Topology and Routing in Overlay Networks

by

Kishore Kothapalli

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

June, 2006

Abstract

In this age of information, new models of information exchange methodologies based on overlay networks are gaining popular attention. Overlay networks provide a logical interconnection topology over an existing physical network. Overlay networks offer benefits such as ease of implementation, flexibility, adaptability, and incremental deployability. Due to the wide range of applications and advantages, formal study of overlay networks is required to understand the various research challenges in this context.

In this thesis, we study two classes of overlay networks namely peer-to-peer networks and wireless ad hoc networks. Our focus will be along two central issues in overlay networks: how to arrive at efficient topologies and how to provide efficient routing strategies.

Peer-to-peer networks have gained a lot of research attention in recent years for various reasons. Despite many advances however, fundamental questions such as how to design deterministic constructions, and how to organize peers of non-uniform bandwidth have remained open. In this thesis, we answer these questions by providing a deterministic overlay topology, *Pagoda*, that can be used for efficient routing, data management and multicasting. Given the difficulty of arriving at good deterministic topologies in a purely decentralized manner, we also propose a unified methodology to create a large class of

overlay topologies via an approach called the *supervised overlay networks*. We show that this approach also has other advantages such as support for rapid peer join/leave and rapid repair.

For the case of wireless ad hoc networks, we start by providing a model for wireless communication that is much more realistic than the models that are being used in the theoretical community. Using this model, we show how to arrive a sparse spanner construction based on dominating sets. We then use the spanner construction to provide efficient algorithms for broadcasting and information gathering in wireless ad hoc networks. All our algorithms are simple, self-stabilizing and require only a constant amount of storage at any node. Thus, our algorithms are also applicable in a wide variety of scenarios such as simple sensor devices.

Advisor: Professor Christian Scheideler

Readers: Professor Rao Kosaraju and Professor Andreas Terzis

Dedicated to

the memory of my mother

Acknowledgements

First and foremost I express my gratitude to my advisor Dr. Christian Scheideler for supporting me, and sharing many of his insights. His clarity of thought and expression, timely and sound advice have been of immense help and a huge inspiration.

Thanks are also due to the members of my thesis committee, Prof. Rao Kosaraju, Prof. James Fill, Prof. Jin Kung, and Prof. Andreas Terzis for their valuable feedback.

I wish to take this opportunity to thank my teachers, Prof. Rao Kosaraju, Prof. James Fill, Prof. Sanjeev Saxena, and many others from whom I have benefited immensely during the course of my education.

It was a rewarding experience to work with Prof. Andrea Richa, Prof. Christian Schindelhauer, Ankur Bhargava, Chris Riley, Mark Thober, and Melih Onus. I wish to thank Prof. Hager for his advice while working towards a qualifier project.

I was lucky to have made some good friends at the Johns Hopkins University Ankur Kapoor, Paritosh Shroff, Sandeep Sarat, Debraj Ghosh, and many others. For all the good times, thank you all. Thanks also to friends from my earlier days at Warangal and Kanpur, especially Kiran Tati, Sriram Gorti, Sreekanth Bharatham, and Subbarao Denduluri.

Last but not the least, thanks are also due to my family members whose constant support and encouragement could always be counted upon even under difficult circumstances.

Contents

List of Figures

Chapter 1

Introduction

As the age of information has dawned upon us, it has become imperative that efficient information exchange methodologies be studied. While traditional network models certainly broadened the knowledge and understanding of information exchange, new and emerging paradigms require a different approach. Overlay networks, which are logical networks over an existing network, are becoming more common. Overlay networks supporting a range of functionality such as grid computing, file sharing, sensor networks, and wireless ad hoc networks are being studied heavily. Evidenced by the success of early applications using overlay networks such as Gnutella [50], and distributed.net [33] the research community has been quick to react and develop a vast array of applications, tools, and techniques to study problems in the area of overlay networks. Figure 1.1 shows an overlay network of six nodes with the bold edges representing the connections in the *logical* network.

Before we proceed further, it is important to understand the basic ideas behind various computing models so that one can appreciate the contribution of overlay networks. Below we first provide a concise review of known models of computing and why new models are gaining attention.

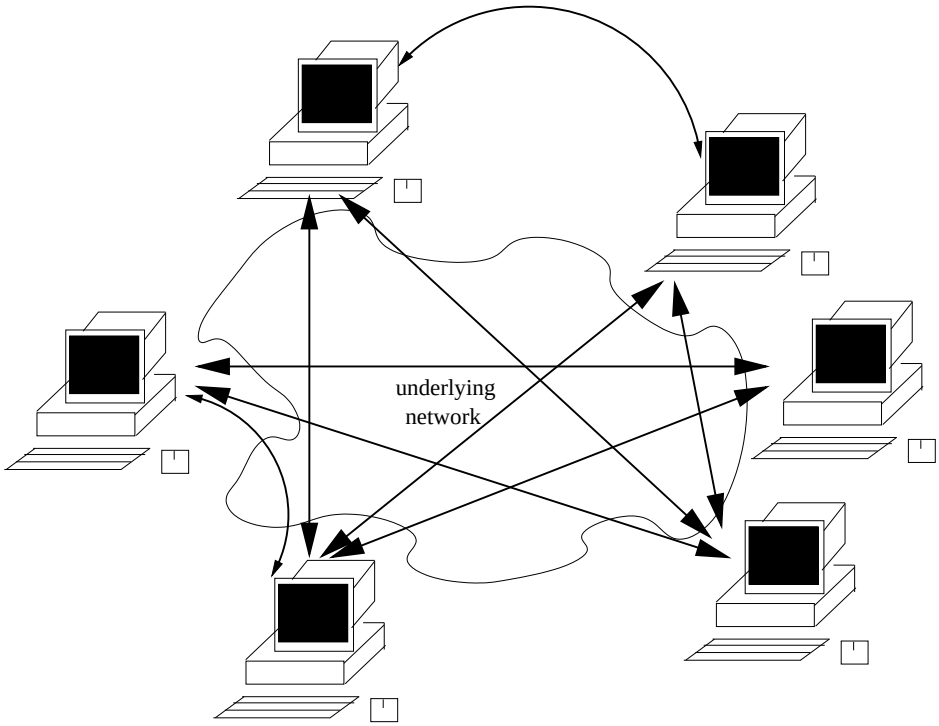


Figure 1.1: A logical (overlay) network.

1.1 Models of Computing

1.1.1 Desktop computing

During the early days of personal computers (PCs), the desktop was seen as the central computing tool. All the applications required by the user are provided in the desktop and when new applications are needed they have to be installed on his/her computer. Clearly, this model of computing becomes expensive and infeasible as the number of applications needed by the user grows. More importantly, this model does not allow any resource sharing between the users. These disadvantages meant that new models had to be designed.

1.1.2 Client-server computing

Client-server computing is a distributed model where two entities, the client and the server, communicate with each other according to some established protocol to perform certain tasks. Examples include (browser, web-server) where using the HTTP protocol the browser sends requests to a web-server and later displays the results, the X Window System (commonly known as X11) where typically a user's local display acts a server, and the like. Figure 1.2(a) shows an example of a client-server computing system.

While this model has better resource utilization compared to desktop computing, the clients are not left with too much of freedom. In most cases, these systems do not allow any interactions between the clients. Moreover, in this model the server might be overburdened as it has to serve multiple clients. Though there exist solutions to deal with such problems, these require providing special purpose costly hardware. Other problems such as a single point-of-failure at the server also exist. What is needed is a model which allows resource sharing and also cost sharing.

1.1.3 Peer-to-peer computing

The recent trend has been towards a model of computing which allows efficient sharing of resources. Also, there is a need to move away from client-server based computing and allow the clients to make some application-level decisions which they are best capable of. This is where the peer-to-peer model of computing enters the picture. To attempt a definition of peer-to-peer, Oram et. al [114] defines peer-to-peer broadly as follows:

> *A peer-too-peer system is a self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services.*

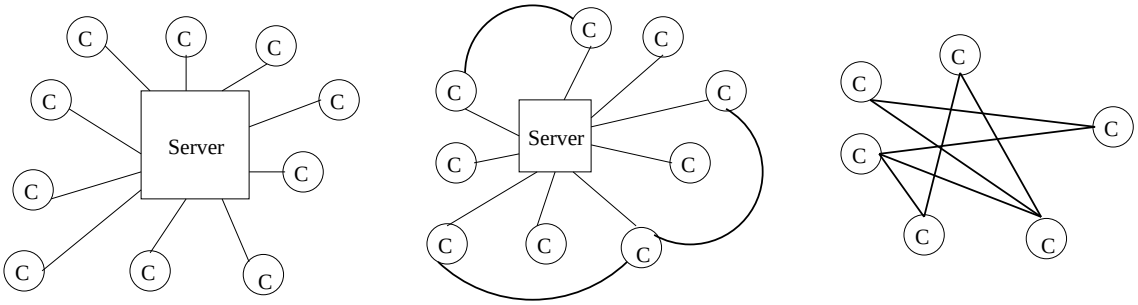


Figure 1.2: Figure (a) shows a client-server model of computing where the server handles all the requests of the clients. Figure (b) shows a supervised peer-to-peer system where the server has certain limited functionality and clients (peers) are allowed to communicate with each other. The bold lines indicate the client-client communication links. Figure (c) shows a pure peer-to-peer system where there is no central server. The figure is based on Figure 2.1 from [140].

As is common in literature, we do not distinguish between the terms peer-to-peer computing and peer-to-peer systems/networks and use them interchangeably. One can classify these further as *supervised* peer-to-peer systems and *pure* peer-to-peer systems. In supervised systems, there is a limited degree of centralization that drives the operation of the system whereas pure peer-to-peer systems are entirely decentralized. Figure 1.2(a–b) show an example of a supervised and pure peer-to-peer system. File sharing application such as Napster, grid computing projects such as distributed.net [33] are examples of supervised peer-to-peer systems, as both these systems involve certain degree of centralization. Later generations of peer-to-peer systems such as Chord [142] are examples of pure peer-to-peer systems. These provide an efficient sharing of resources and cost among the various participants. We shall have more to say on why these systems are popular and the reasons that make them exciting in Section 1.2 by studying a superclass of peer-to-peer systems namely overlay networks.

This thesis deals with not just peer-to-peer networks but overlay networks in general. When we speak about overlay networks in this thesis, some of the remarks are equally applicable to peer-to-peer networks and in some cases we specifically make the class distinction clear. In the next

section, we provide more reasons why overlay networks are appealing.

1.2 Why Logical Networks?

In this section, we state the reasons that make overlay networks suitable for many application scenarios. Some of the benefits of using logical networks are that they provide flexibility, ease of implementation, easy customizability and adaptability, and incremental deployability. These advantages make logical networks a good choice for a lot of applications. To provide further justification, we look at examples such as provisioning special features, Virtual Private Networks (VPN's), and grid computing, that benefit from the above features. In the following discussion we view the Internet as the underlying network unless explicitly mentioned.

1.2.1 Provisioning Special Features

For many applications, designing logical networks has several advantages compared to relying on the underlying network. A logical network provides a certain degree of flexibility and ease of implementation that is not achievable relying on the underlying network. Consider providing Quality-of-Service (QoS) guarantees to Internet traffic which may be demanded by certain applications such as multimedia, or real time industrial applications. In the current Internet, there is no standard way to pass QoS information across routers. Also, intrinsically any solution to guaranteeing service quality would be a case of weak-link phenomenon where the quality guaranteed will be as weak as the guarantee of the worst link in a path. Moreover, various applications have different QoS requirements which make it difficult to capture in any single solution. Thus, there are serious obstacles to providing end-to-end QoS guarantees. Whether to let the underlying network, the Internet, to allow applications to demand QoS guarantees or to have the end-hosts deal

with QoS guarantees is a hotly debated topic in Internet research forums such as the IETF (Internet Engineering Task Force). In this scenario, logical networks offer a solution as proposed in [96]. For example, sites requiring certain guarantees can form a logical network to sustain those guarantees without requiring any changes in the underlying network which might be prohibitively difficult for technical or economic reasons.

IPv6 is a classic example of the difficulties involved in changing the underlying network operation. IPv6 (IP Version 6) is the new generation Internet protocol that is designed to address the limitations such as a small address space, lack of uniform QoS capabilities, and to increase efficiency and flexibility in the current version of the protocol IPv4. The deployment of IPv6 has encountered huge delays as it involves development and deployment of new software on devices that are connected to the network, and upgrading millions of routers on the Internet to use IPv6 instead of IPv4.

When using logical networks such special protocols, or protocols implementing special features that depend on application specific knowledge, can be implemented without in any way burdening the underlying network. This approach also gives an additional ease of maintenance as updates or fixes to the protocols can be carried over with less effort.

1.2.2 Virtual Private Networks (VPNs)

Logical networks can also be used to augment the functionality provided by the underlying network to support additional features such as authentication, anonymity, and security. Consider the scenario where a company has offices at several geographically dispersed locations and wants to offer interconnectivity between these various locations. While using a public network such as the Internet would solve the problem it might introduce security risks which are potentially damaging to the company. Another solution is to use separate leased lines to interconnect the various locations.

But this becomes costly as the lines are billed not only based on usage but also based on fixed monthly fees. Even otherwise, having leased lines to interconnect does not solve the problem in its entirety. Consider the scenario where a traveling employee wishes to access the office private network while having access to only a public network. It is not easy unless the employee is based at one of the company locations.

The common solution these days to these problems is to provide a VPN. A VPN is a private network created on top of a public network such as the Internet with features such as security, service guarantees, reliability, and privacy [134]. The name "virtual" comes due to the fact that the private network is simulated on top of a public network, such as the Internet, using temporary, logical connections that have no physical presence. Unlike leased lines, the cost is based on usage time rather than fixed costs.

1.2.3 Grid Computing

Logical networks also allow efficient sharing of resources such as storage, and processing power that may otherwise sit idle on individual hosts. Consider for example, the grid computing system distributed.net [33] which was introduced around 1998. Individual users can download software from distributed.net which runs on the individual hosts when the hosts are idle. Upon processing the the current work unit, the software reports the results back to a server at distributed.net and downloads a new work unit. Alike distributed.net there are now several distributed computing projects for applications from areas such as genetics (see http://boinc.bakerlab.org/rosetta/), climate prediction modeling (see http://climateapps2.oucs.ox.ac.uk/cpdnboinc/), medicinal applications (see http://www.d2ol.com/), and for detecting signals of intelligent life outside the Earth (see http://setiathome.com).

The success of projects such as distributed.net can be gauged by looking at some of their re-

cent breakthroughs. In 2002, after working for 50 months, a 300,000 user base had tested about 15×10^{18} keys to solve the RC5-64 bit secret key challenge. The RC5-64 challenge is one of a series of contests held to understand the difficulty of finding a symmetric encryption key by exhaustive search. The computational power utilized for the RC5-64 project alone was estimated to be the equivalent of nearly half a million Pentium PCs. While still not being entirely decentralized, these projects show a way of amassing the computational power equivalent to that of modern day supercomputers at a fraction of the cost. Recent results [123] show how to achieve a greater degree of decentralization.

