most δ^+ different dominating sets follows that no schedule can achieve a lifetime larger than δ^+. We use this upper bound to compare against in our simulations.

8.2 SLEEP SCHEDULING ALGORITHM

The MSS problem as formulated above is an LP packing problem with a possibly large number of variables. In this sense, it is very similar to the maximum multicommodity flow problem of Section 3.2.2. In this section we

describe how to modify Algorithm GKMMF (see p. 29) to obtain an algorithm for problem MSS that can be implemented in a distributed manner. Let us begin by formulating the dual of problem MSS, which we denote by DMSS. We introduce dual variables y_v for the capacity constraint of node v and obtain the following:

$$\begin{aligned} \textbf{DMSS:} \quad & \text{minimize} && \sum_v y_v \\ & \text{subject to} && \sum_{v \in D} y_v \geq 1, && \forall D \qquad (8.2) \\ & && y_v \geq 0, && \forall v \in V. \end{aligned}$$

Note that validating constraint (8.2) for given values of y_v corresponds to solving an instance of the minimum weight dominating set problem, where the weight of a node v equals y_v.

Let us assume the existence of an approximation oracle for the MWDS problem, i.e., a distributed algorithm that computes a dominating set of weight at most ϕ times the optimum for any given values of the y_v. By identifying the same structure as in problem MMF, we conclude that the Garg-Könemann algorithm, which we applied to problem MMF in Section 3.2.2, can be applied to obtain an approximation algorithm also for MSS. The centralized version of the resulting algorithm is almost identical to Algorithm GKMMF. Here, as previously, we assume the presence of an initiator node which can be used to initiate and coordinate the computation, as well as to detect its termination. In Section 8.3 we propose a distributed approximation oracle for MWDS.

Prior to the execution of Algorithm GKSCHED, the initiator node starts a run of the setup algorithm BEACSET of page 106 to let nodes discover their neighbors and construct a spanning tree. In addition to node identifiers, we let nodes also include the number of their neighbors, which they have discovered so far, in the beacon messages. Hence, every node obtains an estimate of its neighbors and their degrees in the communication graph. Further, each node v keeps track of its tree neighbors $N_T(v)$, i.e., its neighbors in the computed spanning tree. Note that in addition to the father of node v, the set $N_T(v)$ at termination of the beaconing algorithm contains the nodes that have entered the set expecting_reply during the execution of Algorithm BEACSET.

After the termination of the setup stage, the initiator node broadcasts an initiate message in the network, which contains the index of the current iteration. The broadcast of the initialization message is a signal for the nodes to solve the subproblem within the inner loop of the GK algorithm. All later communication is then done as a sequence of network-wide broadcasts and convergecasts in the spanning tree that was computed in the setup stage, where the same signal of initiating the solution of the subproblem is also used at the beginning of all other iterations.

We denote by $y_v(k-1)$ the value of the dual variable y_v at the beginning of iteration k and by $y_v(k)$ its value at its end, and define $x_D(k)$ analogously. In every iteration k, the oracle computes a dominating set $D(k)$, depending on the current node weights $y(k-1)$. Then, according to the centralized

Algorithm GKSCHED: Distributed GK algorithm for sleep scheduling

```
1  node v with variables father, ε, k, active_slots, is_initiator, N⁺(v),
   N_T(v), weight_sum, status;
2  at start
3      β ← (1 + ε) ((1 + ε) |V|)^{−1/ε}; k ← 1;
4      for u ∈ N⁺(v) do y_u(0) ← β;
5      for u ∈ N_T(v) do status[u] ← processed;
6      if is_initiator then enter state SEND_INITIALIZE;
7      else enter state MWDS_TERMINATED;
8
9  in state SEND_INITIALIZE          // forward initialize to tree children
10     weight_sum ← 0;
11     if k > 1 then
12         for u ∈ N⁺(v) do
13             if u ∈ D(v) then y_v(k − 1) ← (1 + ε)y_v(k − 2);
14             else y_v(k − 1) ← y_v(k − 2);
15         end
16         if v ∈ D(v) then active_slots(k − 1) ← 1;
17         else active_slots(k − 1) ← 0;
18     end
19     for w ∈ N_T(v) \ {father} do
20         status[w] ← wait;
21         unicast initialize(k) to w;
22     end
23     enter state PROCESSING;
24
25 in state PROCESSING
26     for u ∈ N⁺(v) do w(u) ← y_u(k − 1);
27     . . .;                                  // execute MWDS algorithm
28
29 in state MWDS_TERMINATED    // ASYNMWDS locally terminated
30     if is_initiator then
31         if weight_sum < 1 then // results from Algorithm ASYNMWDS
32             k ← k + 1;
33             enter state SEND_INITIALIZE;
34         end
35         else for w ∈ N_T(v) do unicast gk_terminate() to w;
36     end
37     if gk_terminate() is received from father then
38         for w ∈ N_T(v) \ {father} do unicast gk_terminate() to w;
39     end
40     if initialize(k′) is received from father then
41         k ← k′;          // here: k′ = k + 1 for all but first iteration;
42         enter state SEND_INITIALIZE;
43     end
44
```

GK scheme along the lines of Algorithm GKMMF, we would increase the primal variable $x_{D(k)}$. At termination, the value of each primal variable x_D has the form $n_D / \log_{1+\epsilon} \frac{1+\epsilon}{\beta}$, where n_D is the number of times the dominating set D was returned by the oracle. In Algorithm GKSCHED we store the value of each x_D implicitly, by letting nodes record in which iterations they were member of the computed dominating set. More precisely, each node maintains an array called active_slots, whose entries are binary values that when set to true indicate that the node needs to be active in a particular time slot. Using this information, the sleep schedule can later be executed by the network. Note that the length of a time slot corresponds to the value of a single increase in the Garg-Könemann algorithm, which is $1/\log_{1+\epsilon} \frac{1+\epsilon}{\beta}$.

Algorithm GKSCHED delays the update of the variables until the begin of the following iteration. A node detects that a new iteration has begun when it receives an initialization message with an increased iteration counter. At the beginning of iteration k and before forwarding the notification message, each node v multiplies the dual variables for the nodes in the set $N^+(v) \cap D(k-1)$ by $(1+\epsilon)$ to obtain their value for the current iteration. This update requires that all nodes keep track of which nodes in their neighborhoods enter the dominating set in each iteration. Our MWDS algorithm maintains a set $D(v) \subseteq N^+(v)$ at each node v for this purpose. Note that the delay of the update for the dual variables is necessary since the initiator is the only node that can detect termination of an iteration, i.e., detect when the oracle has terminated.

At the end of iteration k, the initiator has obtained the total weight of all nodes in the network as determined by $y(k-1)$. If the total weight is at least one, the initiator detects termination and then initiates a network-wide broadcast to notify nodes to start executing the schedule. If the total weight is smaller than one, the initiator starts a new iteration by transmitting to its children in the spanning tree.[1]

For the parameter L in the scaling factor of Algorithm GKMMF, which is discussed on page 29, it is sufficient to choose $L = |V|$, which is the maximum number of nodes in any dominating set. We therefore assume that nodes know the network size $|V|$, which could be otherwise included in the initialization message, in order to set the value of β as required by Algorithm GKMMF.

Theorem 15. *Algorithm* GKSCHED *computes a sleep schedule of length at least* $(1-\epsilon)^2/\phi$ *times the optimum, where* $0 < \epsilon < 1$ *and* $\phi > 1$ *is the approximation factor of the* MWDS *algorithm. The algorithm terminates after at most* $\lceil (|V|/\epsilon) \log_{1+\epsilon} |V| \rceil$ *iterations.*

Proof. The result follows from the extension of the Garg-Könemann algorithm [48] to be used with an approximation oracle. For details see [134]. □

Corollary 2. *The message complexity of Algorithm* GKSCHED *is*

$$O\left((K(V,E) + |V|) \ \frac{|V|}{\epsilon} \log_{1+\epsilon} |V| \right),$$

[1] This is the termination condition for the version of the algorithm in [48] for general packing LPs, which is different from the version for MMF.

where $K(V, E)$ is the message complexity of the MWDS *approximation algorithm.*

Our distributed MWDS algorithm discussed in Section 8.3 achieves an approximation factor of $\phi = \mathcal{O}(\ln \Delta^+)$, so that the combined algorithm is asymptotically optimal. However, by choosing a different MWDS algorithm it is likely that better approximation guarantees can be established for certain graph families. In Section 8.4 we also discuss experimental results that indicate that one is able to obtain schedules whose lengths approach the upper bound δ^+ already for reasonably large ϵ. Since the number of iterations required has a major effect on the number of messages transmitted, we also consider this aspect in the experiments.