1.2.4 Internet Transparency and Symmetry

Overlay networks also are said to have the potential to "return the Internet to its founding principles" according to [114] by restoring its transparency and symmetric operation. This statement needs some justification.

In the early years of the Internet, hosts acted as peers sharing equal responsibilities. But with the rapid growth of the Internet around 1994 [1] [29] it has met with new challenges and also underwent a shift in the way the hosts behave.

The rise in the number of hosts on the Internet gave rise to challenges such as scaling up the address space, scaling up the Domain Name System (DNS), scaling of capacity, and scaling of protocols and algorithms. It is widely estimated that the 32-bit address space currently used in the Internet would run of addresses in a few years time [113]. To alleviate the address space exhaustion still using the IPv4 protocols, solutions such as Network Address Translator (NAT) devices, Dynamic Host Configuration Protocol (DHCP) are being used. NAT devices sit between a private network and a network connected directly to the public Internet. NATs enable a set of

[1] The number of Internet hosts in 1994 is estimated to be 2 million which reached 72 million by 2000 and is estimated to be 394 million in 2005.

hosts in a private network to share a small set of globally unique addresses. Hosts using DHCP are assigned a unique address when they are connected to the Internet and the address is reclaimed when the hosts is no longer connected to the Internet.

These technologies resulted in the Internet losing its original transparency. When NAT is deployed hosts are no longer uniquely addressable in the Internet which is an important design attribute of the original Internet. Moreover, NAT deployment introduces two addresses for a host - a local address that it knows and a global address that it is known by in the Internet. Similarly when using DHCP, applications cannot rely on IP addresses to uniquely distinguish hosts as the IP address may be in use by different hosts at various points of time. But new applications based on the peer-to-peer paradigm challenge this lack of transparency. In fact, many applications have found ways to work around the problems introduced by NATs, DHCP and firewalls. As the popularity of the new paradigm grows [2] it is imperative that these technologies be updated. Some scenarios studying the problems posed by the current lack of end-to-end transparency are presented in RFC 2775 [3].

With the expanding civilian usage of the Internet, many hosts on the Internet have largely become consumers of information with only a few hosts serving the information. This asymmetry has in fact meant that Internet Service Providers' (ISPs) have built their systems and practices around the idea that most of the end users spend most of their time downloading data from a few central servers. In fact, ISPs using cable technologies and ADSL (Asymmetric Digital Subscriber Line) provide lower upstream bandwidth than download bandwidth and this is fine as long as the users do not upload too much data. With the emergence of peer-to-peer networks it is hoped that this asymmetry in the current Internet, where a majority of the hosts are only consumers of information,

[2] It was reported in a study http://www.sandvine.com/solutions/ p2p_policy_mngmt.asp, that up to 50% of the Internet traffic is due to file sharing applications.

[3] See http://www.rfc-archive.org/getrfc.php?rfc=2775

can be reduced. As applications such as file-sharing, and publish-subscribe boards become popular, hosts can also become content-providers rather than being passive consumers of information.

Indeed the emergence of peer-to-peer applications that blur the distinction between providers and consumers of information has already started to show certain side effects. For example, in the days of the Napster, an ISP company in San Diego notified its users to stop running the Napster application as Napster is consuming too much of bandwidth [4].

1.3 Overlay Networks - A Brief History

Due to their growing importance as outlined above, the study of overlay networks is being treated as an independent area of research since the last decade. In this thesis, we focus on two classes of overlay networks namely, peer-to-peer overlays and overlays for wireless ad hoc networks. We now provide a brief introduction to these two classes of overlay networks.

1.3.1 Peer-to-Peer Networks

Peer-to-peer overlay networks have attracted a lot of research attention in the past few years due to the enormous advantages offered by them. Peer-to-peer (P2P) networks allow improve the efficacy of resources such as computation and storage by seamless sharing of resources. Also, the fact that peer-to-peer systems do not need a central server means that individuals can search for information or cooperate without fees or an investment in additional high-performance hardware.

Peer-to-peer Systems in the Internet

While the term “peer-to-peer” has recent denomination, ever since the emergence of the Internet many applications that are implicitly guided by the principles of peer-to-peer networks are

[4] See http://wired.com/news/technology/0,1282,35523,00.html for this news article.

known. One example is the File Transfer Protocol (FTP) which can be used to transfer files between hosts in the Internet. Each host can act as a peer that hosts certain files and other peers can establish a connection to initiate file transfer. Each host can act as either a server of information or a client of information depending on the context.

Other examples include the Usenet, and the Internet BGP routing scheme. Usenet can be thought of as a publish-subscribe service where users can post and read messages under different topics. Usenet originally relied on UUCP (Unix-to-Unix-Copy Protocol) which provides mechanism for a Unix machine to establish a connection to another Unix machine, exchange files, and terminate the connection. Currently, Usenet uses the Network News Transfer Protocol (NNTP) for exchanging the messages. Usenet has no centralized authority that creates or deletes the topics. The Border Gateway Protocol (BGP) is the routing protocol used to exchange routing information across the Internet. In BGP, the routers have a peer relationship between themselves and send periodic route updates amongst themselves.

Recent P2P Networks

Following the evolution of the peer-to-peer networks, the authors in [145] have categorized them into 3 generations. The first generation of peer-to-peer systems are pioneered by the file sharing application Napster. Napster had a centralized directory of files and their owners but once a owner of a file is found the download can happen without the involvement of the server. This of course has several disadvantages as the central server becomes a bottleneck for a system point of view. Also, Napster itself ran into legal battles over copyright issues and was shutdown after a protracted court battle. Gnutella [50] is also categorized as first generation P2P system and has a similar functionality as that of Napster but without any centralized directory. The authors of [145] cite the ease of deployment as the reason for this categorization and these early networks do not

have provably low lookup time which is important for file sharing applications.

Gnutella places files, *or objects*, at random locations and hence has to use naive flooding based methods to locate objects. In the second-generation P2P systems, this was addressed by placing objects at specified locations so that locating objects can be done faster. Systems that are placed in this category include, e.g., Tapestry [155], Pastry [129] (both use a scheme similar to that of Plaxton-Rajaraman-Richa [121]), Chord [142] based on consistent hashing [69] and, CAN [125] based on hierarchical decomposition. Most of the systems in the second generation category are based on structured overlays where the nodes in the network are mapped into a virtual address space and each node is given a label from this address space. The label of a node also dictates its neighbors in the logical network by using mathematical formulations. The labels are given in a manner that the logical network has certain structured topology such as the hypercube, de Bruijn, butterfly [5]. This allows one to show that lookup time, query path length, peer join/leave time are logarithmic (or poly-logarithmic) in the current size of the network.

These second generation systems can be used as a "Distributed Hash Table" (DHT) which takes the following form. A set of data items from an ordered space are to be mapped to a set of storage units so that the fraction of the data items at any unit is close to the best possible, i.e., all units store an equal proportion of the total data, while supporting operations such as `lookup` and `put`. Structured P2P overlays acting as DHT's are also proposed as a solution for a future generation Internet DNS [123].

Concepts for third generation systems addressing the weakness of the second generation systems include fault-tolerance, security, anonymity, robustness, providing incentives for cooperation, and the like. Some proposals that are provably fault-tolerant under various attack models are [42, 6, 132].

[5] A formal definition of these network topologies is provided in Chapter 2.

For a more detailed introduction to the above systems and a comparison, we refer the reader to [101, 145]. P2P networks can also be used as routing overlays to enhance certain network functionality such as security and authentication by provisioning a virtual private network. The local-control nature of the P2P systems means that single point of failures, generally associated with traditional client-server systems, can also be mitigated.

Peer-to-peer networks have also found applications in diverse areas such as grid computing, online gaming, databases, web-caching, information retrieval, web crawling, and in many such related and emerging areas. The rapid growth of interest in peer-to-peer networks can be judged by the fact that every year there are several conferences catering specifically to topics in peer-to-peer networks.

1.3.2 Wireless Ad Hoc Networks

A wireless ad hoc network comprises of a set of nodes that can communicate over a wireless medium. Initial applications of wireless networks are found in the military domain. A classic example is that of war fighters equipped with wireless devices giving them access to information such as the terrain, location, and strategic documents. However, the recent advent of numerous electronic devices that are capable of communicating over a wireless medium has meant that networks composed of wireless devices are becoming more common also in the civilian and the commercial domain. Examples include campus wide wireless LAN's, home networking, and wireless hot-spots. Recently, some cities such as Philadelphia have also initiated efforts to provide a city wide wireless network that every citizen can use to access the Internet [6]. Wireless networks consisting of devices that cooperate with each other, without the presence of any base station, to forward packets to each other are becoming feasible and widespread. For instance, sensor networks comprising millions of

[6] See http://www.phila.gov/wireless

tiny low-cost, low-power, wireless sensor devices are being used in a wide variety of applications such as disaster recovery, warehouse management, and remote surveillance. In the case of sensor networks, when used for gathering information, there is normally a single observer where all the messages are delivered to, but this observer does not normally provide the functionality of a base station.

One of the important problems for wireless ad hoc networks is to organize the wireless devices, *nodes*, into a logical network that acts as a backbone for communication. The quality of the overlay network, the backbone structure, can be judged based on several criteria such as connectivity, energy-efficiency, and adaptability to mobile hosts.

Proposals or approaches to arrive at overlay networks in wireless ad hoc networks can be classified into 3 generations as follows. The first generation approaches use one or more base stations (or access points) and the wireless nodes always try to maintain contact with at least one base station. The cell-based approaches, and wireless hot-spots fit this approach. The presence of centralized infrastructure in the form of base stations certainly simplifies the problem and the overlay topology obtained is a star topology with the access point at the center of the star as normally each node is connected to a single access point.

The second generation systems are characterized by having a multi-hop approach. In this approach, not all nodes may be communicating directly using a centralized base station. Nodes can communicate by using other nodes in the ad hoc network as relays to reach the base station. More precisely, wireless stations that can reach a base station directly communicate directly with the base station. Other wireless nodes communicate with a base station using multiple hops to forward their traffic. Proposals for arriving at overlay network or communication protocols in wireless ad hoc networks in this scenario include those put forward by the IETF Manet working group, and the

Archipelago project [1, 7].

The third generation approaches aim at constructing overlay networks for wireless ad hoc networks in the absence of any centralized infrastructure. In this scenario, the ad hoc network can be seen as a peer-to-peer network formed by a set of wireless stations, which organize themselves into a temporary network. The lack of any centralized infrastructure and mobility of nodes pose heavy challenges for designing overlay networks for wireless ad hoc networks. Some of the recent proposals include e.g., [36, 84, 5, 47]. There are two approaches to arriving at overlay networks for wireless ad hoc networks, namely, *topological overlays* and *geometric overlays*. In the following, we briefly describe each of these approaches.

In topological overlays, the overlay network is constructed by choosing cluster heads and gateway edges that interconnect some of the cluster heads. Each node is either a cluster head or a member of some clusters. Gateway edges allow communication between clusters. Geometric overlay networks use the relative position of the nodes in the network in arriving at the construction. Several constructions based on geometric position of the nodes such as the Gabriel graph [46], the Yao graph [152], Voronoi diagrams are studied for their ease of implementation. The geometric position of the nodes dictates the structure and hence the quality of the resulting overlay network. For example, when using the Gabriel graph or the Yao graph, it is possible to create situations where the degree of some node in the overlay network is $\Theta(n)$. Variants of these graphs, such as the symmetric Yao graph, the Relative Neighborhood Graph (RNG) [47] are also studied. Some of the above constructions arrive at a planar overlay topology, which is known to be useful in the context of unicasting algorithms such as face routing [86] and its many variants [85, 47, 17, 18, 19].

Higher order communication primitives such as broadcasting, gossiping, unicasting are then supported on top of the overlay network. For example, there is a class of routing protocols called

the *geometric routing* protocols [86] which use a planar overlay network to perform route discovery.

1.4 Research Challenges

Till now we have aimed to provide an introduction to the area of overlay networks. We now aim to understand some of the research challenges that arise in the area of overlay networks.

This thesis deals mainly with two classes of overlay networks: peer-to-peer overlays and overlays for wireless ad hoc networks. When we look at these two types of networks, to design overlay networks one encounters the following challenges. In this section, by the term *overlay network*, we mean peer-to-peer overlay networks or overlays for wireless ad hoc networks.

Recall the early P2P system Napster which introduced the concept of peer-to-peer file sharing. Napster had a central server that is used to store a directory of files and where they are available so that once a user having a particular file is located, the content can be served independent of the central server. But storing the directory at a central server meant that all lookup operations had to go through the central server creating a potential bottleneck and also making it a central point of failure. While Gnutella [50] did away with central indexing, it uses flooding based techniques to query for content. As the number of participants in Gnutella increases, the load on each peer grows proportionately as a result. Such a solution is not easily scalable. Thus, designing overlay networks having desirable properties deserves serious thought.

The topology of the overlay network specifies how the participating entities (peers) can communicate with each other in the overlay network. As overlay networks are being deployed or proposed for a variety of applications, one primary requirement of the overlay network is that the topology should allow for efficient operation. Consider client-server topologies that are known since a long time where there is a central server entity that acts on the requests of the clients. This

can be seen as forming a star topology with the server at the center. This approach hardly meets the efficiency criteria as such a topology does not allow for interaction between the various clients without involving the server thereby overburdening the server.

One can certainly interconnect all the peers resulting in a clique topology. This solution certainly improves the efficiency as all the peers are interconnected and can exchange information quickly. But this approach does not scale well even for a network of relatively moderate size. Overlay networks, however, should be able to operate at a much higher scale. For example, Gnutella [50] on a typical day has reported 2 million users. Operating at such an enormous scale requires that the topology of the network has to be designed carefully to achieve scalability.

Also, the topology should be robust enough so that it can function under difficult or adverse circumstances. While the client-server topology can be made robust by providing special purpose hardware, the topology itself would still be inefficient for many applications.

Thus, devising strategies to satisfy the afore-mentioned criteria is an extremely challenging problem. While efficiency, scalability and robustness are sought after, overlay networks have other equally important challenges. Another important difference in these classes of overlay networks is that the peers are dynamic in nature and hence overlay networks should allow for the participants to join or leave the network and at a rapid rate. To quote statistical observations for Kazaa [72], it is reported in [54] that 50% of the users in Kazaa have a session time of the order of minutes. This requires that the network should be able to efficiently process a join or leave operation, without any centralized control.

Moreover, overlay networks typically are composed of entities that differ significantly in their characteristics, i.e., the participants may introduce heterogeneity in the network. For example, nodes in a peer-to-peer network differ significantly in the amount of available bandwidth or the bandwidth

they can contribute to the P2P network. This requires that the P2P network be flexible enough so as to accommodate nodes of varying bandwidth. Also, future generation P2P systems should allow the users to control or limit the amount of bandwidth they contribute to a particular application as each user may be running several P2P applications together. This means that suitable topologies for overlay networks that operate efficiently in an heterogeneous environment have to be designed.