The MWDS algorithm presented below requires $\mathcal{O}(|V|\Delta^2)$ messages in the worst case. Depending on the problem instance, the combined algorithm may also be more efficient than a simple centralized algorithm that first gathers complete information of the graph, solves the instance at the initiator and then distributes the sleep schedule in the network. We expect this to be the case if the maximum degree Δ is rather small, while the network itself is large.[2]

Extension to arbitrary capacities Similarly to the implementation of the GK scheme for the multicommodity flow problem, we can also consider the case of arbitrary node capacities. Recall that the GK scheme for MMF augments the flow in each iteration by an amount that is equal to the minimum edge capacity along the shortest path. Hence, if the initiator is informed of the minimum node capacity among all members in the dominating set, it can inform all nodes in the following iteration of the minimum capacity, which then update their activity times accordingly. Obtaining the value of the minimum capacity for each iteration can be easily achieved by appending a field to the convergecast of gk_terminate messages at termination. Note, however, that after the modification the length of the k-th time slot equals $c_{\min}/\log_{1+\epsilon}\frac{1+\epsilon}{\beta}$, where $c_{\min}$ is the minimum capacity of any node in dominating set $D(k)$. This means that after the modification time slots can be of different lengths, which may be undesirable in practice.

8.3 MINIMUM WEIGHT DOMINATING SET APPROXIMATION

We now develop our distributed approximation algorithm for the MWDS subproblem that is solved in every iteration of Algorithm GKSCHED. Our algorithm is based on the primal-dual framework for LP covering problems. It is also inspired by parallel algorithms for weighted set-cover proposed by Rajagopalan and Vazirani [116]. Khuller et al. [75] present similar parallel algorithms for set cover and vertex cover. Both approaches employ the duality framework which can be also used to analyze Chvátal's greedy algorithm, as demonstrated in Section 3.2.2. In this sense, our algorithm is also similar to parallel algorithms for weighted set cover proposed even earlier by Berger

[2]This comparison assumes the $\mathcal{CONGEST}$ model (see p. 44) and that no compression of the sleep schedule is allowed.

et al. [10]. However, the model of computation for parallel algorithms is inherently unsuitable for the distributed and asynchronous nature of wireless networks. This is due to the fact that parallel algorithms are usually based on a shared-memory architecture which allows synchronous and error free communication with no delay between the processors.

Recently, also local algorithms have been proposed for computing small dominating sets in graphs. Since the model of distributed computation of local algorithms (see Section 4.1) is very different from ours, we do not consider these for comparison. For an overview and relevant local algorithms for vertex cover and facility location, which is a generalization of set-cover, we refer to the thesis of Moscibroda [95].

Recall that we can formulate the MWDS problem as an instance of the minimum weight set-cover problem by choosing $U = V$ to be the set of elements to be covered and $\mathcal{S} = \{N^+(v) \mid v \in V\}$ to be the collection of candidate cover sets. The weight of a single set is then defined as $w(N^+(v)) = w(v)$. In this section we discuss a distributed algorithm that is based on the set-cover algorithm PDSC (see p. 27), does not require synchronous operations and is also somewhat resilient to message loss, assuming an underlying retransmission scheme. Further, the algorithm makes use of the wireless broadcast advantage by letting nodes opportunistically process overheard messages transmitted between neighboring nodes.

As mentioned in Section 4.2.4, besides sleep scheduling, dominating sets can be useful for a wide range of applications in wireless multihop networks, for example for clustering and routing. Consequently, many distributed and centralized algorithms can be found in the literature. See the survey by Blum et al. [17] for an overview. In comparison to other algorithms, we note that our algorithm for approximating MWDS does not require network-wide synchronization or any geometric restrictions on the graph. Moreover, we inherit the approximation guarantee from Chvátal's algorithm [31]. We first present the general ideas and then elaborate on the algorithm.

8.3.1 Problem formulation

In Section 3.2.2 we introduced Algorithm PDSC as a method which operates on a pair of primal and dual linear programs, where here the primal is the linear relaxation of problem MWDS. Let us introduce binary variables z_v for each v that when set to 1 indicate that a node is chosen to be in the dominating set. Denote the weight of node v by $w(v)$. We can then formulate the minimum weight dominating set problem as follows.

$$\begin{aligned} \textbf{MWDS:} \quad \text{minimize} \quad & \sum_{v \in V} w(v)\, z_v & & \\ \text{subject to} \quad & \sum_{u \in N^+(v)} z_u \geq 1, & \forall v \in V \quad & (8.3) \\ & z_v \in \{0,1\}, & \forall v \in V & \end{aligned}$$

The problem asks for a minimum-weight subset of nodes such that every node is covered by at least one node in its extended neighborhood $N^+(v)$. By removing the integrality constraints for the z_v, we can form the dual problem

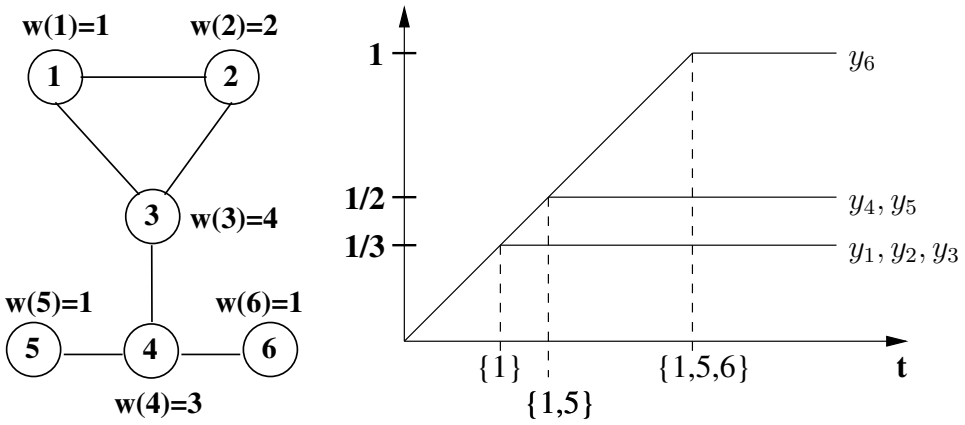


Figure 8.1: Example of continuous-time greedy algorithm for MWDS; the labels on the x-axis mark nodes entering the dominating set, while the plot shows the evolution of the dual variables until they become frozen.

of the linear relaxation of MWDS.

$$\textbf{DMWDS:} \quad \text{maximize} \quad \sum_{v \in V} y_v$$
$$\text{subject to} \quad \sum_{u \in N^+(v)} y_u \leq w(v), \qquad \forall v \in V \qquad (8.4)$$
$$y_v \geq 0, \qquad \forall v \in V$$

Here, we introduced dual variables y_v for each coverage constraint (8.3). Each dual variable y_v can be interpreted as a *bid* that node v is willing to pay to become covered, as we describe below.

8.3.2 Distributed voting scheme

We now assume that each node knows the weight and degree of each of its neighbors and can also identify its neighbors in a spanning tree rooted at the initiator, e.g., by executing the setup stage described in Section 8.2. During an execution of the algorithm, a node is in one of three *cover states*: *uncovered*, *covered*, or *dominator*. We let all nodes v maintain a set of uncovered nodes $U(v)$ and dominators $D(v)$ within their one-hop neighborhoods $N^+(v)$. Initially, $U(v) = N^+(v)$ and $D(v) = \emptyset$.

Recall that Algorithm PDSC starts from an all-zero solution for primal and dual variables and then raises the duals until they become frozen, i.e., they appear in a dual constraint (8.4) for some v which has become tight. At this point the first node, say v, is added to the dominating set, which corresponds to $N^+(v)$ being added to the set cover. At the same time, the state of all nodes in $N(v) = N^+(v) \setminus \{v\}$ changes to covered and the y_v variables for all nodes in $N^+(v)$ become frozen. The algorithm then continues to raise the other dual variables which are not frozen yet and only considers their values when determining which constraint gets tight next.[3] Figure 8.1 shows a

[3]The term "tight" is a misnomer here, since some dual constraints may actually become violated in the process.

small example instance that illustrates how in the continuous-time algorithm the dominating set and the values for the dual variables evolve over time.

For a given set of uncovered nodes $U(v) \subseteq N^+(v)$, let us define the *price* of a node v as

$$p(v) := \begin{cases} \frac{w(v)}{|U(v)|} = \frac{w(v)}{\text{span}(v)} & \text{if } U(v) \neq \emptyset, \\ \infty & \text{otherwise,} \end{cases}$$

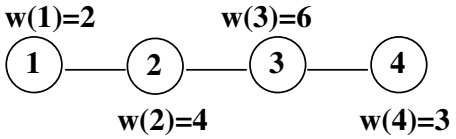
where we also denote the number of uncovered nodes $|U(v)|$ in v's neighborhood as $\text{span}(v)$. In the continuous-time primal-dual algorithm a node v would become a dominator at time $p(v)$ if the set $U(v)$ stayed unchanged until then. If the set $U(v)$ changes before time $p(v)$, the price $p(v)$ can only increase, since the number of uncovered neighbors a node has can only decrease. Note that a change in price before time $p(v)$ can only be caused by another node $u \in N_2^+(v)$ entering the dominating set, which causes a previously uncovered node $w \in N^+(v)$ to become covered. In Figure 8.1, for example, the price of node 1 is initially $1/3$, which is the minimum within its two-hop neighborhood. The price of node 3 raises from 1 to 4 when node 1 enters the dominating set at time $1/3$.