Further, when considering overlay networks for wireless nodes, for example, resources such as power is highly expensive. In some cases such as wireless sensor networks, it may in fact be difficult if not impossible to recharge the sensors once they are deployed. Hence the overlay network should ideally support mechanisms to minimize the usage of such expensive resources so as to increase the lifetime and availability of the individual devices.

Thus, there are a lot of challenges that one has to take into account when designing overlay networks. The initial works such as Napster, and distributed.net have showed the remarkable power of peer-to-peer systems. Based on these successes, the academic community has reacted quickly to bring this line of work into the research mainstream so as to set them on a formal footing where they can be studied rigorously. Over the last decade, research in overlay networks has produced a vast amount of literature leading to various insights, techniques, and solutions.

In this thesis, we undertake a formal study of overlay networks to address the above challenges with focus on peer-to-peer networks and wireless ad hoc networks. We address how to design efficient topologies for overlay networks and how to provide efficient routing strategies for various routing problems that arise in the context of overlay networks. For example, we show how to design a peer-to-peer network that can operate efficiently in an heterogeneous environment which solves an open problem in that area. Similarly, we show how to provide sparse backbone structures for wireless ad hoc networks that can then be used to perform broadcasting and information gathering

efficiently. (For a summary of contributions made in this thesis and their technical significance we refer the reader to Chapter 3).

1.5 Relation to other areas

In this section we discuss briefly other recent and emerging research areas in Computer Science that have some relation to overlay networks. We look at examples such as (traditional) distributed systems, and content distribution networks.

1.5.1 Distributed Systems

Distributed systems with no global memory involve a set of computing entities interconnected via a certain topology and computation is done by the entities exchanging information through messages. As communication is treated as an expensive resource, one of the goals in distributed computing is to use as little communication as possible. There appears to be a lot of commonality in the solution techniques employed in distributed systems and overlay networks. In fact, the idea of self-stabilization [32] has its roots in distributed systems. Also, the theoretical limitations of computation in distributed systems carry over to overlay networks also. But certain important differences exist.

In traditional distributed systems, while the entities are treated as being autonomous, in most cases they are homogeneous in nature. We have seen that on the other hand overlay networks tend to be rather heterogeneous. Also, distributed systems mostly are not dynamic in nature and in many cases do not have to deal with issues such as power consumption. These, and other differences, make it important to treat the study of overlay networks as separate from that of distributed systems.

Models with shared memory, for example the parallel computing models, were also studied

during the previous decades. In this model, there is a set of processors that have a globally accessible shared memory. Based on the model of memory access, several variations exist such as the Concurrent Read Exclusive Write (CREW) and the weaker Concurrent Read Concurrent Write (CRCW), and the Exclusive Read Exclusive Write (EREW) model. These are generally referred to as the Parallel Random Access Machines (PRAMs). But due to the lack of realization of such models, these gradually disappeared from research currency.

1.5.2 Content Distribution Network (CDN)

Content distribution networks have become popular with the growth of the WWW and offer several advantages. Imagine a web server that has to serve multiple requests to the same popular object, such as a web page containing a news flash of wide public interest. It is very likely that the server is overburdened quickly to keep up with the pace of the requests. Also, letting only one server handle all the requests becomes inefficient and expensive in terms of network usage as the server may have to serve requests from clients spread across various ISPs. In this case it might be efficient if the object is cached at various places in the network, for example at ISP boundaries, so that future requests can be handled from the cache without even involving the server. Such caches are also referred to as *surrogate servers*. Presently, many popular web sites make use of such surrogate servers provided by popular CDNs such as Akamai and Digital Island. Having surrogate servers itself does not solve the problem unless there is a way to make use of them. For this purpose, CDNs also provide *redirectors* that forward client requests to a surrogate server based on several criteria such as geographic proximity, server throughput, latency time, and client location [7]. The entire scheme can be picturized as shown in Figure 1.3.

The relation they have with overlay networks is that like peers in an overlay network, the redi-

[7] Notice that this is not the same as geographic proximity.

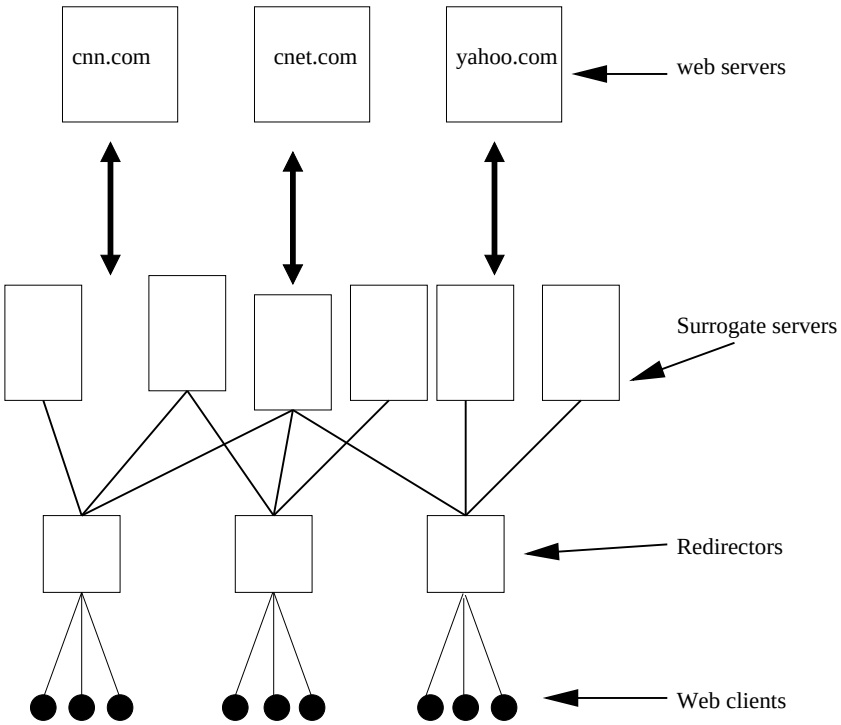


Figure 1.3: A CDN in operation. The figure is based on [120, Figure 9.27].

rectors make an application-level routing decision. Also, several problems such as which surrogate server to redirect, how to choose surrogate servers, have their equivalents in overlay networks so that solutions and techniques developed for one may prove to be useful in the other.

1.6 Organization of the thesis

The rest of the thesis is organized as follows. In Chapter 2, we introduce most of the terminology and notation that is common throughout the thesis. This serves as a background on the various technical terms used in the rest of the thesis. In Chapter 3, we provide a technical summary and significance of the results contained in this thesis.

In Part I, we look at vertex coloring algorithms. In Chapter 4, we present and analyze our distributed vertex coloring algorithm for oriented graphs.

Chapters 5–6 form Part II of the thesis dealing with peer-to-peer overlay networks. In Chapter 5 we describe our deterministic construction of overlay P2P networks and analyze concurrent multi-casting in the overlay network. In Chapter 6 we argue the case for supervised P2P overlay networks and provide a unified framework to create such a system. We also show how to provide robustness guarantees under a very powerful adversarial model.

Part III of the thesis focuses on overlay networks for wireless ad hoc networks. In Chapter 7 we describe our new model for wireless communication and proceed to show how to construct a constant density spanner. In Chapter 8 we show how to design efficient algorithms for broadcasting and information gathering in wireless networks.

The thesis ends with some concluding remarks and potential for further work in Chapter 9.

Chapter 2

Terminology and Notation

In this chapter we introduce the notation that is common across the rest of the thesis. We start by stating well known inequalities from algebra and also probability. We then provide a basic introduction to graph theory and will then introduce some popular families of networks and their structural properties. Finally, a short introduction to routing theory and terminology is presented.

2.1 Basic Notation

We denote by $\mathbb{N}$ the set of natural numbers $\{1, 2, 3, \ldots\}$ and by $\mathbb{N}_0$ the set of natural numbers including 0, i.e. the set $\{0, 1, 2, \ldots\}$. By $\mathbb{R}$ we denote the set of real numbers and by $\mathbb{R}^+$ we denote the set of non-negative real numbers. For any $x \in \mathbb{N}_0$, we denote by $[x]$ the set of natural numbers $\{0, 1, \ldots x-1\}$. If $x \in \mathbb{R}^+$, then $[x]$ would be the set $\{1, 2, \ldots, \lceil x \rceil\}$. By "log" we mean the logarithm to base 2 unless specified otherwise. For strings $x \in \{0,1\}^*$ we denote by $x/2$ as the string obtained by shifting x to the right by one position.

We use standard notation concerning the asymptotic behavior of functions. Consider any two functions f, g with domain from the set of natural numbers $\mathbb{N}$. We write $f(n) = O(g(n))$ if there

exist positive constants c, n_0 such that $\forall n \geq n_0,\ 0 \leq f(n) \leq c \cdot g(n)$. We write $f(n) = \Omega(g(n))$ if there exist positive constants c, n_0 such that $\forall n \geq n_0,\ \ f(n) \geq c \cdot g(n) \geq 0$. If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ then we write $f(n) = \Theta(g(n))$. We sometimes use the small-o, $o(.)$, and the small-omega, $\omega(.)$, notation defined as follows. We write $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ and write $f(n) = \omega(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, if the above limits exist.

We often use the following inequalities.

Proposition 2.1.1

- *For all* $x \in \mathbb{R}$, $1 + x \leq e^x$, *with equality occurring at* $x = 0$.
- *For all* $n, k \in \mathbb{N}$ *and* $k \leq n$, $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$.

2.2 Basic Probability

We start by defining probability and then introduce some well-known inequalities that we often use.

Let Ω be an arbitrary set, called the sample space. We start by defining a σ–field, also sometimes called a σ–algebra.

Definition 2.2.1 (σ–field) *A collection $\mathcal{F}$ of subsets of Ω is called a σ–field if it satisfies:*

1. $\Omega \in \mathcal{F}$
2. $A \in \mathcal{F}$ *implies* $A^c \in \mathcal{F}$, *and*
3. *For any countable sequence* $A_1, A_2, \ldots$, *if* $A_1, A_2, \ldots \in \mathcal{F}$ *then* $A_1 \cup A_2 \cup \ldots \in \mathcal{F}$.

Definition 2.2.2 *A set function* $\Pr$ *on a σ–field $\mathcal{F}$ of subsets of Ω such that* $\Pr : \mathcal{F} \to [0, 1]$ *is called a probability measure if it satisfies:*

1. $0 \leq \Pr(A) \leq 1, \ \forall A \in \mathcal{F}$.

2. $\Pr(\Omega) = 1$, *and*

3. *If* $A_1, A_2, \ldots$ *is a disjoint sequence of sets in* $\mathcal{F}$ *then*

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i)$$

The triad $(\Omega, \mathcal{F}, \Pr)$ is often called a probability space. For equivalent and alternate definitions, examples, and a more complete introduction, we refer the reader to standard textbooks on probability [15]. In the following, if no probability space is mentioned then any space $(\Omega, \mathcal{F}, \Pr)$ can be taken.

We often use the following inequality called "Boole's inequality" which is part of a general Boole-Bonferroni inequalities [109] and this is also sometimes referred to as the "union bound" as it provides a bound on the probability of a union of events. This inequality is also referred to as the (finite) sub-additivity property of the probability measure.

Proposition 2.2.3 (Boole's inequality) *For any arbitrary events* $A_1, A_2, \ldots A_n$,

$$\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr(A_i)$$

The notion of independence is an important concept in the study of probability.

Definition 2.2.4 (Independence) *A collection of events* $\{A_i : i \in I\}$ *is said to be* independent *if for all* $S \subseteq I$, $\Pr(\cap_{i \in S} A_i) = \Pi_{i \in S} \Pr(A_i)$.

We now define random variable, which is any *measurable* function from Ω to $\mathbb{R}$. Let $\mathcal{R}$ denote the standard Borel σ–field associated with $\mathbb{R}$, which is the σ–field generated by left-open intervals

of $\mathbb{R}$ [15].

Definition 2.2.5 (Random Variable) *Given a probability space* $(\Omega, \mathcal{F}, \Pr)$*, a mapping* $X : \Omega \rightarrow \mathbb{R}$ *is called a random variable if it satisfies the condition that* $X^{-1}(R) \in \mathcal{F}$ *for every* $R \in \mathcal{R}$*.*

We represent as $\{X \leq x\}$ as the set $\{\omega \in \Omega | X(\omega) \leq x\}$ for $x \in \mathbb{R}$ and also write $\Pr(X \leq x)$ as the probability of the above event. Similar definition can be made for representing the set $\{\omega \in \Omega | X(\omega = x\}$ as $\{X = x\}$.

The notion of independence also extends to random variables. Two random variables X and Y are said to be *independent* if the events $\{X \leq x\}$ and $\{Y \leq y\}$ are independent for $x, y \in R$. The definition extends to multiple random variables just as in Definition 2.2.4.

Associated with any random variable is a distribution function defined as follows.

Definition 2.2.6 (Distribution function) *The distribution function* $F : \mathbb{R} \rightarrow [0, 1]$ *for a random variable* X *is defined as* $F_X(x) = \Pr(X \leq x)$*.*

A random variable X is said to be a *discrete* random variable if the range of X is a finite or countably infinite subset of $\mathbb{R}$. For discrete random variables, the following definition can be provided for the *density* of a random variable.

Definition 2.2.7 (Density) *Given a random variable* X*, the density function* $f_X : \mathbb{R} \rightarrow [0, 1]$ *of* X *is defined as* $f_X(x) = \Pr(X = x)$*.*

The above definition can be extended to all types of random variables also with proper care. In the rest of this section, we focus on discrete random variables only and hence the definitions are made for the case of discrete random variables. With proper care, the definitions however can be extended [15].

An important quantity of interest of a random variable is its expectation.

Definition 2.2.8 (Expectation) *Given a probability space* $(\Omega, \mathcal{F}, \Pr)$ *and a random variable* X*, the expectation of* X*, denoted* $E[X]$*, is defined as*

$$E[X] = \sum_{x \in \mathbb{R}} x \Pr[X = x]$$

with the convention that $0 \cdot \infty = \infty \cdot 0 = 0$*.*

We now state tail inequalities of random variables. These are called tail inequalities since they provide a bound on the probability that a random variable deviates from its expectation.

Proposition 2.2.9 (Markov Inequality) *Given a non-negative-valued random variable* X *and any* $t \in \mathbb{R}^+ \setminus \{0\}$*,*

$$\Pr(X \geq tE[X]) \leq 1/t$$

Random variable X is said to have Bernoulli distribution with parameter p, where $p \in [0, 1]$, if X has the following density function.

$$f_X(x) = \begin{cases} 1 - p & \text{if } x = 0 \\ p & \text{if } x = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Using Proposition 2.2.9, the following famous inequality can be shown. For a proof, we refer the reader to standard text books such as [109].

Proposition 2.2.10 (Chernoff Bounds) *Let* $X_1, X_2, \ldots, X_n$ *be* n *independent Bernoulli random variables with* $\Pr(X_i = 1) = p$ *for all* $1 \leq i \leq n$*, and let*

$$X := \sum_{i=1}^{n} X_i$$

and $\mu := E[X] = np$*. Then for any* $\delta > 0$*,*

$$\Pr(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

and given $0 < \delta \leq 1$*,*

$$\Pr(X \leq (1-\delta)\mu) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$$

There exist several simplifications of the above proposition and the following form is often useful as it allows to bound the deviation δ required so that the tail probability is polynomially small, i.e., of the order $1/n^c$ for a constant $c > 0$.

Proposition 2.2.11 *Under the definitions of Proposition 2.2.10,*

$$\Pr(X \geq (1+\delta)\mu) \leq \begin{cases} e^{-\mu\delta^2/3} & \text{if } \delta < 1 \\ e^{-\mu\delta\log\delta} & \text{otherwise} \end{cases}$$

and similarly,

$$\Pr(X \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2}$$

More generally, the upper tail inequalities for X hold if we do not know μ but have an upper bound of μ so that $\mu \leq \mu^+$ and the lower tail inequalities hold if we have a lower bound on μ so that $\mu \geq \mu^-$. For these and other forms, we refer the reader to [108]. Such tail inequalities are known for sums of independent random variables that are distributed geometrically, hyper-geometrically, and other distributions. We refer the reader to [108] for these inequalities.

By the phrase "with high probability" (or w.h.p. for short) we mean a probability of at least $1 - (1/n^k)$ for some constant $k > 0$, where n is the number of elementary events (usually the number of messages) in a random experiment.

2.3 Basic Graph Theory

A *graph* $G = (V, E)$ consists of a set of *nodes* (or *vertices*) V and a set of *edges* (or *arcs*) $E \subseteq V \times V$. The nodes represent the processing units and the edges represent the communication links between the units. Often, we will set $n := |V|$ (the size of V) and $m := |E|$. The *size* of G is defined as the number of nodes it contains. For all $v, w \in V$, (v, w) denotes a *directed* edge from v to w, and $\{v, w\}$ denotes an *undirected* edge from v to w. G is called *undirected* if $E \subseteq \{\{v, w\} \mid v, w \in V\}$ and *directed* if $E \subseteq \{(v, w) \mid v, w \in V\}$. Unless explicitly mentioned, in the following we assume that G is undirected.

A sequence of contiguous edges in G is called a *path*. The *length* of the path is defined as the number of edges it contains. A graph G is said to be *connected* if there exists a path between every pair of vertices $u, v \in V$. A sequence of contiguous edges $(u_1, u_2), (u_2, u_3), \ldots, (u_{n-1}, u_n)$ forms a *cycle* if $u_n = u_1$. G is called a *tree* if it is connected and contains no cycle. A graph $T = (V', E')$ is called a *spanning tree* of G if $V' = V$, $E' \subseteq E$, and T is a tree. G is called *bipartite* if its node set can be partitioned into two node sets V_1 and V_2 such that $E \subseteq \{\{v, w\} \mid v \in V_1,\ w \in V_2\}$.

For any pair of nodes $v, w \in V$, let $d(v, w)$ denote the *distance* between v and w in G, that is, the length of a shortest path from v to w. The *diameter* D of G is defined as $\max\{d(v, w) \mid v, w \in V\}$. If the graph G is not connected, then we say that the diameter of the graph is infinite. If $\{v, w\} \in E$ then v is called a *neighbor* of w. For any subset $U \subseteq V$, the *neighborhood* of U is defined as

$$\Gamma(U) = \{v \in V \setminus U \mid \exists u \in U,\ \{u, v\} \in E\} .$$

The number of neighbors of v is called the *degree* of v and denoted by d_v. The degree of G is defined as $\Delta = \max\{d_v \mid v \in V\}$.

A family of graphs $\mathcal{G} = \{G_n \mid n \in \mathbb{N}\}$ has degree $d(n)$ if for all $n \in \mathbb{N}$ the degree of G_n is

$d(n)$. If it is clear to which family a is considered to belong, we say that this graph has constant (or bounded) degree if its family has constant (or bounded) degree.

A *network* is specified by a graph $G = (V, E)$ with edge capacities given by a function $c : E \to \mathbb{R}^+$. Given a graph G with capacities c, let the capacity of a node $v \in V$ be defined as

$$c(v) = \sum_{w \in V} c(v, w)$$

and the capacity of any node set $U \subseteq V$ be defined as $c(U) = \sum_{u \in U} c(u)$. Given a subset $U \subseteq V$, $(U, \bar{U})$ denotes the set of all edges $\{u, v\} \in E$ (or $(u, v) \in E$ if G is directed) with $u \in U$ and $v \in \bar{U}$. So $c(U, \bar{U})$ is the sum of the capacities of all edges in $(U, \bar{U})$. The *edge expansion* α of a network G with capacities c is defined as

$$\alpha = \min_{U \subseteq V} \frac{c(U, \bar{U})}{\min\{c(U), c(\bar{U})\}} .$$

In the above definition, U or $\bar{U}$ cannot be taken to be Φ or V. For every network $G = (V, E)$ with non-negative edge capacities, the edge expansion can be at most 1. The *node expansion* of a network G is defined as the ratio $\min_{S \subset V,\, |S| \leq |V|/2} \Gamma(S)/|S|$. In the definition of node expansion, the set S cannot be taken to be empty. The node expansion can also be at most 1.

2.4 Basic Network Topologies

Unless explicitly mentioned, we will treat all edges in the following to be of capacity 1. The most basic network topologies used in practice are trees, cycles, and meshes. Many other popular networks can be seen as either combinations or extensions of these. We start by recalling the definition of a tree.

Definition 2.4.1 (Tree) *A graph $G = (V, E)$ is called a* tree *if it is connected and contains no cycle.*

Definition 2.4.2 (Mesh) *Let $m, d \in \mathbb{N}$. The (m, d)-*mesh *$M(m, d)$ is a graph with node set $V = [m]^d$ and edge set*

$$E = \left\{ \{(a_{d-1}, \ldots, a_0), (b_{d-1}, \ldots, b_0)\} | a_i, b_i \in [m], \sum_{i=0}^{d-1} |a_i - b_i| = 1 \right\} .$$

*The (m, d)-*torus *$T(m, d)$ is a graph that consists of an (m, d)-mesh and additionally wrap-around edges from $(a_{d-1} \ldots a_{i+1}(m-1)\ a_{i-1} \ldots a_0)$ to $(a_{d-1} \ldots a_{i+1}\ 0\ a_{i-1} \ldots a_0)$ for all $i \in [d]$ and all $a_j \in [m]$ with $j \neq i$.*

$M(m, 1)$ is also called a *line* or *path*, $T(m, 1)$ a *cycle*, and $M(2, d) = T(2, d)$ a *d-dimensional hypercube*. Figure 2.1 presents a tree and Figure 2.2 presents a line, a torus, and a hypercube.

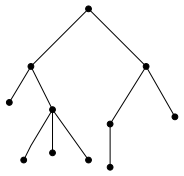


Figure 2.1: The structure of a tree.

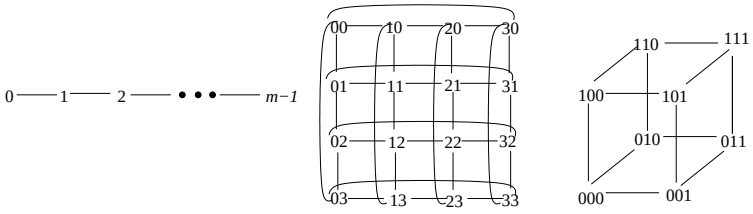


Figure 2.2: The structure of $M(m, 1)$, $T(4, 2)$, and $M(2, 3)$.

The hypercube is a very important class of networks, and many modifications, the so-called *hypercubic networks*, have been suggested for it. Prominent among these are the butterfly, and de Bruijn graph. We start with the butterfly.

Definition 2.4.3 (Butterfly) *Let $d \in \mathbb{N}$. The d-*dimensional butterfly *$BF(d)$ is a graph with node set $V = [d+1] \times [2]^d$ and edge set $E = E_1 \cup E_2$ with*

$$E_1 = \{\{(i,\alpha),(i+1,\alpha)\} \mid i \in [d],\ \alpha \in [2]^d\}$$

and

$$E_2 = \{\{(i,\alpha),(i+1,\beta)\} \mid i \in [d],\ \alpha,\beta \in [2]^d,\ \alpha \text{ and } \beta \text{ differ precisely at the } i\text{th position}\}\ .$$

The node set $\{(i,\alpha) \mid \alpha \in [2]^d\}$ represents level *i of the butterfly.*

Figure 2.3 shows the 3-dimensional butterfly $BF(3)$. The $BF(d)$ has $(d+1)2^d$ nodes, $2d \cdot 2^d$ edges and degree 4. Contracting the node sets $\{(i,\alpha) \mid i \in [d]\}$ into a single node results in the hypercube. Thus, the butterfly graph can be seen as a rolled-out version of a hypercube.

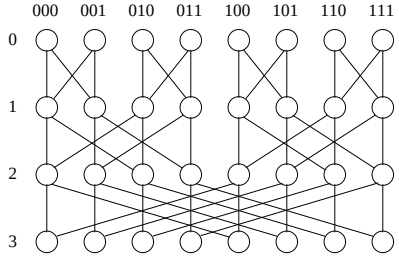


Figure 2.3: The structure of BF(3).

Definition 2.4.4 (de Bruijn) *The b-*ary de Bruijn graph of dimension *d $DB(b,d)$ is an undirected graph $G = (V,E)$ with node set $V = \{v \in [b]^d\}$ and edge set E that contains all edges $\{v,w\}$ with the property that $w \in \{(x, v_{d-1}, \ldots, v_1) :\ x \in [b]\}$, where $v = (v_{d-1}, \ldots, v_0)$.*

Two examples of a de Bruijn graph can be found in Figure 2.4.

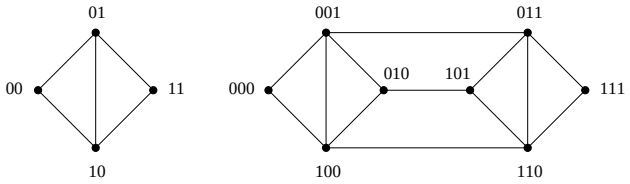


Figure 2.4: The structure of $DB(2,2)$ and $DB(2,3)$.

For the classes of graphs we presented above the expansion is quite complicated to compute. Therefore, we just state that the d-dimensional hypercube, butterfly, and de Bruijn graph with uniform edge capacities all have an expansion of $\Theta(1/d)$.

2.5 Basic Routing Theory

In this section we provide definitions for the various routing problems that arise in communication networks and also some parameters to measure the effectiveness of routing strategies. In a graph $G = (V, E)$ let the nodes be numbered distinctly from $[n]$.

Given a network G, some of the well known routing paradigms that are studied usually are:

- **Gathering:** In this mode, also called information-gathering, all the packets have the same destination in G. The destination is sometimes referred to as the *sink*.
- **Unicasting:** In this mode, each packet has a single destination in G.
- **Permutation Routing:** In this model, let $\pi : [n] \rightarrow [n]$ be any permutation. The problem is then to send one packet from node numbered i to node numbered $\pi(i)$ for each $i \in [n]$.
- **Multicasting:** In multicasting each packet has a set $T \subseteq V$ of destinations and the packet has to be delivered to all the nodes in T.
- **Broadcasting:** Here, each packet has to be delivered to all the nodes in G.

Two important parameters that are widely used to measure the quality of a routing strategy are *congestion* and *dilation*. To define these terms we need some more notation.

Given a network $G = (V, E)$ with an assignment of non-negative capacity to edges, $c : E \rightarrow \mathbb{R}^+$, a *multi-commodity flow* instance on G is a set of ordered pairs $(s_1, t_1), (s_2, t_2), \cdots, (s_k, t_k)$. Each pair (s_i, t_i) denotes a commodity with source s_i and target t_i. The task is to maximize the amount of flow traveling from the sources to the corresponding targets subject to the capacity constraints. The problem is studied in two variations, called the *maximum throughput multi-commodity flow problem* and the *maximum concurrent flow problem*. In the former, one is interested in finding a feasible solution $\mathcal{S}$ that maximizes the total flow over all commodities. In the latter each (s_i, t_i) has an associated demand d_i and the objective is to maximize the fraction of the demand that can be shipped simultaneously for all commodities. Given a feasible solution $\mathcal{S}$ to a maximum concurrent flow problem, the *concurrent flow value* is the minimum over all commodities of the fraction of the demand met by $\mathcal{S}$.

The *congestion* of $\mathcal{S}$, denoted $C(\mathcal{S})$, is defined as the reciprocal of the concurrent flow value of the solution $\mathcal{S}$. Intuitively, the congestion specifies by what factor the edge capacities would have to be increased in order to satisfy the demands of all the commodities when using the solution $\mathcal{S}$. The *dilation* of $\mathcal{S}$, denoted $D(\mathcal{S})$, is the length of the longest flow path in $\mathcal{S}$. For a good solution, the congestion and the dilation should be small.

Chapter 3

Our Contributions

In this chapter, we provide a technical overview of the results contained in this thesis. We first show the common thread that brings together the questions answered in this thesis before showcasing the contributions of the thesis and their technical significance.

3.1 Key Questions

Several projects involving overlay networks for file sharing, data management, grid computing, etc. have been initiated in the recent past both inside and outside of the research community. We argue that, as a common theme in the study of overlay networks, the following three questions are of central interest.

1. **Connectivity:** How to ensure that the overlay network has a single connected component?
2. **Maintenance:** How to maintain the overlay network as nodes join and leave the network?
3. **Routing:** How to solve routing problems in the overlay network efficiently?

Why are the above mentioned questions important and non-trivial? As can be observed, for the overlay network to be useful for routing, it is crucial that the overlay network has at least one path for each pair of nodes in the network. This means that the overlay network should have a single connected component. In the case of randomized constructions, one also states this requirement as the overlay network being connected with high probability or having a giant connected component. A giant connected component means that there exist a connected component whose size is of the order $n(1 - o(1))$ when there are n nodes in the network.

One of the features of overlay networks that sets them apart from traditional networks is that overlay networks are not static and change over time. For example, in a peer-to-peer network with peers connected via the Internet, nodes may join/leave the network at a rapid rate. Similarly, having mobile nodes as in the case of a wireless ad hoc network results in changes to the underlying topology over time. In such cases, it is important that the overlay network adapts to changes in an efficient manner and also preserve connectivity in the existing network. By efficiency, we mean that the network be able to perform few local-control operations and minimal amount of work to return to a valid state as the network changes over time.

Routing in overlay networks requires careful selection of paths so that the routing problem can be solved efficiently. Depending on the nature of the routing problem and the nature of the network additional challenges arise. Some of the parameters we will be interested in are, the congestion caused by the routing strategy, the time taken for packets to reach their destination, and the buffer overhead needed at the nodes in the network.