A simple implementation based on the observations above would let nodes enter the dominating sets when they have the minimum price within their two-hop neighborhoods. This algorithm, however, is the same as algorithm SYNMWDS, which requires clock synchronization. Instead, consider the following voting scheme, which basically inverts the flow of information in Algorithm SYNMWDS. Rather than letting a node actively monitor the prices in its two-hop neighborhood $N_2^+(v)$, we let each of its neighbors u *vote* for v, if v has the minimum price in u's neighborhood $N^+(u)$. Whenever a node v has received votes from all nodes in $U(v)$, it can determine that it should become a dominator, assuming $|U(v)| > 0$. Obviously, this scheme also requires some dissemination of node prices within the neighborhood of each node. As we argue below, by including additional information in votes the resulting overhead can be reduced.

In the implementation of our algorithm we add a *limit* value to each vote. In this sense, a vote sent by u to v is conditional on $p(v)$ not exceeding the limit. When v receives a vote with a limit larger than its current price $p(v)$, it infers that u is indifferent to any change in v's price unless it exceeds the limit. This is due to the fact that the prices of nodes can only increase during the algorithm and therefore u will not change its voting decision at least until the limit is reached. Hence, node v will not send any price update to u until $p(v)$ exceeds the limit. In our implementation we set the limit to the second-lowest price in $N^+(u)$ that u knows about. In this sense a vote corresponds to a contract between the sender and the recipient of the vote.

Consider again the example in Figure 8.1. Initially, when all nodes are uncovered, node 3 votes for node 1 and includes a limit value of $2/3$ in its vote. This value equals the price of node 2, which has the second lowest price in the neighborhood of node 3.

More formally, we call the set of nodes in $N^+(v)$ that have sent votes to v with limit values larger than $p(v)$ the *supporters* of v and denote its set of supporters by $S(v)$. A node u enters $S(v)$ when u sends a vote with limit



i	1	2	3	4
$p(i)$	1	4/3	2	3/2

i	1	2	3	4
$p(i)$	∞	4	3	3/2

Figure 8.2: Example for voting procedure; the two tables on the right show the initial (top) and the node prices after node 1 becomes dominator.

$l \geq p(v)$ to v. Node u leaves $S(v)$ when v's price increases above l. However, node u may later re-enter $S(v)$ when it casts a new vote with a higher limit value. Once we have $S(v) = U(v)$ and $|S_v| > 0$, node v becomes a dominator. Note that also nodes that were previously covered may enter the dominating set if this condition is satisfied.

Consider the example in Figure 8.2. Node 3 initially votes for node 2 with a limit value of 3/2, which is the price of node 4, and thus enters $S(2)$. When 2 itself becomes covered due to 1 entering the dominating set, the price of 2 increases and it not longer achieves the minimum price in the neighborhood of node 3. Hence, node 3 sends a new vote to 4 with a limit value equal to its own (new) price. Let us emphasize two important properties of the algorithm.

1. Node prices can only increase.

2. If a node receives votes from all its uncovered neighbors, it has the minimum prize in its two-hop neighborhood.

To reduce the number of price update messages even further, we let nodes only inform those neighbors of a price update that are forced to leave the set of supporters due to the increase in price. Hence, instead of maintaining the actual prices of its neighbors, each node keeps only lower bounds for these prices. Unfortunately, some information is lost since outdated price information may lead to *invalid votes* with too small limit values. If this situation occurs, i.e., if a node v later receives a vote from u with a too small limit, v replies with a message informing of an update in v's price. Our experiments indicate that this method is able to reduce the number of messages required, although it may require two additional votes for each price update in the worst case. As an invariant of the algorithm, we require that each node knows the correct value of its own price, which we achieve by letting nodes inform neighbors when they become covered or dominator.

Allowing lower bounds for node prices has additional benefits. In our implementation we use unicast transmissions for all message types, including price updates. However, we let nodes also overhear price updates that are sent between neighboring nodes in order to update the information in their own neighbor lists. By overhearing price updates nodes may improve the lower bounds on the prices of their neighbors and therefore also change their voting decision. This opportunistic behavior aims at exploiting the wireless broadcast advantage.

8.3.3 Implementation

We now describe the distributed implementation of Algorithm ASYNMWDS given on pages 126 and 127. The second part on page 127 consists of functions that are called from the main part on the previous page. Note that the expression $v \in U(v)$? U : C in lines 7 and 78 of the pseudocode evaluates to U if v is uncovered and to C otherwise.

We let each node v maintain a neighbor list containing entries for v and all its neighbors. More precisely, the list NL of v contains a tuple $\mathrm{NL}_u = (\mathrm{id}, \mathrm{weight}, \mathrm{degree}, \mathrm{span}, \mathrm{limit}, \mathrm{notify})$ for each node $u \in N^+(v)$, where id $= u$, *degree* is the degree of u, *span* is the number of uncovered nodes in $N^+(u)$, *limit* is the highest price limit received in any vote from u for v, and *notify* is a Boolean variable which indicates whether u has to be notified of a change in $p(v)$. All nodes keep their neighbor lists sorted in increasing order of price, breaking ties based on node identifiers. Additionally, each node v maintains the sets $U(v)$, $S(v)$ and $D(v)$ as described above.

Based on information in the neighbor list, a node v can compute its own price, estimate the price of nodes in $N(v)$, and determine which of its neighbors are currently voting for v, i.e., its set of supporters. To simplify the exposition, we refer to the information stored in the list as the actual price values, although these are in fact lower bounds, as described earlier. In addition to referring to an entry in NL on a per-node basis, we use the notation $\mathrm{NL}(k)$ to denote the kth entry from the beginning of the neighbor list of a particular node v, where $1 \leq k \leq |N^+(v)|$. We also refer to the value of a single field in $\mathrm{NL}(k)$ by noting the name of the field. For example, $\mathrm{NL}(1)$[id] refers to the identifier of the node that achieves minimum price in the neighbor list.

Initialization (lines 46-54)
Initially, we have $\mathrm{NL}_u = (u, w_u, \delta_u, \delta_u + 1, 0, \mathrm{false})$ for all $u \in N^+(v)$, where δ_u is the degree of u. Node v calculates its price as $p_v = \frac{w_v}{\delta_v + 1}$ and initializes $D(v)$, $S(v)$ and $U(v)$ accordingly. After receiving an initialization message, v consults its neighbor list NL and votes for the node at the head of the list. However, it only sends an actual message if $v \neq \mathrm{NL}(1)$[id].

Voting (lines 1-8 and 55-73)
So after receiving the initializer, v sends the message vote(limit) to the node with id $\mathrm{NL}(1)$[id], say $u \neq v$, where limit $= p_{\mathrm{NL}(2)[\mathrm{id}]}$. Recall that we assumed G to be connected and hence $|N^+(v)| > 1$. When u receives this vote, it first checks if the vote is valid, i.e., it checks whether it holds that limit $\geq p_u = \frac{w_u}{\delta_u + 1}$. If the vote is valid, u records v entering the set of supporters $S(u)$ and sets the limit value NL_v[limit]. If v has the lowest price in its own list, i.e., if $v = u$, then node v performs exactly the same modifications to its own neighbor list without sending the vote.

Whenever a node v receives a valid vote or when its price changes due to a neighbor becoming covered or a dominator, it checks whether all uncovered nodes in $N^+(v)$ are currently voting for v and if there is at least one such node. If this is the case, i.e., if $S(v) = U(v)$ and $|S(v)| > 0$, then v decides to become dominator and informs all its neighbors by sending each of them a dominator($N(v)$) message. Node v includes a list of its one-hop neighbors in

Algorithm AsynMWDS: Distributed MWDS algorithm at v (part 1)

```
1  if v receives vote(limit) from u then
2      if NL_v[weight]/NL_v[span] < limit then                    // vote is valid
3          S(v) ← S(v) ∪ {u};
4          if NL_u[limit] < limit then NL_u[limit] ← limit;
5          if chk_covered&voted() then become_dominator();
6      end
7      else unicast price(NL_v[span], v ∈ U(v) ? U : C) to u;
8  end
9  if v receives price(nspan, nstate) from u then      // unicast or overheard
10     old_first ← NL(1);
11     if u ∈ U(v) and nstate = C then
12         U(v) ← U(v) \ {u};NL_v[span] ← NL_v[span] − 1;   // u covered
           for w ∈ S(v) do
13             if NL_w[limit] < NL_v[weight]/NL_v[span] then
14                 NL_w[notify] ← true; S(v) ← S(v) \ {w};
15             end
16         end
17     end
18     NL_u[span] ← nspan;
19     if v ∉ D(v) and chk_covered&voted() then become_dominator();
20     else chk&terminate();
21     if v ∈ U(v) then
22         if old_first != NL(1)[id] or (old_first = u and msg unicasted)
           then cast_vote();
23         if old_first = NL(1)[id] and old_first = v then
24             NL_v[limit] ← NL(2)[weight]/NL(2)[span];
25             if chk_covered&voted() then become_dominator();
26         end
27     end
28 end
29 if v receives dominator(N(u)) from u then
30     D(v) ← D(v) ∪ {u}; NL_u[span] ← 0;
31     if |U(v) \ N+(u)| < NL_v[span] then
32         NL_v[span] ← |U(v) \ N+(u)|;
33         for w ∈ S(v) with NL_w[limit] < NL_v[weight]/NL_v[span] do
34             NL_w[notify] ← true; S(v) ← S(v) \ {w};
35         end
36     end
37     if v ∈ U(v) then                                  // v just became covered
38         for w ∈ N(v) \ (N+(u) ∪ D(v)) do
39             send price(NL_v[span], C) to w;      // send immediate update
               NL_w[notify] ← false;
40         end
41     end
42     U(v) ← U(v) \ N+(u);
43     if v ∉ D(v) and chk_covered&voted() then become_dominator();
44     else chk&terminate();
45 end
```