In this thesis, we study the above issues in detail with specific focus on two classes of overlay networks. In Part I of the thesis we look at vertex coloring algorithms when the overlay network is modeled as a graph. Vertex coloring algorithms find applications in other problems such as

scheduling and routing. In Part II of the thesis, we look at peer-to-peer overlay networks for load balancing, concurrent multicasting and robustness issues in peer-to-peer networks. Our construction has been designed to handle heterogeneous peers in an efficient way and an additional focus will be on deterministic constructions so that one can provide guarantees on the properties of the network such as the diameter and the degree and also have the advantage of self-stabilization. In Part III of the thesis, we consider overlay networks for wireless ad hoc networks. In wireless ad hoc networks, the lack of centralized infrastructure and mobility pose heavy challenges on the design of efficient overlay networks. Also, routing in wireless networks is prone to interference problems and hence the overlay network has to handle such problems to be able to perform efficient routing.

3.2 Vertex Coloring

In the first part of the thesis, we focus on vertex coloring algorithms. Vertex coloring is a fundamental problem in graph theory and has applications to many problems such as scheduling and clustering. Since vertex coloring is used as a sub-routine in many higher-order communication and computation tasks in algorithms for overlay networks, faster, efficient and local-control vertex coloring algorithms are of interest. For example, consider a wireless ad hoc network where the topology may undergo changes over time. In this case, when using coloring for scheduling tasks, one has to recompute the coloring so as to arrive at a new schedule reflecting the changes in the topology. We treat the vertex coloring problem in the distributed setting where we are given a certain graph and has to arrive at a coloring in a local-control manner.

Given a graph G with maximum degree Δ it is easy to see that G can be vertex colored using $\Delta + 1$ colors. Distributed vertex coloring algorithms that color a graph G of n vertices with maximum degree Δ in a logarithmic number of communication rounds are known since more than

a decade [102]. We first show that this is the best possible, i.e., *any* distributed coloring algorithm in which every node has the same initial state and initially only knows its own neighbors requires $\Omega(\log n)$ rounds, with high probability, to arrive at a proper coloring. We show the following theorem. By a Las Vegas algorithm, it is meant that the algorithm always produces a correct output while the time required for the algorithm is a random variable.

Theorem 3.2.1 *For every Las Vegas algorithm A there is an infinite family of non-oriented graphs $\mathcal{G}$ s.t. A has a bit complexity of at least $\Omega(\log n)$ on $\mathcal{G}$, with high probability, to compute a proper vertex coloring.*

But, what if the edges in the graph are oriented, i.e., the end-points of an edge agree on its orientation while bits can still flow in both directions? We show that in this new model, 3-coloring an oriented cycle graph of n nodes can be done by exchanging $O(\sqrt{\log n})$ bits. We also show that this result is best possible by proving a lower bound of $\Omega(\sqrt{\log n})$ on the number of bits exchanged by any distributed algorithm to arrive a proper coloring of an oriented cycle, for any finite number of colors. We then extend our analysis to $(\Delta + 1)$–color graphs of degree Δ provided that G has no oriented cycle of length less than $\sqrt{\log n}$ and Δ is bounded by a constant. Using more techniques, we then provide a distributed algorithm for obtaining an $(1 + \epsilon)\Delta$–coloring of a graph of n nodes with degree Δ using only $\tilde{O}(\sqrt{\log n})$ bit exchanges, with high probability, provided that the graph does not have any oriented cycle of length less than $\sqrt{\log n}$. By $g(n) = \tilde{O}(f(n))$ we mean $g(n) = O(f(n)\text{polylog}(f(n)))$. We show the following theorem:

Theorem 3.2.2 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, for any constant $\epsilon > 0$, a $(1 + \epsilon)\Delta$–vertex coloring of G can be obtained by exchange of $O(\log \Delta) + \tilde{O}(\sqrt{\log n})$ bits, with high probability.*

To the best of our knowledge, this is the first sub-logarithmic algorithms for vertex coloring. By sub-logarithmic we mean that algorithms that require only sub-logarithmic number of bits to be exchanged during the algorithm. The result is possible only because we consider the case where the edges are oriented.

3.3 Peer-to-peer networks

The second part of the thesis deals with P2P overlay networks.

3.3.1 Deterministic Construction for Heterogeneous Peers

While topologies for peer-to-peer overlay have been studied heavily in the theoretical community, see [142, 125, 6] for example, two fundamental questions remained. One question is whether it is possible to arrive at deterministic constructions that match the performance of the randomized constructions. Deterministic constructions are sought after as it is easy to provide guarantees about their properties and can be made to self-stabilize [32], which is an important property for overlay networks. The second question asks whether nodes of non-uniform bandwidth can be integrated into the network efficiently so as to take advantage of them. Existing constructions treat the situation where all the nodes have the same bandwidth and hence in their basic form fail to utilize the presence of nodes with significantly high bandwidth. In reality, however, peers have connections to the Internet using different mechanisms that differ in the amount of bandwidth available to the peer. Moreover, even if all peers have sufficient bandwidth, as mentioned in Chapter 1, future generation P2P systems have to allow for peers to contribute bandwidth based on their needs and be able to operate efficiently under this form of heterogeneity. Hence, topologies that take into account the fact that the peers may differ in their bandwidth by orders of magnitude are of interest.

In Chapter 5, we propose a deterministic topology that address both of the above questions in a satisfactory manner. Our construction called *Pagoda* has the following properties.

Pagoda network for nodes of uniform bandwidth

For the uniform case, i.e., where all the nodes have the same bandwidth, the Pagoda network of n nodes can:

- **Maintenance:** perform an isolated peer insertion or removal with $O(\log n)$ time and work.
- **Routing:** route arbitrary permutation of size n in $O(\log n)$ steps, with high probability, and
- **Data Management:** distribute data among nodes in the network so that no node receives more than an expected $O(1/n)$ fraction of the data, provided there are at least n data items.

Pagoda network for nodes of non-uniform bandwidth

For the case of the non-uniform peers, the Pagoda network on n nodes can:

- **Maintenance:** perform an isolated peer insertion or removal with $O(\log^2 n)$ time and work, and
- **Multicasting:** route arbitrary concurrent multicast requests with a congestion that is only by an $O(\Delta + \log n)$ factor larger than the congestion created in an optimal network of degree Δ for *the particular problem*.

We also provide strategies for admission control and also show that the existential routing strategy for the case of concurrent multicasting can be turned into local-control strategies for building and maintaining multicast trees.

All the above properties make our solution highly attractive. Our construction shows that efficient peer-to-peer networks that operate under heterogeneous environments can be designed. An additional advantage of our construction is that it is deterministic in nature and matches the properties of most known randomized constructions for join/leave operations. Thus, we solve two open problems in this area.

3.3.2 Supervised Peer-to-peer Systems

Another direction that was pursued in P2P networks is the study of supervised P2P networks. For purposes of brevity, here we let the supervisor be a special node in the network which is responsible for guiding the peers during a join or leave operation where as the rest of the operations in the peer-to-peer network do not involve the supervisor. We argue in Chapter 6 that supervised peer-to-peer systems offer the benefits of both traditional server-based systems and those of peer-to-peer systems without inheriting their disadvantages.

Supervised overlay networks with specific topologies such as a tree, and the de Bruijn graph have been studied recently by Riley and Scheideler [128, 127]. We present a unified methodology that allows one to build a large class of supervised P2P networks by combining techniques such as the hierarchical decomposition technique [125], the continuous-discrete approach [110] and the recursive labeling technique. We also show how to extend the basic scheme so as to allow for concurrent operations, and also rapid repair.

We show that such supervised overlay networks have applications to many areas such as grid computing and multi-player online gaming. We also show how to provide robustness guarantees under a strong adaptive adversarial model [132].

3.4 Wireless Ad Hoc Networks

In the third part of the thesis, we consider overlay networks for wireless ad hoc networks. Overlay networks for wireless ad hoc networks have been studied mostly in the packet radio model or the even simpler unit disk model. These models have significant drawbacks and fail to model the realities of wireless networks in many situations. Hence we propose a new model for wireless networks that tackles the limitations of the existing models. In our model, the non-uniformity in the environment is addressed by having a cost function model the transmission and interference ranges. Additionally, we are the first to introduce a model for physical carrier sensing in wireless networks. Without physical carrier sensing, it was shown in [66] that in an n–node network, $\Omega(n)$ time steps are required even to transmit one message successfully under the assumption that nodes do not have any knowledge of the network.

Using our model of wireless ad hoc networks, in Chapter 7, we propose local-control algorithms that build a constant density spanner of the original network. A constant density spanner for a graph G can be defined as follows. Given an undirected graph $G = (V, E)$, a subset $U \subseteq V$ is called a *dominating set* if all nodes $v \in V$ are either in U or have an edge to a node in U. A dominating set U is called *connected* if U forms a connected component in G. The *density* of a dominating set is the maximum over all nodes $v \in U$ of the number of neighbors that v has in U. In our context, *constant density spanner* is a connected dominating set U of constant density with the property that for any two nodes $v, w \in V$ there are two nodes $v', w' \in U$ with $\{v, v'\} \in E$, $\{w, w'\} \in E$, and a path p from v' to w' along nodes in U so that the length of p is at most a constant factor larger than the distance between v and w in G.

Our algorithms are self-stabilizing and require only constant storage at any node and do not require that nodes have globally unique labels. Thus our algorithms are applicable to a wide range

of situations such as simple wireless sensor networks. While distributed algorithms to construct topological spanners are known, (see e.g., [36, 84]), it is non-trivial to arrive at such a construction using our model of wireless communication.

Let us denote by $G = (V, E)$ the graph representing the topology of the wireless ad hoc network where V denotes the set of wireless stations and $E \subseteq V \times V$ denotes the set of edges with edge $(u, v) \in E$ if and only if u and v can communicate directly with each other. Notice that this definition does not specify the condition when nodes u and v can communicate directly as that can be influenced by the model for wireless communication. We arrive at the following.

Theorem 3.4.1 *For any initial situation, given the graph G, the dominating set protocol generates a constant density dominating set of G in $O(\log^4 n)$ communication rounds, with high probability.*

Using the above dominating set and additional techniques, we show how to arrive at a constant density spanner. We show the following theorem in Chapter 7.

Theorem 3.4.2 *For any initial situation, given the graph G, the spanner protocol generates a constant density spanner of G in $O(D \log^2 n + \log^4 n)$ communication rounds, with high probability, where D is the maximum number of nodes that are within the transmission range of a node.*

Subsequently in Chapter 8, we show how to support higher order communication primitives such as broadcasting, and information gathering in a time- and work efficient manner.

While broadcasting itself appears to be an easy problem, and is heavily studied analytically and empirically, realizing it in an efficient and reliable way is very difficult. Often, the over-simplification introduced in the model render many algorithms inefficient in some situations. We construct such situations explicitly in Chapter 8. Our algorithms build on top of the constant density spanner and are also self-stabilizing. For broadcasting $m \geq 1$ messages from a source node s, we show the following theorems concerning the time and work requirement.

Theorem 3.4.3 *Given the constant density spanner of G, the broadcast algorithm needs $O(T(s,m) + \log n)$ rounds, with high probability, where $T(s,m)$ is the optimal amount of time required to deliver m broadcast message to all nodes.*

Theorem 3.4.4 *Given a constant density spanner of G, the broadcast algorithm needs $O(W(s,m))$ work where $W(s,m)$ is the optimal work required to send m broadcast messages to all nodes in the system.*

We then consider the situation where m packets distributed arbitrarily in the network are to be delivered to a special sink node, s. We provide a two stage protocol for this problem that achieves the following time and work bounds. Let Δ_m be the maximum density of nodes having at least one packet to send to s. We show the following results in Chapter 8.

Theorem 3.4.5 *Given a constant density spanner of G, the information gathering protocol needs $O(T'(s,m) + \Delta_m \log^2 n)$ time steps w.h.p. where $T'(s,m)$ is the optimal time required for all the packets to be delivered to the sink node s.*

Theorem 3.4.6 *Given a constant density spanner of G, the gathering protocol needs $O(W'(s,m))$ work where $W'(s,m)$ is the optimal work required to send all the m packets to the sink node s.*

Part I

Vertex Coloring

Chapter 4

Vertex Coloring

We consider the well-known vertex coloring problem: given a graph G, find a coloring of its vertices so that no two neighbors in G have the same color. It is trivial to see that every graph of maximum degree Δ can be colored with $\Delta+1$ colors, and distributed algorithms that find a $(\Delta+1)$-coloring in a logarithmic number of communication rounds, with high probability, are known since more than a decade. This is in general the best possible if only a constant number of bits can be sent along every edge in each round. In fact, we show that for the n-node cycle the *bit complexity* of the coloring problem is $\Omega(\log n)$. More precisely, if only one bit can be sent along each edge in a round, then *every* distributed coloring algorithm (i.e., algorithms in which every node has the same initial state and initially only knows its own edges) needs at least $\Omega(\log n)$ rounds, with high probability, to color the n–node cycle, for *any* finite number of colors. But what if the edges have orientations, i.e., the endpoints of an edge agree on its orientation (while bits may still flow in both directions)? Edge orientations naturally occur in dynamic networks where new nodes establish connections to old nodes. Does this allow one to provide faster coloring algorithms?

Interestingly, for the n–node cycle in which all edges have the same orientation, we show

that a simple randomized algorithm can achieve a 3-coloring with only $O(\sqrt{\log n})$ rounds of bit transmissions, with high probability (w.h.p.). This result is tight because we also show that the bit complexity of coloring an n–node oriented cycle is $\Omega(\sqrt{\log n})$, with high probability, no matter how many colors are allowed. The 3-coloring algorithm can be easily extended to provide a $(\Delta + 1)$-coloring for all graphs of maximum degree Δ in $O(\sqrt{\log n})$ rounds of bit transmissions, w.h.p., if Δ is a constant, the edges are oriented, and the graph does not contain an oriented cycle of length less than $\sqrt{\log n}$. Using more complex algorithms, we show how to obtain an $O(\Delta)$-coloring for arbitrary oriented graphs of maximum degree Δ using essentially $O(\log \Delta + \sqrt{\log n})$ rounds of bit transmissions, w.h.p., provided that the graph does not contain an oriented cycle of length less than $\sqrt{\log n}$.

4.1 Introduction

A fundamental problem in distributed systems is to compute a proper vertex coloring. The importance of vertex coloring can be seen by observing that many distributed algorithms use such a coloring as a sub-routine in higher-order communication and computation tasks. Examples include scheduling [88], resource allocation [26], and synchronization. Vertex coloring has applications also in wireless networks to determine cluster heads, (see for example [75] and the references therein), routing in wireless networks [88], and in many parallel algorithms [68, 70]. Thus, it is not surprising that this problem has been heavily studied not only in the distributed setting but also in the PRAM model of computation starting with Karp and Wigderson [70] and Luby [102].

We consider distributed systems that can be modeled as a graph $G = (V, E)$ with nodes representing the processors and the edges representing the communication links. Given a graph $G = (V, E)$ with maximum degree Δ, the vertex coloring problem is to find a color assignment

for the vertices of G so that no two adjacent vertices are given the same color. The minimum number of colors required to properly color a graph is called its *chromatic number*, and is denoted by $\chi(G)$. While it is easy to see that a graph with maximum degree Δ can be colored using at most $\Delta+1$ colors, computing the chromatic number of a graph is NP–hard [48]. Further, $\chi(G)$ cannot be approximated to any reasonable bound in general [41]. Thus, efficient algorithms that color using $\Delta+1$ colors are of interest.

In the distributed model of computing, communication is an expensive resource and distributed algorithms therefore aim at using as little communication as possible. Distributed algorithms for vertex coloring take the approach of minimizing the number of communication rounds assuming that in each round a reasonable number of bits can be communicated. Deterministic distributed algorithms for $(\Delta+1)$-coloring that run in a polylogarithmic number of rounds are not known. The best known deterministic algorithm [117] requires $n^{O(1/\sqrt{\log n})}$ rounds where n is the number of vertices. However, randomization can improve the runtime exponentially and in some special cases, such as highly dense graphs, even double exponentially [53]. Randomized distributed algorithms that compute a $(\Delta+1)$–coloring in $O(\log n)$ rounds, with high probability[1], are known since more than a decade [103, 64]. In this work we show that, interestingly, if the underlying graph G is provided with an orientation on its edges such that the orientation does not induce oriented cycles of length at most $\sqrt{\log n}$, then vertex coloring with $(1+\epsilon)\Delta$ colors for a constant $\epsilon > 0$, can be obtained by exchanging essentially $O(\log\Delta + \sqrt{\log n})$ bits, with high probability. Thus, we show that having orientations on the edges significantly improves the performance of distributed vertex coloring algorithms. We refer the reader to Section 4.1.3 for precise statements regarding our results.

We also note that providing an orientation is not cumbersome. If the nodes have unique labels

[1]With high probability (w.h.p.) means a probability that is at least $1-(1/n^c)$ for $c \geq 1$.

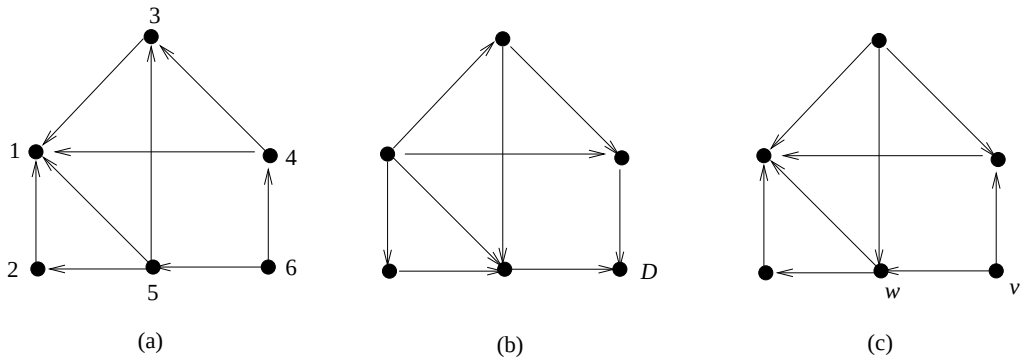


Figure 4.1: Figure shows that edge orientations can be provided naturally in many scenarios.

that are taken from a set with a total order then the labels induce a natural acyclic orientation on the edges. Edge $e = (u, v)$ is oriented from $u \rightarrow v$ if label$(u) >$ label(v) and vice-versa as shown in Figure 4.1(a). Another natural orientation can be provided as follows, for example in sensor networks. Information gathering is an important communication primitive for sensor networks where all the packets have to be forwarded to a single common destination called the *observer* [61, 78]. Many protocols for information gathering in sensor networks [61, 78] assume that the direction to the observer is available. In such a scenario, an orientation for the edges can be provided according to the distance of the endpoints to the observer. Ties between nodes with equal distance to the observer can be broken arbitrarily and the resulting orientation will be free of cycles as shown in Figure 4.1(b). Edge orientations naturally occur also in dynamic networks where new nodes establish connections to old nodes. Here, edge (v, w) may be oriented as $v \rightarrow w$ if w is an existing node in the network and v is a new node joining the network as shown in Figure 4.1(c).

4.1.1 Model and Definitions

We model the distributed system as a graph $G = (V, E)$ with V representing the set of computing entities, or processors, and $E \subseteq V \times V$ representing all the available communication links. We assume that all the communication links are undirected and hence bidirectional. All the processors

start at the same time and time proceeds in synchronized rounds. We let $n = |V|$. The degree of node u is denoted d_u and by Δ we denote the maximum degree of G, i.e., $\Delta = \max_{u \in V} d_u$. When there is no confusion, d_u will also be used to refer to the number of uncolored neighbors of node u. By N_u we denote the set of neighbors of node u and when there is no confusion, we use N_u to refer to the set of uncolored neighbors of u. We do not require that the nodes in V have unique labels of any kind. For our algorithms to work, it is enough that each node knows a constant factor estimate of the logarithm of the size of the network apart from its own degree and neighbors. When we consider graphs of constant degree, *no* global knowledge is required for our algorithm and it suffices that each node knows its own degree.

Given a graph $G = (V, E)$ a vertex coloring is a mapping $c : V \rightarrow [C]$ such that if $\{u, v\} \in E$ then $c(u) \neq c(v)$, i.e., no two adjacent vertices receive the same color. Here C denotes the number of colors used in the coloring. We say that a coloring is a *local coloring* if every node u with degree d_u has a color in $[\epsilon d_u]$ when the coloring uses $\epsilon\Delta$ colors. The interest in local coloring arises from the fact that a local coloring has nice implications when using the coloring in scheduling and routing problems [88].

In our model, the measure of efficiency is the number of bits exchanged. We also refer to this as the *bit complexity*. We view each round of the algorithm as consisting of 1 or more *bit rounds*. In each bit round each node can send/receive at most 1 bit from each of its neighbors. We assume that the rounds of the algorithm are synchronized. The bit complexity of algorithm A is then defined as the number of bit rounds required by algorithm A. We note that, since the nodes are synchronized, each round of the algorithm requires as many bit rounds as the maximum number of bit rounds needed by any node in this round. In our model, we do not count local computation performed by the nodes. This is reasonable as in our algorithms nodes perform only simple local computation.

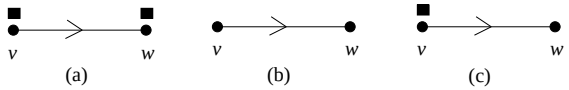


Figure 4.2: Orientation helps in symmetry breaking. In Figure (a) both v and w choose the same color. In (b), for existing algorithms both remain uncolored whereas in (c), when using orientation, node v may get colored.

In our model, we assume that the edges in E have an orientation associated with them. That is, for any two neighbors v, w exactly one of the following holds for the edge $\{v, w\}$: $\{v, w\}$ is oriented either $v \rightarrow w$ or as $w \rightarrow v$. In the former we also call v *superior* to w and vice-versa in the latter. Having orientation on the edges is a property that has not been studied in the context of vertex coloring though it is a natural property since networks usually evolve and for every connection there is usually a node that initiated it. We show that algorithms for symmetry breaking can be greatly improved provided that the underlying graph is oriented. The exact way in which orientation is used for symmetry breaking is explained in Figure 4.2. As shown, if nodes v and w choose the same color during any round of the algorithm, in the existing algorithms, both nodes remain uncolored as in Figure 4.2(b) and have to try in a later round. With orientation, if the edge $\{v, w\}$ is oriented as $v \rightarrow w$ as shown in Figure 4.2(c), then node v can retain its choice provided that there is no edge $\{u, v\}$ oriented $u \rightarrow v$ and u also chooses the same color.

Even though the graphs is equipped with orientation on the edges we still allow that on any edge bits can still flow in both the directions. Thus we consider undirected graphs but with the edge orientations.

One parameter that will be important for our investigations is the length of the shortest cycle in the orientation. We formalize this notion in the following definition.

Definition 4.1.1 (*ℓ–acyclic Orientation)* *An orientation of the edges of a graph is said to be ℓ–acyclic if the minimum length of any directed cycle induced by the orientation is at least ℓ. Note*

that this is not the girth of the given graph.

We always assume that the input graph is provided with a $\sqrt{\log n}$–acyclic orientation.

4.1.2 Related Work

The problem of vertex coloring in distributed systems has a long and rich history. It is an open problem whether deterministic poly-logarithmic time distributed algorithms exist for the problem of $(\Delta+1)$-vertex coloring [117]. The best known deterministic algorithm to date is presented in [117] and requires $n^{O(1/\sqrt{\log n})}$ rounds. Following considerations known from the radio broadcasting model [12] the problem cannot be solved at all in a deterministic round model without the use of unique identification numbers. Hence, most of the algorithms presented are randomized algorithms.

Karp and Widgerson [70] have shown that a MIS can be found in $O(\log^3 n)$ rounds w.h.p. and Luby [102] presents algorithms to find MIS in arbitrary graphs in $O(\log n)$ round with high probability. Luby [103] and Johansson [64] present parallel algorithms that can be interpreted as distributed algorithms that provide a $(\Delta + 1)$–coloring of a graph G in $O(\log n)$ rounds, with high probability. In the algorithm presented in [103], in every round each node that is not yet colored has a probability of choosing a color which is set to $1/2$. Luby's algorithm requires only pairwise independence and a derandomization was also shown in [103] for the PRAM (Parallel Random Access Machine) model of computation. Without such wakeup Johansson [64] presents an algorithm for $\Delta + 1$ distributed coloring. Recent empirical studies [43] have shown that the constant factors involved are small and also that a wakeup probability of 1 as in the algorithm of [64] reduces the number of rounds required. However, the analytical reason for this behavior is not known. Algorithms for vertex coloring are also presented in [51, 68] in the PRAM model of computation.

All of the above cited algorithms can be implemented as distributed algorithms in the message passing model and run in poly-logarithmic rounds with bit complexity $O(\log n \log \Delta)$, with high probability. Cole and Vishkin [28] and Goldberg et. al. [51] have shown that a $(\Delta + 1)$–coloring of the cycle graph on n nodes can be achieved in $O(\log^* n)$ communication rounds. This was shown to be optimal in Linial [98] by establishing that 3-coloring an n–node cycle graph cannot be achieved in less than $(\log^* n - 1)/2$ rounds. When unlimited local computation is available Linial [98] shows how to obtain an $O(\Delta^2)$ coloring in $O(\log^* n)$ rounds. This was later improved by De Marco and Pelc [106] to show that an $O(\Delta)$ coloring can be achieved in $O(\log^*(n/\Delta))$ rounds.

In a related work, Grable and Panconesi [53] present a distributed algorithm in the message passing model for edge coloring that runs in $O((1+\alpha^{-1}) \log \log n)$ rounds provided that the degree of any node in the graph is $\Omega(n^{\alpha/\log \log n})$ for any $\alpha > 0$. Our analysis of Phase I for arbitrary graphs follows the analysis of Phase II in [53].

Distributed algorithms with the underlying graph equipped with sense of direction have been studied in [136, 44]. Sense of direction is a similar notion to that of orientation on edges. Singh [136] shows that leader election in an n-node complete graph equipped with sense of direction can be performed in a distributed setting via exchange of $O(n)$ messages. In [44], the authors show that having sense of direction reduces the communication complexity of several distributed graph algorithms such as leader election, spanning tree construction, and depth-first traversal.

4.1.3 Our Results

We start by investigating the bit complexity of distributed vertex coloring algorithms. We first show that the bit complexity of the coloring problem is $\Omega(\log n)$ for an non-oriented n-node cycle graph. That is, any distributed algorithm in which all the nodes start in the same state and know only about n and Δ apart from their neighbors needs $\Omega(\log n)$ rounds with high probability to arrive

at a proper coloring using any finite number of colors. We then show that when the edges in the cycle graph are provided with an orientation, then the bit complexity of distributed vertex coloring algorithms is $\Omega(\sqrt{\log n})$, with high probability, when using any finite number of colors. This leads us to the question whether matching upper bounds can be shown for coloring oriented graphs.

We start with the case of constant degree $\sqrt{\log n}$–acyclic oriented graphs and present an algorithm to obtain a $(\Delta+1)$–coloring with a bit complexity of $O(\sqrt{\log n})$ with high probability. Thus, we show the following theorem.

Theorem 4.1.2 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, if Δ is a constant, a $(\Delta + 1)$–vertex coloring of G can be obtained in $O(\sqrt{\log n})$ bit rounds, with high probability.*

The above theorem directly implies that oriented cycle graphs can be 3–colored in $O(\sqrt{\log n})$ bit rounds. Additionally, for the case of constant degree graphs we can also arrive at a local coloring where the color of every node u is in $[d_u + 1]$. In our algorithm for constant degree oriented graphs, it suffices that nodes know only their local degree.

We then extend our algorithm and analysis to the case of arbitrary $\sqrt{\log n}$–acyclic oriented graphs with maximum degree Δ. Our main result is a distributed $(1 + \epsilon)\Delta$–coloring algorithm for arbitrary $\sqrt{\log n}$–acyclic oriented graphs of maximum degree Δ. Our algorithm has a bit complexity of $O(\log \Delta) + \tilde{O}(\sqrt{\log n})$. By $g(n) = \tilde{O}(f(n))$ we mean $g(n) = O(f(n)\text{polylog}(f(n)))$. Specifically, we prove the following theorem.

Theorem 4.1.3 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, a $(1 + \epsilon)\Delta$–vertex coloring of G, for any constant $\epsilon > 0$, can be obtained in $O(\log \Delta) + \tilde{O}(\sqrt{\log n})$ bit rounds, with high probability.*

By further tightening the analysis, we show that the bit complexity can be reduced to $O(\log \Delta + \sqrt{\log n \log\log n})$, with high probability, for $\sqrt{\log n}$–acyclic oriented graphs with $\Delta \geq \log n$. For the case of arbitrary $\sqrt{\log n}$–acyclic oriented graphs, our algorithm and analysis can be modified easily to get a local coloring such that every node u gets a color in $[(1+\epsilon)d_u]$.

4.1.4 Summary of our approach

We now provide a brief summary of our basic approach. Our approach has the same flavor as existing distributed vertex coloring algorithms [103, 64]. Given any $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of constant degree Δ, the algorithm for $(\Delta + 1)$–coloring proceeds as follows. Communication proceeds in rounds and in each round each yet uncolored node v chooses a color c_v among the available colors in $[\Delta + 1]$ independently and uniformly at random. Node v then communicates this color choice to all of its uncolored neighbors. If a node chooses a color that is in conflict with any of the choices of its neighbors, the conflict resolution rule specifies the course of action. In the algorithm of Luby[103], Johansson[64], and most other works, the conflict resolution rule is that uncolored nodes in conflict remain uncolored and have to try again in subsequent rounds. The conflict resolution rule we use is based on the orientation on the edges as explained in Section 4.1.1. Our algorithm is thus similar to the existing distributed vertex coloring algorithms [103, 64] except for the conflict resolution rule.