Algorithm ASYNMWDS: Distributed MWDS algorithm at v (part 2)

```
46 function void initialize at v  // run once after wake-up message received
47     D(v) ← ∅; U(v) ← N+(v); S(v) ← ∅; NL ← ∅;
48     for u ∈ N+(v) do
49         NL ← NL ∪ (id: u, weight: w_u, degree: δ_u,
50             span: δ_u + 1, limit: 0, notify: false);
51     end
52     schedule price_update_timer() after T1 seconds;
53     schedule cast_vote() after T2 seconds;
54 end
55 function void cast_vote() at v        // sends vote to lowest-price neighbor
56     limit ← NL(2)[weight]/NL(2)[span];
57     if NL(1)[id] ≠ v then send vote(limit) to NL(1)[id];
58     else
59         NL_v[limit] ← limit; S(v) ← S(v) ∪ {v};
60         if chk_covered&voted() and v ∉ D(v) then
61             become_dominator();
62         end
63     end
64 end
65 function void become_dominator() at v
66     D(v) ← D(v) ∪ {v}; U(v) ← ∅; NL_v[span] ← 0;
67     for u ∈ N(v) do send dominator(N(v)) to u;
68     stop price_update_timer(); chk&terminate();
69 end
70 function bool chk_covered&voted() at v
71     if S(v) = U(v) and |S(v)| > 0 then return true;
72     else return false;
73 end
74 function void price_update_timer() at v
75     for u ∈ U(v) with NL_u[notify] = true do
76         NL_u[notify] ← false;
77         if NL_u[limit] < NL_v[weight]/NL_v[span] then
78             send price(NL_v[span], v ∈ U(v) ? U : C) to u;
79         end
80         if v ∉ D(v) then
81             schedule price_update_timer() after T1 seconds;
82         end
83     end
84 end
85 function void chk&terminate() at v // termination test and convergecast
86     if U(v) = ∅ and all child nodes in spanning tree have reported
87        weight of their subtree for current GK iteration then
88         send mwds_terminate(weight_sum) to father;
89         enter state MWDS_TERMINATED;
90     end
91 end
```

the dominator message to let each recipient $u \in N(v)$ update its own price, depending on the number of mutual neighbors $N^+(v) \cap N^+(u)$ that were just covered by v.

If the vote received by v from u is invalid, i.e., if limit $< p_v$ holds, then v immediately replies with a price update price(span, state), where span is the current span of v and state is v's current cover state. This price update informs node u which originated the vote of stale information in its neighbor list and lets it update the span and cover state for v. The update may also cause a price update for u if v indicated it is covered but $v \in U(u)$ prior to receiving the price update from v. In the case of an invalid vote no limit value for u is recorded by v and $S(v)$ remains unchanged.

Price update (lines 9-28 and 74-84)

Whenever the price of a node v changes due to a decrease in $|U(v)|$, unless v itself was covered, v iterates through its set of supporters and sets the notify bit for those neighbors $u \in N(v)$ that are forced to leave $S(v)$ because v's price just exceeded the vote limit $\mathrm{NL}_u[\mathrm{limit}] < p(v)$. Each node periodically sends price(span, state) messages to neighbors which have the notify bit set to true, after which it is set to false. The length of the update reporting period is determined by a parameter which we call *price update interval* (PUI), whose effect we evaluate in network simulations. It was deemed necessary to introduce a buffering of price updates to avoid message drops due to congestion, which leads to IFQ overflows.

Upon receiving a price update, each node v which is uncovered checks whether the update caused a change in the lowest-price entry NL(1). Let u be the neighbor which previously had the lowest entry and let $u \neq v$. If there was a change, i.e., if $u \neq \mathrm{NL}(1)[\mathrm{id}]$, then v casts a vote for the new best entry as described above. If it remained unchanged then v sends a new vote anyway, if the price update message originated from u itself. This is necessary since the limit value for v in the neighbor list of u is outdated and u has removed v from its supporters $S(u)$ prior to sending the price update. Hence, v has to confirm its voting for u by sending a new vote with an updated limit value and thus re-enter $S(u)$. In the trivial case that $u = v$, then v only records the new limit value for NL_v.

Tracking dominators (lines 29-45)

When an uncovered node v receives a dominator($N(u)$) message from a neighbor u, it informs those neighbors which it does not share with u of becoming covered if these are not dominators already. More precisely, v sends a message price(new_span, state), where state= covered and new_span is its updated span, to each neighbor in $N(v) \setminus (N(u) \cup D(v))$, independently of their limit value. Note that these price updates are not buffered, since we want every node to have valid information on its own price, which requires a correct value for its span.

Also a previously covered node can not ignore a dominator message but must track the members of the dominating set in its neighborhood and observe changes in its own price. Further, it is possible that a neighboring dominator of v covers the missing nodes from $S(v)$, in which case v itself becomes dominator.

Termination (lines 85-91)
Algorithm ASYNMWDS performs a convergecast for termination detection at the initiator, where each node waits until it itself and all its neighbors are covered before it forwards the termination message. Only after the termination of the convergecast one can be sure that all nodes know the dominators in their neighborhood, which is required for algorithm GKSCHED to operate correctly. The convergecast also computes the total weight in the network, i.e., the objective value of the current dual solution, which is required for the termination condition of the Garg-Könemann scheme.

Analysis
Let us now consider the approximation guarantee and the message complexity of the algorithm. From the discussion above we obtain the following two lemmas.

Lemma 4. *At any time during an execution of Algorithm* ASYNMWDS *every node v knows its span and hence its current price. Every node also maintains a valid lower bound on the price of its neighbors.*

Lemma 5. *In Algorithm* ASYNMWDS *a node v enters the set of dominators if and only if it has the minimum price among all nodes in $N_2^+(v)$, breaking ties by node identifiers, and there exists an uncovered node in $N^+(v)$.*

Theorem 16. *Algorithm* ASYNMWDS *obtains a dominating set of weight at most H_{Δ^+} times the weight of a* MWDS*, where H_n is the n-th harmonic number.*

Proof. We only need to show that algorithms ASYNMWDS and PDSC when applied to the MWDS problem are equivalent . Let D_d be the set obtained by the distributed algorithm and D_c be the set obtained by Algorithm PDSC applied to the corresponding set-cover instance. Suppose that $D_d \not\subseteq D_c$ and order the nodes in D_d in the order in which they entered the set. Let v be the first node such that $v \in D_d$ and $v \notin D_c$ and let $D'_d \subseteq D_c$ be the nodes that enter D_d prior to v.

At the time when v entered D_d, it had the smallest price among all nodes in $N_2^+(v)$. This implies that in the run of algorithm PDSC, no other node in $N_2^+(v)$ can enter $D_c \setminus D'_d$ before v, since prices can only increase as nodes become covered. However, there must be one, since one node in $N^+(v)$ would remain uncovered – the one that first became covered when v was added to D_d – and hence we must have $v \in D_c$, which shows $D_d \subseteq D_c$. The other direction $D_c \subseteq D_d$ follows directly by the update rules for the primal-dual algorithm and Lemma 5. □

Theorem 17. *The message complexity of* ASYNMWDS *is $\mathcal{O}(|V|\Delta^2)$.*

Proof. Each node can change its price at most Δ^+ times. When the price of a node changes, it can cause the transmission of at most Δ price updates, each of which can result in the transmission of at most one vote. Hence, the total number of vote and price update messages is $\mathcal{O}(|V|\Delta^2)$. At most $|V|$ nodes may enter the dominating set. Each such node sends at most Δ dominator messages, which each contain information on at most Δ neighbors. □

8.4 EXPERIMENTAL EVALUATION

When evaluating our approach for approximating the sleep-scheduling problem, we first considered the quality of solutions obtained by the combination of the GK scheme and the MWDS approximation algorithm in a centralized setting. In our experiments we found that the length of schedules obtained by the algorithm is better than one would expect from the guarantees in Theorems 15 and 16. We also measured the number of iterations required, since this value determines the runtime of the complete algorithm.

After establishing the effectiveness of the approach, we turned to NS2 network simulations of the combined distributed algorithms GKSCHED and ASYNMWDS. The goal of the simulations was chosen to estimate the actual message complexity and execution time required by the algorithm. The findings indicate that Algorithm ASYNMWDS may perform much better in practice than what one would expect from Theorem 17, which could suggest that this bound is not tight.