In our analysis, after $O(\sqrt{\log n})$ rounds we arrive at the situation where connected components of uncolored nodes only have simple oriented paths of length less than $\sqrt{\log n}$, with high probability. Coupled with the $\sqrt{\log n}$–acyclic orientation, it can be shown that the nodes in each such connected component can be organized into less than $\sqrt{\log n}$ layers. The layering has the property that all the oriented edges are from a node in a lower-numbered layer to a node in a higher numbered layer. This property of the layering guarantees a successful coloring of all remaining uncolored nodes in less

than $\sqrt{\log n}$ rounds. This gives us the result for constant degree oriented graphs. (Theorem 4.1.2). To arrive at the bit complexity for arbitrary graphs, (Theorem 4.1.3) we need a few additional tricks as our analysis shows.

4.1.5 Organization of the Chapter

The rest of the chapter is organized as follows. In Section 4.2 we establish the lower bound results. In Section 4.3, we present and analyze our algorithm for $(\Delta + 1)$–coloring constant degree oriented graphs. This will serve as a base for the $(1 + \epsilon)\Delta$–coloring algorithm for arbitrary oriented graphs of maximum degree Δ for any constant $\epsilon > 0$, in Section 4.4.

4.2 Lower Bounds

In this section we establish lower bounds on the bit complexity of finding a proper vertex coloring. Recall that a Las Vegas algorithm is a randomized algorithm that always produces a correct result, with the only variation being its runtime. First, we prove a lower bound for non-oriented graphs, and then we prove a lower bound for oriented graphs. Notice that both bounds hold for any finite number of colors.

Theorem 4.2.1 *For every Las Vegas algorithm A there is an infinite family of non-oriented graphs $\mathcal{G}$ s.t. A has a bit complexity of at least $\Omega(\log n)$ on $\mathcal{G}$, with high probability, to compute a proper vertex coloring.*

Proof. Consider the cycle of n nodes, and let $S_\ell = (u_\ell, \ldots, u_1, v_1, \ldots, v_\ell)$ be the set of nodes along a path of length 2ℓ of the cycle. Initially, every node in S_ℓ is in the same state s_0, with the only difference that for every $i \in \{1, \ldots, \ell - 1\}$, u_i considers its left connection to go to u_{i+1} whereas v_i considers its left connection to go to v_{i+1}. (Notice that the cycle is non-oriented, so we can

choose any orientation we want for the individual nodes.) Associated with s_0 is a fixed probability distribution $P_\epsilon = (p_x^\epsilon)_{x\in\{-,0,1\}}$ for sending bit x along the right edge, where "$-$" represents the case that no bit is sent and ϵ represents the empty history. Since P_ϵ has only three probability values, there must be an x_0 with $p_{x_0}^\epsilon \geq 1/3$. Let E_1 be the event that nodes u_1 and v_1 choose that option. Then u_1 and v_1 receive the same information from their right neighbor. Let $P_y = (p_x^y)_{x\in\{-,0,1\}}$ be the probability distribution for sending bit x along the right edge in the second round given that bit y was received from the right edge in the first round. Then P_{x_0} applies to u_1 and v_1. Since P_{x_0} has only three probability values, there must be an x_1 with $p_{x_1}^{x_0} \geq 1/3$. Let E_2 be the event that nodes u_1 and v_1 choose that option. Then u_1 and v_1 again receive the same information from their right neighbor.

Continuing with this argumentation, it follows that there are events $E_1, \ldots, E_\ell$ with E_i having a probability of at least $1/3$ for all i so that u_1 and v_1 have received the same information from their right neighbors. Algorithm A cannot terminate in this case because in this case the same probability distribution for choosing a color applies to u_1 and v_1, and hence, the probability that u_1 and v_1 choose the same color is non-zero.

When choosing $\ell = \log_3(n/2\log^2 n)$, the probability for the events $E_1, \ldots, E_\ell$ to occur is at least $\left(\frac{1}{3}\right)^{\log_3(n/2\log^2 n)} = \frac{2\log^2 n}{n}$. Moreover, notice that $E_1, \ldots, E_\ell$ only depend on the nodes in S_ℓ because information can only travel a distance of ℓ edges in ℓ rounds. Hence, we can partition the n-node cycle into $n/2\ell$ many sequences S where each sequence has a probability of at least $\frac{2\log^2 n}{n}$ of running into the events $E_1, \ldots, E_\ell$ that is independent of the other sequences. Thus, the probability that all node sequences can avoid the event sequence $E_1, \ldots, E_\ell$, which is necessary for A to terminate, is at most $\left(1 - \frac{2\log^2 n}{n}\right)^{n/2\ell} \leq 1/n$, which implies that A needs $\Omega(\log n)$ bit-rounds, with high probability, to finish. □

Theorem 4.2.2 *For every Las Vegas algorithm A there is an infinite family of oriented graphs $\mathcal{G}$ s.t. A has a bit complexity of at least $\Omega(\sqrt{\log n})$ on $\mathcal{G}$, with high probability, to compute a proper vertex coloring.*

Proof. Consider the cycle of n nodes in which all the edges are oriented in the same direction. Let $S_\ell = (u_\ell, \ldots, u_1, v_1, \ldots, v_\ell)$ be the set of nodes along a path of length 2ℓ of the cycle. Initially, every node in S_ℓ is in the same state s_0. Associated with s_0 is a fixed probability distribution $P_0 = (p^0_{x,y})_{x,y\in\{-,0,1\}}$ for sending bit x along the left edge and bit y along the right edge, where "$-$" represents the case that no bit is sent. Since P_0 has only nine probability values, there must be an x_0 and y_0 with $p^0_{x_0,y_0} \geq 1/9$. Let E_1 be the event that all nodes in S_ℓ choose that option. Then all nodes in $S_{\ell-1} = (u_{\ell-1}, \ldots, u_1, v_1, \ldots, v_{\ell-1})$ receive the same information and must therefore be in the same state s_1. Associated with s_1 is a fixed probability distribution $P_1 = (p^1_{x,y})_{x,y\in\{-,0,1\}}$ for sending bit x along the left edge and bit y along the right edge. Since P_1 has only nine probability values, there must be an x_1 and y_1 with $p^1_{x_1,y_1} \geq 1/9$. Let E_2 be the event that all nodes in $S_{\ell-1}$ choose that option. Then all nodes in $S_{\ell-2}$ receive the same information and must therefore be in the same state s_2.

Continuing with this argumentation, it follows that there are events $E_1, \ldots, E_\ell$ with E_i having a probability of at least $(1/9)^{2(\ell-i+1)}$ for all i so that all nodes in $S_{\ell-i}$ are in the same state s_i. Since these nodes are neighbors, algorithm A cannot terminate within ℓ bit exchanges if $E_1, \ldots, E_\ell$ are true because whatever probability distribution A chooses on the colors, the probability that two neighboring nodes choose the same color is non-zero, which would violate the assumption that A is a Las Vegas algorithm.

The probability that $E_1, \ldots, E_\ell$ are true is at least $\left(\frac{1}{9}\right)^{\sum_{i=1}^{\ell} 2(\ell-i+1)} \geq \left(\frac{1}{9}\right)^{\ell^2/2}$ and when choosing $\ell = \sqrt{2\log_9(n/2\log^2 n)}$, this results in a probability of at least $(2\log^2 n)/n$. Moreover,

notice that $E_1, \ldots, E_\ell$ only depend on the nodes in S_ℓ because information can only travel a distance of ℓ edges in ℓ rounds. Hence, we can partition the n-node cycle into $n/2\ell$ many sequences S where each sequence has a probability of at least $\frac{2\log^2 n}{n}$ of running into the events $E_1, \ldots, E_\ell$ that is independent of the other sequences. Hence, the probability that all node sequences can avoid the event sequence $E_1, \ldots, E_\ell$, which is necessary for A to terminate, is at most $\left(1 - \frac{2\log^2 n}{n}\right)^{n/2\ell} \leq 1/n$, which implies that A needs $\Omega(\sqrt{\log n})$ bit-rounds, with high probability, to finish. □

Thus, oriented graphs appear to be easier to color than non-oriented graphs. In the next section we show that this is indeed the case by providing a matching upper bound for constant-degree graphs.

4.3 Upper Bound for Constant Degree Oriented Graphs

In this section we present and analyze the algorithm for $(\Delta + 1)$–coloring constant degree oriented graphs. This demonstrates the efficacy of using orientation in vertex coloring algorithms. We defer the case of arbitrary oriented graphs to Section 4.4 as it requires more complicated arguments than for constant degree graphs.

The algorithm for vertex coloring constant degree oriented graphs is given in Figure 4.3. In the algorithm, the parameter C_u refers to the number of colors used in the coloring by node u. Each node executes the algorithm Color-Random until it gets colored.

We analyze algorithm Color-Random for constant degree oriented graphs with a $\sqrt{\log n}$–acyclic orientation and show that algorithm Color-Random can be used to obtain a $(\Delta+1)$–coloring with a bit complexity of $O(\sqrt{\log n})$. The reduction in the bit complexity from $\Omega(\log n)$ (due to Theorem 4.2.1) to $O(\sqrt{\log n})$ comes from the fact that once every simple oriented path of length $\sqrt{\log n}$ has at least one colored node, the $\sqrt{\log n}$–acyclic orientation guarantees us connected components

Algorithm Color-Random(C_u)
While u is not colored do
1. Node u chooses a color c_u from the available colors in $[C_u]$ uniformly at random.
2. Node u communicates its choice c_u, from step 1, to all of its uncolored neighbors that have a lower priority over u, i.e. to nodes v such that $u \rightarrow v$.
3. If node u does not receive a message from any of its neighbors w with $w \rightarrow u$ and $c_w = c_u$, then node u gets colored with color c_u. Otherwise node u remains uncolored.
4. If u is colored during step 3 of the current round, then u informs all of its uncolored neighbors about the color of u.
5. Node u updates the list of available colors according to colors taken up by u's neighbors.

Figure 4.3: Coloring constant degree oriented graphs by random choices.

of uncolored nodes where each such component only has simple oriented paths of length less than $\sqrt{\log n}$. The $\sqrt{\log n}$-acyclicity of the orientation allows us to finish in a further $\sqrt{\log n}$ rounds.

Theorem 4.3.1 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, if Δ is a constant, a $(\Delta + 1)$–vertex coloring of G can be obtained in $O(\sqrt{\log n})$ bit rounds, with high probability.*

Proof. The analysis below cuts the time into two phases. Phase I ends once every simple oriented path of length $\ell = \sqrt{\log n}$ has at least one colored node, and phase II ends once all nodes are colored. We show that phase I takes at most $r = 4\sqrt{\log n}$ rounds, with high probability. For Phase II, the proof uses the $\sqrt{\log n}$–acyclic orientation to argue that a further $\sqrt{\log n}$ rounds suffice to color all nodes. For simplicity, we set $C_u = 2\Delta$ for every node u, but the analysis works, with minor modifications, for $C_u = \Delta + 1$, as long as Δ is a constant.

Consider any simple oriented path P of length ℓ. For any node $u \in P$ with C'_u remaining colors and d'_u remaining uncolored neighbors, the probability that it chooses a color that is identical to the choice of any of its uncolored neighbors is at most $\sum_{j=1}^{d'_u} 1/C'_u \leq d'_u/(2\Delta - (d_u - d'_u)) \leq 1/2$ as $C'_u = 2\Delta - (d_u - d'_u)$ and $d'_u \leq d_u$.

For any $i \geq 1$, denote by $E_{P,i}$ the event that all nodes in P have a color conflict in round i.

Since each node chooses the color independently and uniformly at random, and P is oriented, one can identify a distinct witness for each color conflict so as to upper bound $\Pr[E_{P,i} \mid \cap_{j=0}^{i-1} E_{P,j}]$ as $\Pr[E_{P,i} \mid \cap_{j=0}^{i-1} E_{P,j}] \leq (1/2)^{\ell}$.

Denote by E_P the event that the event $E_{P,i}$ occurs for r consecutive rounds. Then,

$$\Pr[E_P] = \Pr[\bigcap_{i=1}^{r} E_{P,i}] = \Pi_{i=1}^{r} \Pr[E_{P,i} \mid \cap_{j=1}^{i-1} E_{P,j}] \leq (1/2)^{\ell r}.$$

Let E denote the event that for some simple oriented path P the event E_P occurs. The number of simple oriented paths of length ℓ is at most $n\Delta^{\ell}$ by choosing the first vertex from n available choices and choosing each of the next ℓ vertices from the at most Δ available choices. Thus,

$$\Pr[E] = \Pr[\bigcup_{P} E_P] \leq n\Delta^{\ell} \Pr[E_P] \leq 1/n^2.$$

for the above value of r since $\Delta = O(1)$. This completes Phase I of the analysis.

Consider connected components of uncolored nodes. At the end of Phase I, since any simple oriented path of length ℓ has at least one colored node, each such component only has simple oriented paths of length less than ℓ, with high probability. Moreover, the input graph does not have oriented cycles of length less than $\sqrt{\log n}$ which implies that each such component can be organized into less than $\sqrt{\log n}$ layers with oriented edges going only from a node in a lower-numbered layer to a node in a higher numbered layer. This layering can be achieved by the following process. Nodes with no superiors are assigned to layer 0. After removing these nodes, nodes in the rest of the component with no superiors are assigned to layer 1, and so on, until there are no nodes left. Such a procedure terminates in less than $\sqrt{\log n}$ rounds, implying that the layer number of any node is less than $\sqrt{\log n}$. Otherwise, there must exist either a simple oriented path of length at

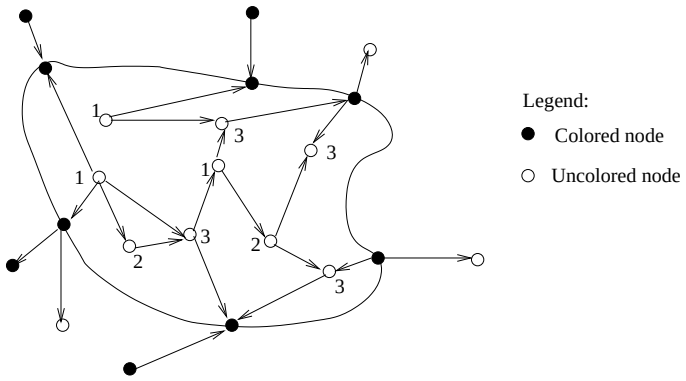


Figure 4.4: Connected component of uncolored nodes. The number at the uncolored nodes within the connected component gives the layer number they belong to.

least $\sqrt{\log n}$ or an oriented cycle of length less than $\sqrt{\log n}$. Both of these conditions result in a contradiction and hence the layering process must terminate in less than $\sqrt{\log n}$ rounds. Figure 4.4 shows an example along with the assignment of nodes to layers.

Now, in Phase II, during every round the uncolored nodes assigned to the lowest layer number presently get colored as the nodes assigned to the lowest layer can always retain their color choice from Step 1. This implies that Phase II can finish in less than $\sqrt{\log n}$ rounds.