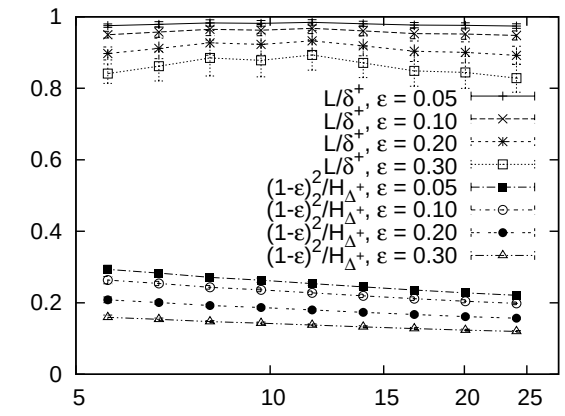
The set of problem instances we used in the experiments were formed by 20 random disk graphs with 150 nodes, whose locations were drawn from a uniform distribution in square areas of various sizes. The dimension of an area was chosen so that one obtained a given expected node degree, which is representative of the node density, disregarding boundary effects. As before in previous experiments, edges in the communication graph were determined by the NS2 default transmission range. Instances whose node locations did not yield a connected communication graph were discarded.

Centralized GK scheme

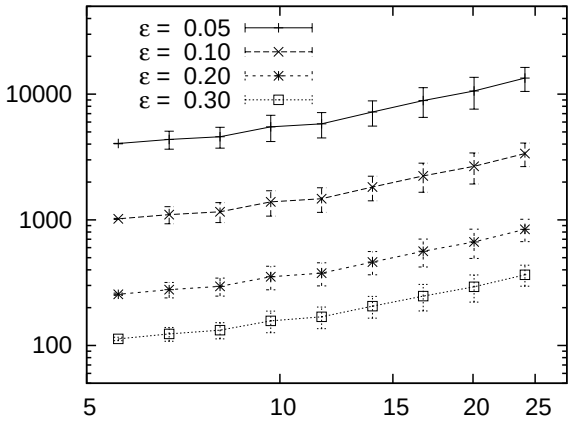
Our first goal was to compare the length of schedules obtained by the centralized implementation of the algorithm to its approximation guarantee. The complete algorithm, GK scheme and greedy MWDS algorithm, was implemented using Matlab. Since the task of finding optimal solutions to all instances is very computation intensive, we computed for comparison the instance-dependent upper bound δ^+ on the length of an optimal schedule instead. Note that this value is somewhat related to the density of nodes in the network, assuming a uniform distribution of nodes.

The outcome of our first set of experiments is shown by Figure 8.3(a). The plot shows data for different values of ϵ and compares the achieved length of the schedules to their upper bound δ^+. Even for relatively large values of ϵ one observes that the total lifetime is much closer to its upper bound than what one would expect from the approximation guarantee. Based on the data, one may conjecture that using the approximation oracle instead of an optimal solution in the GK algorithm does not prevent the algorithm from finding good solutions.

The number of iterations required is shown in Figure 8.3(b). Note the logarithmic scale. Not surprisingly, the choice of ϵ has a major effect on the number of iterations. However, the number of iterations also increases with node density. This can be explained by the fact that for increasing node degrees the total achievable schedule length also increases. The errorbars shown in all plots show the standard deviation over the 20 instances for the



(a) Lifetime/δ^+ for different ϵ



(b) Number of iterations for different ϵ

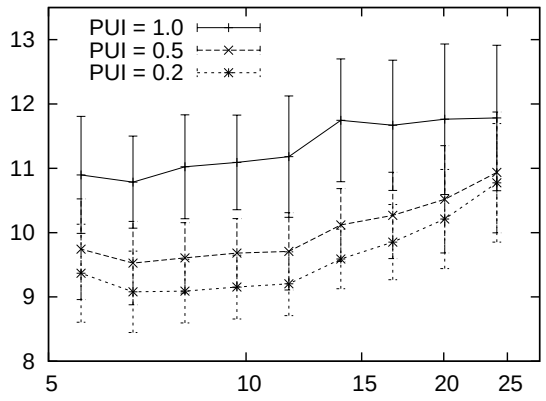
Figure 8.3: Algorithm performance depending on node density; the x-axis shows the expected node degree on a logarithmic scale disregarding area boundary effects; plot a) also shows the approximation guarantee.

same node density.

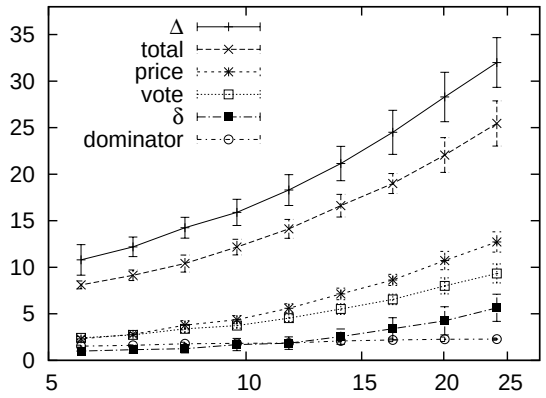
Network simulations

We implemented the combined Algorithm GKSCHED-ASYNMWDS as a protocol agent in NS2 and performed simulations for the same set of problem instances as above. Based on the experiments with the centralized algorithm we fixed $\epsilon = 0.2$. Choosing the same instances had the benefit of being able to test the correct operation of the algorithm by checking whether the results matched those obtained by the centralized implementation.

We evaluated algorithm performance in terms of the number of control messages required and simulated running time per iteration. One may expect that the execution time of the distributed algorithm largely depends on the price-update interval length PUI. Figure 8.4(a) shows the average duration of a single iteration in Algorithm GKSCHED when one varied the value of PUI. From the data it appears the actual value of PUI only has a minor effect on the runtime of algorithm ASYNMWDS. This seems to be particularly the



(a) Seconds per iteration for different PUI length



(b) Messages per node per iteration according to type for PUI=1.0

Figure 8.4: Number of control messages and time required per iteration for networks of varying node density; the x-axis shows the expected node degree on a logarithmic scale disregarding area boundary effects; plot b) also includes retransmissions caused by collisions.

case when the network is dense. Based on this observation, we settled on a value of one second for the remaining experiments.

Figure 8.4(b) shows the average total number of messages per node per iteration required by AsynMWDS. One can see that the total number of messages per node generally lies between the maximum and the average degree in the graph. The figure also breaks down the total number of messages according to their type. It is also apparent from Figure 8.4(b) that price updates have the largest contribution to the number of messages, followed by votes and dominator messages. We note that one modification that may be worthwhile to consider would be to replace unicast transmissions of price updates by broadcast transmissions with selective acknowledgments, which are sent in reply by supporter nodes. One would expect that this modification would lead to a minor reduction in the number of price updates, possibly offset by a more complicated MAC layer protocol.

9 CONCLUSIONS

This thesis discussed distributed algorithms for multihop networks that are based on a formal model of an underlying optimization problem. We started with a brief introduction in Chapter 1 and proceeded in Chapter 2 to outline models, major problems and challenges in multihop wireless networks. In Chapter 3 we continued with a short overview of mathematical programming techniques from linear and convex optimization theory that are relevant in this context. These methods form tools that were later applied to practical network optimization problems. We then formalized the model of distributed computation that forms the base for the algorithms proposed in this thesis. The model and several example algorithms found from the literature were discussed in Chapter 4, which concluded the introductory part of the thesis.

The remaining chapters presented own work on distributed optimization algorithms. In Chapter 5 we first addressed the problem of minimizing network congestion subject to routing demands and formulated a linear programming model of the problem. Then we designed a distributed algorithm based on an LP approximation technique due to Young [154]. Further, we provided results of extensive network simulations, which we performed to evaluate our algorithm using the well-known simulator NS2. We then turned to the problem of fairness of routing when there are several source-destination pairs that use a multipath routing algorithm. In Chapter 6 we proposed an algorithm that is based on the dual-decomposition technique and lets sources continuously update their routes based on node-congestion information.

The following two chapters discussed the part of our work relevant to sensor networks. Energy-efficient operation of these networks is an important design goal when considering almost all applications. Therefore, in Chapter 7, we considered a graph-theoretic model for the problem of maximizing the sensor-network lifetime. We proposed two algorithms, which we evaluated and compared by network simulations. The computed communication graph is guaranteed to be connected and to achieve maximum lifetime under the assumption of uniform load on the nodes. For applications that lead to short duty cycles, i.e., when the time that is spent by nodes in idle states dominates, in order to achieve long lifetime it is important to let sensor network nodes remain in a low-power sleeping state whenever possible. We approached the resulting sleep scheduling problem in Chapter 8, where we proposed a distributed algorithm for computing solutions that are guaranteed to achieve lifetime within a factor of an optimal solution.

The model of distributed computation applied in the context of multihop wireless networks is a challenging one, especially if both the integrity of nodes and messages passed between them cannot be guaranteed. However, algorithms that operate in this model and prove themselves useful have the potential to be applied in other settings as well. As part of future work the author plans to consider extensions of the primal-dual framework to problems in other domains of distributed computing. It would also be interesting to consider generalizations of the sleep-scheduling problem, for example, in combination with data-aggregation techniques.

The methods and algorithms discussed here could be seen from a higher-

level perspective. It would be interesting to identify alternative, possibly more diverse, problem settings in which the same or related techniques can be applied.[1] In the future we may envision a method of compiling a high-level description of a network optimization problem into a low level distributed optimization algorithm. Although user interaction may be required in most cases, it would relieve the protocol designer from dealing with every algorithmic detail by giving the opportunity to focus on high-level design decisions.

[1]Possibly the author suffers from Maslow's hammer effect [92], although reviewing the literature and finding related techniques for diverse problems have convinced him otherwise.

BIBLIOGRAPHY

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

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

[3] K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[4] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

[5] P. C. B. Vatinlen, F. Chauvet and P. Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185:1390–1401, 2008.

[6] G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA '02)*, pages 101–113, London, UK, 2002. Springer-Verlag.

[7] G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42:231–248, 2005.

[8] S. Basagni, M. Mastrogiovanni, and C. Petrioli. A performance comparison of protocols for clustering and backbone formation in large scale ad hoc networks. In *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 70–79, 2004.

[9] A. Basu, A. Lin, and S. Ramanathan. Routing using potentials: A dynamic traffic-aware routing algorithm. In *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 37–48, New York, NY, USA, 2003. ACM Press.

[10] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 54–59. IEEE Computer Society, 1989.

[11] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, pages 2329–2334, March 2004.

[12] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

[13] D. P. Bertsekas, A. Nedi'c, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[14] D. P. Bertsekas and J. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Prentice Hall, 1989. Available for download at `http://dspace.mit.edu/handle/1721.1/3719`.

[15] M. Bhardwaj, S. Misra, and G. Xue. Distributed topology control in wireless ad hoc networks using β-skeletons. In *Workshop on High Performance Switching and Routing*, pages 371– 375, 2005.

[16] D. Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*, volume 53 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[17] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and manets. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 329–369. Kluwer Academic Publishers, 2005.

[18] S. Borbash and E. Jennings. Distributed topology control algorithm for multihop wireless networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, pages 355–360, 2002.

[19] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, 2004.

[20] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, 1998. Special Issue on Geometric Representations of Graphs.

[21] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, New York, NY, USA, 1998. ACM.

[22] R. Bruno, M. Conti, and E. Gregori. Mesh networks: commodity multihop ad hoc networks. *IEEE Communications Magazine*, 43(3):123–131, 2005.

[23] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic. Towards optimal sleep scheduling in sensor networks for rare-event detection. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 20–27, Piscataway, NJ, USA, 2005. IEEE Press.

[24] M. Caramia and A. Sgalambro. An exact approach for the maximum concurrent k-splittable flow problem. *Optimization Letters*, 2(2):251–265, 2007.

[25] A. Cerpa and D. Estrin. ASCENT: adaptive self-configuring sensor networks topologies. *IEEE Transactions on Mobile Computing*, 3(3):272–285, July–Aug 2004.

[26] J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 22–31, 2000.

[27] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wirel. Netw.*, 8(5):481–494, 2002.

[28] D. Chen and P. K. Varshney. QoS support in wireless sensor networks: A survey. In *Proceedings of the 2004 International Conference on Wireless Networks (ICWN 2004)*, 2004.

[29] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95:255–312, 2007.

[30] I. Chlamtac, M. Conti, and J. J. N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13–64, 2003.

[31] V. Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[32] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

[33] A. E. F. Clementi, P. Penna, and R. Silvestri. Hardness results for the power range assignment problem in packet radio networks. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (RANDOM-APPROX)*, pages 197–208, London, UK, 1999. Springer-Verlag.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[35] B. Cărbunar, A. Grama, J. Vitek, and O. Cărbunar. Redundancy and coverage detection in sensor networks. *ACM Trans. Sen. Netw.*, 2(1):94–128, 2006.

[36] D. Culler, D. Estrin, and M. Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37:41–49, 2004.

[37] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Heidelberg, 4th edition, 2010.

[38] I. Dietrich and F. Dressler. On the lifetime of wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(1):1–39, 2009.

[39] A. Ephremides. Energy concerns in wireless networks. *IEEE Wireless Communications*, 9(4):48–59, 2002.

[40] O. Escalante, T. Pérez, J. Solano, and I. Stojmenovic. RNG-based searching and broadcasting algorithms over Internet graphs and peer-to-peer computing systems. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pages 47–54, 2005.

[41] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[42] U. Feige, M. M. Halldórsson, and G. Kortsarz. Approximating the domatic number. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC'00)*, pages 134–143, New York, NY, USA, 2000. ACM.

[43] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discret. Math.*, 13(4):505–520, 2000.

[44] P. Floréen, P. Kaski, J. Kohonen, and P. Orponen. Lifetime maximization for multicasting in energy-constrained wireless networks. *IEEE Journal on Selected Areas in Communications*, 23(1):117–126, 2005.

[45] P. Floréen, P. Kaski, T. Musto, and J. Suomela. Local approximation algorithms for scheduling problems in sensor networks. In *Proceedings of the Third International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'07)*, volume 4837 of *Lecture Notes in Computer Science*, pages 99–113, 2008.

[46] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[47] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: Single-path routing vs. multi-path routing. In *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, March 2004.

[48] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

[49] L. Georgiadis. Bottleneck multicast trees in linear time. *IEEE Communications Letters*, 7(11):564–566, 2003.

[50] A. J. Goldsmith and S. B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications*, pages 8–27, 2002.

[51] F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM Journal on Computing*, 38(3):825–840, 2008.

[52] S. Guo, O. W. W. Yang, and V. C. M. Leung. Tree-based distributed multicast algorithms for directional communications and lifetime optimization in wireless ad hoc networks. *EURASIP Journal on Wireless Communications and Networking*, 2007:Article ID 98938, 10 pages, 2007.

[53] S. K. S. Gupta and P. K. Srimani. Self-stabilizing multicast protocols for ad hoc networks. *Journal of Parallel and Distributed Computing*, 63(1):87–96, 2003.

[54] H. Haanpää, A. Schumacher, and P. Orponen. Distributed algorithms for lifetime maximization in sensor networks via min-max spanning subgraphs. *Wireless Networks*, 16:875–887, 2010.

[55] H. Haanpää, A. Schumacher, T. Thaler, and P. Orponen. Distributed computation of maximum lifetime spanning subgraphs in sensor networks. In H. Zhang, S. Olariu, J. Cao, and D. Johnson, editors, *Proceedings of the 3rd International Conference on Mobile Ad-hoc and Sensor Networks (MSN '07)*, volume 4864 of *Lecture Notes in Computer Science*, pages 445–456, Berlin / Heidelberg, 2007. Springer-Verlag.

[56] S. Haldar. An 'all pairs shortest paths' distributed algorithm using $2n^2$ messages. *Journal of Algorithms*, 24(1):20–36, 1997.

[57] M. M. Halldórsson. Approximations of independent sets in graphs. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX '98)*, pages 1–13, London, UK, 1998. Springer-Verlag.

[58] J. He, M. Bresler, M. Chiang, and J. Rexford. Towards robust multilayer traffic engineering: Optimization of congestion control and routing. *IEEE Journal on Selected Areas in Communications*, 25:868–880, 2007.

[59] J. He, J. Rexford, and M. Chiang. Don't optimize existing protocols, design optimizable protocols. *SIGCOMM Comput. Commun. Rev.*, 37(3):53–58, 2007.

[60] M. Hempstead, M. J. Lyons, D. Brooks, and G.-Y. Wei. Survey of hardware systems for wireless sensor networks. *Journal of Low Power Electronics*, 4(1):11–20, 2008.

[61] X. Hou and D. Tipper. Impact of failures on routing in mobile ad hoc networks using DSR. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, Jan 2003.

[62] IETF. IETF mobile ad hoc network (MANET) working group charter, 2006.

[63] M. Ilyas, editor. *The Handbook of Ad Hoc Wireless Networks*. CRC Press, 2002.

[64] A. Iyer, C. Rosenberg, and A. Karnik. What is the right model for wireless channel interference? *IEEE Transactions on Wireless Communications*, 8(5):2662–2671, May 2009.

[65] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *J. ACM*, 50(6):795–824, 2003.

[66] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4):193–205, 2002.

[67] B. Johansson. *On Distributed Optimization in Networked Systems*. PhD thesis, Royal Institute of Technology (KTH), 2008.

[68] B. Johansson, M. Rabi, and M. Johansson. A simple peer-to-peer algorithm for distributed optimization in sensor networks. In *Proceedings of the IEEE Conference on Decision and Control*, 2007.

[69] B. Johansson, P. Soldati, and M. Johansson. Mathematical decomposition techniques for distributed cross-layer optimization of data networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1535–1547, Aug. 2006.

[70] D. B. Johnson, Y.-C. Hu, and D. A. Maltz. The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4. Technical report, IETF, 2007. RFC 4728.

[71] D. Julian, M. Chiang, D. O'Neill, and S. Boyd. QoS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In *Proceedings of IEEE INFOCOM*, pages 477–486, 2002.

[72] I. Kang and R. Poovendran. Maximizing network lifetime of broadcasting over wireless stationary ad hoc networks. *Mobile Networks and Applications*, 10(6):879–896, 2005.

[73] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[74] F. P. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998.

[75] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17(2):280 – 289, 1994.

[76] L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power consumption in packet radio networks. *Theoretical Computer Science*, 243(1-2):289–305, 2000.

[77] M. Kohvakka, J. Suhonen, M. Hannikainen, and T. D. Hamalainen. Transmission power based path loss metering for wireless sensor networks. In *17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5, 2006.

[78] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 575–578, 2002.

[79] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom '04)*, pages 260–274, New York, NY, USA, 2004. ACM.

[80] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad hoc networks beyond unit disk graphs. *Wirel. Netw.*, 14(5):715–729, 2008.

[81] S. Kumar, V. S. Raghavan, and J. Deng. Medium access control protocols for ad hoc wireless networks: A survey. *Ad Hoc Networks*, 4(3):326 – 358, 2006.

[82] I. Lestas and G. Vinnicombe. Combined control of routing and flow: a multipath routing approach. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 2390–2395, 2004.

[83] X.-Y. Li. Algorithmic, geometric and graphs issues in wireless networks. *Wireless Communications and Mobile Computing*, 3(2):119–140, 2003.

[84] X. Lin and N. B. Shroff. An optimization-based approach for QoS routing in high-bandwidth networks. *IEEE/ACM Trans. Netw.*, 14(6):1348–1361, 2006.

[85] X. Lin and N. B. Shroff. Utility maximization for communication networks with multipath routing. *IEEE Transactions on Automatic Control*, 51:766–781, 2006.

[86] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[87] E. L. Lloyd, R. Liu, M. V. Marathe, R. Ramanathan, and S. Ravi. Algorithmic aspects of topology control problems for ad hoc networks. *Mobile Networks and Applications*, 10(1-2):19–34, feb 2005.

[88] S. Low. Multipath optimization flow control. In *ICON '00: Proceedings of the 8th IEEE International Conference on Networks*, page 39, Washington, DC, USA, 2000. IEEE Computer Society.

[89] S. H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Netw.*, 11(4):525–536, 2003.

[90] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, CA, USA, 1996.

[91] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[92] A. H. Maslow. *The Psychology of Science : A Reconnaissance*. Harper and Row, 1966.

[93] S. McCanne, S. Floyd, K. Fall, and K. Varadhan. The network simulator NS2, 1995. The VINT project, available for download at http://www.isi.edu/nsnam/ns/.

[94] V. Mhatre and C. Rosenberg. Design guidelines for wireless sensor networks: communication, clustering and aggregation. *Ad Hoc Networks*, 2(1):45 – 63, 2004.

[95] T. Moscibroda. *Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks*. PhD thesis, ETH Zurich, 2006.

[96] T. Moscibroda and R. Wattenhofer. Maximizing the lifetime of dominating sets. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 242.2, 2005.

[97] MOSEK ApS. *The MOSEK optimization tools manual. Version 5.0.*, 2008. Available for download at `http://www.mosek.com`.

[98] S. Mueller, R. P. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. *Lecture Notes in Computer Science*, 2965:209–234, Jan 2004.

[99] S. Narayanaswamy, V. Kawadia, R. S. Sreenivas, and P. R. Kumar. Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the COMPOW protocol. In *Proceedings of the European Wireless Conference*, pages 156–162, 2002.

[100] A. Nasipuri, R. Castañeda, and S. R. Das. Performance of multipath routing for on-demand protocols in mobile ad hoc networks. *Mobile Networks and Applications*, 6(4):339–349, 2001.

[101] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4):1–17, 2008.

[102] ns 3 project. ns-3 tutorial, 2010. Available online at `http://www.nsnam.org/docs/tutorial.pdf`.

[103] C. A. Oliveira and P. M. Pardalos. Ad hoc networks: Optimization problems and solution methods. In M. X. Cheng, Y. Li, and D.-Z. Du, editors, *Combinatorial Optimization in Communication Networks*, pages 147–169. Springer US, 2006.

[104] R. Oliveira, L. Bernardo, and P. Pinto. The influence of broadcast traffic on IEEE 802.11 DCF networks. *Computer Communications*, 32(2):439 – 452, 2009.

[105] F. Paganini. Congestion control with adaptive multipath routing based on optimization. In *40th Annual Conference on Information Sciences and Systems*, pages 333–338, 2006.

[106] A. Panconesi and M. Sozio. Fast primal-dual distributed algorithms for scheduling and matching problems. *Distributed Computing*, 22(4):269–283, 2010.

[107] I. Papadimitriou and L. Georgiadis. Energy-aware broadcast trees in wireless networks. *Mobile Networks and Applications*, 9:567–581, 2004.

[108] J. Papandriopoulos, S. Dey, and J. Evans. Optimal and distributed protocols for cross-layer design of physical and transport layers in manets. *IEEE/ACM Trans. Netw.*, 16(6):1392–1405, 2008.

[109] M. R. Pearlman, Z. J. Haas, P. Sholander, and S. S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. In *Proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc '00)*, pages 3–10, Piscataway, NJ, USA, 2000. IEEE Press.

[110] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Philadelphia, PA, 2000.

[111] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. Technical report, IETF, July 2003. RFC 3561.

[112] S. Prasad, A. Schumacher, H. Haanpää, and P. Orponen. Balanced multipath source routing. In T. Vazão, M. Freire, and I. Chong, editors, *Information Networking. Towards Ubiquitous Networking and Services. Proceedings of the 21st International Conference on Information Networking (ICOIN '07). Revised Selected Papers*, volume 5200 of *Lecture Notes in Computer Science*, pages 315–324. Springer Berlin/Heidelberg, 2008.

[113] F. L. Presti. Joint congestion control: routing and media access control optimization via dual decomposition for ad hoc wireless networks. In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 298–306, New York, NY, USA, 2005. ACM Press.

[114] M. Rabbat and R. Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN '04)*, pages 20–27, New York, NY, USA, 2004. ACM.

[115] R. Raffard, C. Tomlin, and S. Boyd. Distributed optimization for cooperative agents: application to formation flight. In *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC'2004)*, pages 2453–2459, 2004.

[116] S. Rajagopalan and V. V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, 28(2):525–540, 1999.

[117] R. Ramanathan and R. Rosales-Hain. Topology control of multihop wireless networks using transmit power adjustment. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 404–413, 2000.

[118] V. Rodoplu and T. H. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1333–1344, sep 1999.

[119] M. Sanchez, P. Manzoni, and Z. J. Haas. Determination of critical transmission range in ad-hoc networks. In *Proceedings of Multiaccess, Mobility and Teletraffic for Wireless Communications Conference*, 1999.

[120] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37:164–194, 2005.

[121] N. Santoro. *Design and Analysis of Distributed Algorithms*. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2006.

[122] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*, 2006. 11 pages.

[123] A. Schumacher and H. Haanpää. Distributed network utility maximization in wireless networks with a bounded number of paths. In *Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM^{2}HW^{2}N 2008)*, pages 96–103, New York, NY, USA, 2008. ACM.

[124] A. Schumacher and H. Haanpää. Distributed sleep scheduling in wireless sensor networks via fractional domatic partitioning. In R. Guerraoui and F. Petit, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 5873/2009 of *Lecture Notes in Computer Science*, pages 640–654. Springer-Verlag Berlin / Heidelberg, 2009.

[125] A. Schumacher, H. Haanpää, S. E. Schaeffer, and P. Orponen. Load balancing by distributed optimisation in ad hoc networks. In J. Cao, I. Stojmenovic, X. Jia, and S. K. Das, editors, *Mobile Ad-hoc and Sensor Networks*, volume 4325/2006 of *Lecture Notes in Computer Science*, pages 873–884, Berlin / Heidelberg, 2006. Springer-Verlag.

[126] A. Schumacher, P. Orponen, T. Thaler, and H. Haanpää. Lifetime maximization in wireless sensor networks by distributed binary search. In R. Verdone, editor, *Proceedings of the 5th European Conference on Wireless Sensor Networks (EWSN 2008)*, volume 4913/2008 of *Lecture Notes in Computer Science*, pages 237–252, Berlin / Heidelberg, 2008. Springer-Verlag.

[127] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 188–200, New York, NY, USA, 2004. ACM.

[128] R. Sivakumar, B. Das, and V. Bharghavan. Spine routing in ad hoc networks. *Cluster Computing*, 1(2):237–248, 1998.

[129] P. Soldati, B. Johansson, and M. Johansson. Proportionally fair allocation of end-to-end bandwidth in STDMA wireless networks. In *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, pages 286–297, New York, NY, USA, 2006. ACM Press.

[130] M. Sozio. *Efficient Distributed Algorithms via the Primal-Dual Schema*. PhD thesis, Sapienza University, Rome, Italy, 2006.

[131] K. Srinivasan and P. Levis. RSSI is under-appreciated. In *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets)*, 2006. Available for download at `http://sing.stanford.edu/pubs/rssi-emnets06.pdf`.

[132] J. Suomela. Locality helps sleep scheduling. In *Working Notes of the Workshop on World-Sensor-Web: Mobile Device-Centric Sensory Networks and Applications*, 2006. Available for download at `http://www.sensorplanet.org/wsw2006/8_Suomela_WSW2006_final.pdf`.

[133] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.

[134] G. Tsaggouris and C. Zaroliagis. QoS-aware multicommodity flows and transportation planning. In R. Jacob and M. Müller-Hannemann, editors, *ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[135] C. Tschudin, P. Gunningberg, H. Lundgren, and E. Nordström. Lessons from experimental MANET research. *Ad Hoc Networks*, 3(2):221–233, 2005.

[136] J. N. Tsitsiklis and G. D. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. *J. ACM*, 42(2):382–400, 1995.

[137] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Department of Operations and Research and Financial Engineering, Princeton University, 2001.

[138] R. Vannier and I. G. Lassous. Towards a practical and fair rate allocation for multihop wireless networks based on a simple node model. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '08)*, pages 23–27, New York, NY, USA, 2008. ACM.

[139] V. V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[140] S. Vutukury and J. J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 227–238, New York, NY, USA, 1999. ACM Press.

[141] P.-J. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.

[142] J. Wang, L. Li, S. Low, and J. Doyle. Can shortest-path routing and TCP maximize utility. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '03)*, pages 2049–2056, 2003.

[143] L. Wang, L. Zhang, Y. Shu, and M. Dong. Multipath source routing in wireless ad hoc networks. In *Electrical and Computer Engineering, 2000 Canadian Conference on*, volume 1, pages 479–483 vol.1, 2000.

[144] W.-H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Perform. Eval.*, 52(2-3):119–132, 2003.

[145] Y. Wang, W. Wang, and X.-Y. Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '05)*, pages 2–13, New York, NY, USA, 2005. ACM.

[146] A. Warrier, S. Park, J. Min, and I. Rhee. How much energy saving does topology control offer for wireless sensor networks? - a practical study. *Computer Communications*, 30(14-15):2867 – 2879, 2007. Network Coverage and Routing Schemes for Wireless Sensor Networks.

[147] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1388–1397, April 2001.

[148] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 585–594, 2000.

[149] G. Wittenburg, K. Terfloth, F. L. Villafuerte, T. Naumowicz, H. Ritter, and J. Schiller. Fence monitoring: experimental evaluation of a use case for wireless sensor networks. In *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN '07)*, pages 163–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[150] K. Wu and J. Harms. Performance study of a multipath routing method for wireless mobile ad hoc networks. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, pages 99–107, Washington, DC, USA, 2001. IEEE Computer Society.

[151] L. Xiao, M. Johansson, and S. P. Boyd. Simultaneous routing and resource allocation via dual decomposition. *IEEE Transactions on Communications*, 52:1136–1144, July 2004.

[152] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.

[153] Y. Yi and M. Chiang. Stochastic network utility maximization – a tribute to kelly's paper published in this journal a decade ago. *European Transactions on Telecommunications*, 19(4):421–442, 2008.

[154] N. E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, pages 170–178, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[155] M. Youssef, A. Agrawala, and A. Udaya Shankar. Wlan location determination via clustering and probability distributions. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 143 – 150, 2003.

Index

ad hoc network, 4, 9–14
  critical transmission range, 94
  dynamic source routing, 13–14
  medium access control, 10
  network utility maximization, 73
  routing algorithms, 12
  transmission scheduling, 73
approximation algorithm, 21
approximation factor, 22

barrier coverage, 16
beaconing method, 105
binary search, 100
bottleneck cost, 95
branch-and-bound algorithm, 31, 33
broadcast model, *see* distributed algorithm, communication, broadcast model
broadcast transmission, 11

CBR source, 62
Chvátal's algorithm, 26, 53, 54, 115
clear-to-send message, 12
$\mathcal{CONGEST}$ model, 44, 120
congestion, *see* edge, congestion
convex
  function, 35–36
  program, 36
    complementary slackness, 39
    dual function, 37
    Lagrangian function, 37
    Slater's condition, 38
    weak duality, 38
  set, 35
  strictly concave, 35
  strictly convex, 35

distributed algorithm, 42
  communication
    broadcast model, 43, 100
    unicast model, 43, 93
  communication delay, 42
  convergecast, 49, 102
  execution time, 44
  leader election, 47
  message complexity, 43
  reactive model, 42
  register, 41
  termination, 43
  time complexity, 43
  timer, 41
  unique initiator, 42
distributed computing environment, 41
DMMT algorithm, 109–110
domatic number, *see* graph, domatic number
DSR algorithm, 12–14
dual decomposition, 37, 73, 79

edge
  congestion, 30, 58
  flow, 28, 30
exponential back-off, 11
exposed terminal problem, 10

flow-balance constraint, 33, 86

Garg-Könemann scheme, 28, 117
Gauss-Seidel method, 40, 78
graph, 5
  α-spanner, 7
  bottleneck spanning tree, 93
  bounded independence graph, 8, 53
  communication graph, 7, 41
  connected, 7
  diameter, 6
  disk graph, 8
  domatic number, 18, 116
  dominating set, 6, 53
  independent set, 7, 51
  interference graph, 51
  minimum dominating set, 6
  minimum spanning tree, 6, 47

minimum weight dominating set, 6, 26, 53, 121
minmax spanner, 7, 93, 95–99
neighborhood, 6
relative neighborhood graph, 9, 96, 99, 108
shortest paths, 47–49
shortest-path spanning tree, 47
spanning tree, 45–47
subgraph, 6
unit disk graph, 8

harmonic number, 27, 53
hidden terminal problem, 10

interface queue, 13
overflows, 64
interference range, 12

Lagrangian
duality, 36–39
function, 37
multiplier, 37, 79
lifetime maximization, *see* sensor network, lifetime maximization
linear programming
canonical form, 23
covering LP, 23
dual fitting, 27
dual problem, 23
duality theorem, 23
integrality gap, 25
linear relaxation, 24, 25, 31
packing LP, 23
primal-dual algorithm, 25, 121
weighted set-cover problem, 26
load balancing, 57
local algorithm, 44, 121

MAC layer, 10–12
maximum concurrent flow problem, 30
maximum multicommodity flow problem, 27
maximum source-route length, 13, 61, 70, 72
medium access control, *see* MAC layer
mesh network, 4
minimum maximum congestion problem, 30, 57–58
minimum weight dominating set, *see* graph, minimum weight dominating set
minimum weight set cover problem, 25–27, 53, 117, 122
minmax spanner, *see* graph, minmax spanner
multicast lifetime maximization problem, 93
multihop routing, 4
multihop wireless network, 4
multipath routing, 57–58

n-of-*n* lifetime, 16, 18, 91, 93
network
coverage, 16
flow, *see* maximum multicommodity flow problem
optimization, 1
synchronizer, 55–56
utility maximization, 73–74
node
cover state, 122
level, 51
neighbor list, 125
neighbors, 6, 7
price, 123
update interval, 128, 131
span, 54, 125
supporters, 123
tree neighbors, 6, 45, 117
NS2, 18–20, 57, 61, 105, 108, 130

parallel algorithm
computation model, 121
path
flow, 28, 60
path loss
exponent, 15
model, 16, 92, 95, 105–107, 110
Prim's algorithm, 94, 109
projected gradient method, 39, 80
propagation delay, 107
propagation model, *see* path loss, model
proportional fairness, 75
proximal optimization, 40

range assignment problem, 15, 94
reactive model, *see* distributed algorithm, reactive model

redundancy graph, *see* sensor network, redundancy graph
relay region, 94
request-to-send message, 12
RNG, *see* graph, relative neighborhood graph
RSSI, 107

sensor network, 4–5, 14–18
  data gathering, 5, 91, 93
  forest-fire detection, 92
  lifetime maximization, 16, 93–94
  redundancy graph, 17, 115
  sleep scheduling, 17, 115
  topology control, 15, 91
set cover
  greedy algorithm, *see* Chvátal's algorithm
set cover problem, *see* minimum weight set cover problem
SINR, 12
sleep scheduling, *see* sensor network, sleep scheduling
source-routing protocol, 13
spatial reuse, 15
splittable flow, 31
strictly concave, *see* convex, strictly concave
strictly convex, *see* convex, strictly convex

target coverage, 16
TCP, 1, 73
threshold signal strength, 105
topology control, *see* sensor network, topology control
two-ray ground model, 109

unicast model, *see* distributed algorithm, communication, unicast model
unicast transmission, 11

wireless broadcast advantage, 12, 43, 121, 124
wireless transmission medium, 10

TKK DISSERTATIONS IN INFORMATION AND COMPUTER SCIENCE

ISBN 978-952-60-3480-5 (Print)
ISBN 978-952-60-3481-2 (Online)
ISSN 1797-5050 (Print)
ISSN 1797-5069 (Online)