Since in each round each uncolored node has to exchange $O(\log \Delta) = O(1)$ bits, the bit complexity of the algorithm Color-Random is $O(\sqrt{\log n})$. □

We note that the same proof also holds for 3–coloring cycle graphs, with any orientation, with minimal changes. Coupled with the lower bound result in Theorem 4.2.2, our analysis for the case of constant degree graphs is tight with respect to the bit complexity, up to constant factors. The algorithm and the analysis can be modified easily to achieve a local coloring also.

4.4 Upper Bound for Arbitrary Oriented Graphs

In this section we describe and analyze our algorithm for vertex coloring an arbitrary $\sqrt{\log n}$–acyclic oriented graph G using $(1+\epsilon)\Delta$ colors for any constant $\epsilon > 0$.

Our algorithm and the analysis in this case requires more tools than that for constant degree graphs while having the same flavor. Theorem 4.3.1 fails to hold once the degree of the input graph is bounded away from any constant. Graphs below logarithmic degree, but bounded away from a constant, pose additional problems as graphs with degree below a certain threshold are not easily amenable to nice probabilistic bounds. In many papers, for example [53, 116, 37], this problem was overcome by assuming that the number of colors available is $\max\{(1+\epsilon)\Delta, \log n\}$ so that sub-logarithmic degree graphs are colored with $\log n$ colors. We instead take the approach of coloring with $(1+\epsilon)\Delta$ colors as coloring with few colors is more appealing when vertex coloring is used as a sub-routine in other higher order tasks.

To arrive at our result, we proceed in stages. Based on techniques from [53], we first show how to arrive at a bit complexity of $\tilde{O}(\log\Delta + \sqrt{\log n})$. Later, using advanced techniques, we show how to arrive at a bit complexity of $O(\log\Delta) + \tilde{O}(\sqrt{\log n})$. Finally, for graphs with $\Delta \geq \log n$, we show how to arrive at a bit complexity of $O(\log\Delta + \sqrt{\log n \log\log n})$.

Our algorithm for any node u is presented in Figure 4.5. The parameter C_u denotes the number of colors each vertex u can choose from. Each node runs the algorithm in Figure 4.5 while it remains uncolored.

We now provide a summary of our analysis of algorithm Color. Our analysis cuts time into two phases. In the first phase we show that for any vertex the number of uncolored neighbors reduces to at most $c_2 \log n$ for a constant c_2, in $O(\log\log n)$ rounds, with high probability. In the second phase we first show that the graph can be decomposed into connected components of uncolored

Algorithm Color(C_u)
Phase I
1. Set $C_u := c_1\Delta$ for a constant $c_1 \geq 3$.
2. While $d_u \geq c_2 \log n$ for a constant c_2 do
   3. Use Algorithm Color-Random(C_u).
Phase II
4. Set $C_u := \min\{2c_2 \log n, 2d_u\}$.
5. Use Algorithm Color-Random(C_u).

Figure 4.5: Algorithm for any node u.

nodes such that each such connected component only has simple oriented paths of length less than $\sqrt{\log n}$, with high probability. The analysis then proceeds to show that all the nodes can be colored in a further $\sqrt{\log n}$ rounds.

In the algorithm and the analysis we also set $C_u = c_1\Delta$ for a constant $c_1 \geq 3$, for every node u, for the sake of simplicity. Using techniques from [53], it is possible to extend the following analysis to use only $(1+\epsilon)\Delta$ colors, for any constant $\epsilon > 0$.

4.4.1 Analysis for Phase I

In this phase, we show that the number of uncolored neighbors of any node u reduces in a double-exponential fashion, (i.e., in $O(\log \log n)$ rounds) to $c_2 \log n$. This analysis has strong connections to occupancy problems [109, Problem 3.4],[13], and the edge coloring algorithm of [53].

Let $d_u(i), N_u(i), C_u(i)$ refer to the number of uncolored neighbors, the set of uncolored neighbors, and the size of the color palette of node u respectively, at the beginning of round i. Also, let $\hat{d}(i) = \max_u d_u(i)$.

Lemma 4.4.1 *If $d_u(1) \geq c_2 \log n$ then $d_u(c' \log \log n) \leq c_2 \log n$, with high probability for some constant $c' \geq 1$.*

Proof. The intuition behind the proof is that at the end of every round, the number of remaining uncolored neighbors decreases double-exponentially.

During round i the probability that an uncolored node u fails to get colored can be computed as: $P_u(i) := \Pr[u \text{ does not get colored during round } i] \leq \sum_{j=1}^{d_u(i)} \frac{1}{C_u(i)} \leq \frac{\hat{d}(i)}{\alpha\Delta}$, as, for c_1 sufficiently large, it holds that $C_u(i) \geq \alpha\Delta$ for $\alpha = c_1 - 1$.

The expected number of neighbors of u that are still uncolored after round i is, $\mathrm{E}[d_u(i+1)] = \sum_{v \in N_u(i)} P_v(i) \leq \hat{d}(i)^2/\alpha\Delta$.

Consider the following recurrence relation between $\hat{d}(i+1)$ and $\hat{d}(i)$ for a constant c''.

$$\hat{d}(i+1) \leq \frac{\hat{d}^2(i)}{\alpha\Delta} + \sqrt{c''\hat{d}(i)\log n}. \tag{4.1}$$

Using a large deviation bound [52, 53], it can be shown that $d_u(i+1)$ exceeds its expected value by more than $\sqrt{c''\hat{d}(i)\log n}$ with probability less than n^{-2} for some constant c''. Thus, it holds that $d_u(i+1) \leq \hat{d}(i+1)$ w.h.p., for all nodes u.

The recurrence relation in Equation (4.1) can be solved as follows (cf. [53]). Let $\hat{d}^2(i)/\alpha\Delta$ dominate the second term for r_1 rounds. The value of r_1 then satisfies $\hat{d}^2(i)/\alpha\Delta \leq 2\sqrt{c''\hat{d}(i)\log n}$. For this we require that $\hat{d}^3(r_1) \leq 4c''\alpha^2\Delta^2\log n$.

So for r_1 rounds, we have:

$$\hat{d}(i+1) \leq 3\hat{d}^2(i)/\alpha\Delta \tag{4.2}$$

Using Equation (4.2), it follows that

$$\hat{d}(r_1) \leq (3/\alpha)^{2^{r_1}-1}\Delta$$

Hence, we require that:

$$\left((3/\alpha)^{2^{r_1-1}}\Delta\right)^3 \leq 4c''\alpha^2\Delta^2\log n$$

which results in a value of $r_1 = O(\log\log n)$.

From this point on, it holds that

$$\hat{d}(i+1) \leq 5\sqrt{c''\hat{d}(i)\log n} \tag{4.3}$$

Solving Equation (4.3) for a value of r_2 so that $\hat{d}(r_2) \leq c_2\log n$ results in $r_2 = O(\log\log n)$.

Thus after $i^* = r_1 + r_2 = O(\log\log n)$ rounds, we have for any node u, $d_u(i^*) \leq \hat{d}(i^*) \leq c_2\log n$. □

Thus, at the end of $O(\log\log n)$ rounds of the algorithm, the number of uncolored neighbors for every node is at most $c_2\log n$. This completes Phase I of the analysis.

4.4.2 Analysis for Phase II

The analysis in this phase consists of two sub-phases. In sub-phase II(a), we argue that along any simple oriented path of length $\sqrt{\log n}$ there exists at least one colored node, with high probability. In the second sub-phase we show that all the remaining uncolored nodes successfully get colored within $\sqrt{\log n}$ rounds. Notice that since the number of uncolored neighbors of any node at the beginning of this phase is at most $c_2\log n$, nodes can use a color palette of size $\min\{2c_2\log n, 2d_u\}$ for this phase as shown in the algorithm in Figure 4.5.

Analysis for Phase II(a)

We now establish the following lemma which shows that every simple oriented path of length $\sqrt{\log n}$ has at least one colored node after $O(\sqrt{\log n})$ rounds, with high probability. Let Δ_* denote the maximum number of uncolored neighbors for any node u. After phase I, it holds that $\Delta_* \leq c_2 \log n$.

Lemma 4.4.2 *For arbitrary $\sqrt{\log n}$–acyclic oriented graphs G at the end of $O(\sqrt{\log n})$ rounds, any simple oriented path of length $\ell = \sqrt{\log n}$ will have at least one colored node, with high probability. Further, the bit complexity of this phase is $O(\sqrt{\log n} \log\log n)$.*

Proof. The proof of this lemma is similar to the proof of Phase I of Theorem 4.3.1. Consider any simple oriented path P of length $\ell = \sqrt{\log n}$. Let $E_{P,i}$ denote the event that all the nodes in P are in a color conflict during a given round i. Then, along the lines of Phase I of Theorem 4.3.1, it holds that $\Pr[E_{P,i} \mid \cap_{j=1}^{i-1} E_{P,j}] \leq (1/2)^{\ell}$. Define E_P to be the event that the event $E_{P,i}$ occurs for $r = 4c_2\sqrt{\log n}$ consecutive rounds. Then, it holds that

$$\Pr[E_P] = \Pr[\cap_{i=1}^{r} E_{P,i}] = \Pi_{i=1}^{r} \Pr[E_{P,i} \mid \cap_{j=1}^{i-1} E_{P,j}] \leq (1/2)^{\ell r}.$$

Let E denote the event that there exists a path P for which the event E_P occurs. Since the number of simple oriented paths of length P is at most $n\Delta_*^{\ell}$,

$$\Pr[E] \leq \sum_{P} \Pr[E_P] \leq n\Delta_*^{\ell} \left(\frac{1}{2}\right)^{r\ell}.$$

As $r = 4c_2\sqrt{\log n}$ and $\ell = \sqrt{\log n}$ and $\Delta_* \leq c_2 \log n$, the above probability is polynomially small.

The bit complexity of this phase is $O(\sqrt{\log n} \log\log n)$ as in each round each uncolored exchanges $O(\log\log n)$ bits. □

Analysis for Phase II(b)

Consider connected components of uncolored nodes. At the end of Phase II(a), since any simple oriented path of length $\sqrt{\log n}$ has at least one colored node, w.h.p., it holds that each such component has only simple oriented paths of length less than $\sqrt{\log n}$ with high probability. Also, the input graph G does not have oriented cycles of length less than $\sqrt{\log n}$. For Phase II(b), we show the following lemma.

Lemma 4.4.3 *In Phase II(b), after less than $\sqrt{\log n}$ rounds, all nodes in G are colored properly. Further, the bit complexity of Phase II(b) is $O(\sqrt{\log n}\log\log n)$.*

Proof. The proof of this lemma is similar to that of the proof of Phase II in Theorem 4.3.1. It can be shown that any connected component of uncolored nodes only has simple oriented paths of length less than ℓ, with high probability. Moreover, the input graph does not have oriented cycles of length less than $\sqrt{\log n}$ which implies that each such component can be organized into less than $\sqrt{\log n}$ layers with the oriented edges going only from a node in a lower-numbered layer to a node in a higher numbered layer.

Using the layering, then it can be shown that during each further round, at least one node gets colored successfully. Thus, this phase requires less than $\sqrt{\log n}$ rounds and each node exchanges $O(\log\log n)$ bits during every round of this phase. Thus the bit complexity of this phase is $O(\sqrt{\log n}\log\log n)$. □

From the above discussion, the following theorem holds.

Theorem 4.4.4 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, for any constant $\epsilon > 0$, a $(1+\epsilon)\Delta$–vertex coloring of G can be obtained in $\tilde{O}(\log\Delta + \sqrt{\log n})$ bit rounds, with high probability.*

Proof. Phase I has a bit complexity of $O(\log \Delta \log\log n)$ as for $O(\log\log n)$ rounds, each node exchanges $O(\log \Delta)$ bits. For Phase II, the bit complexity is $O(\sqrt{\log n}\log\log n)$. Adding the bit complexity of both the phases, we arrive at the theorem. □

4.4.3 Further Improvements

In this section, we show that the bit complexity of Phase I can be reduced to $O(\log \Delta + \log\log n)$ thereby reducing the bit complexity of the algorithm for arbitrary $\sqrt{\log n}$–acyclic oriented graphs to $O(\log \Delta) + \tilde{O}(\sqrt{\log n})$. We then show a tighter analysis for *high* degree graphs to arrive at a bit complexity of $O(\log \Delta + \sqrt{\log n \log\log n})$. By *high* degree graphs, we mean graphs with $\Delta \geq \log n$.

Improvements to the Analysis of Phase I

The tightness of the analysis stems from a gradual reduction in the number of colors during Phase I. This results in savings in the bit complexity of Phase I. For this purpose, Phase I is divided into sub-phases as follows. For $j \geq 1$, sub-phase j starts when the number of uncolored neighbors of u is at most $D_u^{(j)}$, where $D_u^{(j)}$ acts a threshold on the number of uncolored neighbors of node u. $D_u^{(j)}$ is defined as $D_u^{(1)} = d_u$ and $D_u^{(j)} = \sqrt{D_u^{(j-1)}}$ for $j \geq 2$. Let $C_u^{(j)}$ denote the size of the color palette of node u during sub-phase j. At the beginning of sub-phase j node u reduces the size of its color palette so that $C_u^{(j)} = c_1\sqrt{C_u^{(j-1)}}$ with $C_u^{(1)} = c_1\Delta$.

This effectively reduces the number of bits required to be sent in each sub-phase by a factor of 2 but the proof of Lemma 4.4.1 holds with minimal changes. Thus, in Phase I, it can be seen that over the $O(\log\log n)$ sub-phases the number of bits each node u sends is at most

$$\sum_{j=1}^{O(\log\log n)} \log C_u^{(j)} \leq \sum_{j=1}^{O(\log\log n)} 2\log c_1 + ((\log\Delta)/2^j) \;=\; O(\log\log n + \log\Delta).$$

Algorithm PhaseI
1. $D_u := \sqrt{d_u}, C_u := c_1\Delta$.
2. While $d_u \geq c_2 \log n$ do
   3. Run Algorithm Color–Random(C_u).
   4. If $d_u \leq D_u$ then
      5. $D_u := \sqrt{D_u},\ C_u := c_1\sqrt{C_u}$
end-while.

Figure 4.6: Improved algorithm for Phase I.

Thus, the bit complexity for Phase I reduces to $O(\log\log n + \log\Delta)$.

The modified algorithm for Phase I for node u is described in Figure 4.6. Using the tighter analysis for Phase I and Lemmata 4.4.2–4.4.3, we arrive at the following theorem.

Theorem 4.4.5 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree Δ, a $(1+\epsilon)\Delta$–vertex coloring of G for any constant $\epsilon > 0$, can be obtained in $O(\log\Delta) + \tilde{O}(\sqrt{\log n})$ bit rounds, with high probability.*

Improvements to Phase II

For the case of high degree graphs, we now show how to reduce the bit complexity of Phase II to $O(\sqrt{\log n \log\log n})$. The algorithm for Phase II remains the same as shown in Figure 4.5. The analysis of Phase II now consists of 3 sub-phases.

In sub-phase II(a), we show that the number of uncolored neighbors of any node decreases to $O(\sqrt{\log n}\log\log n)$ after $O(\sqrt{\log n}/\log\log n)$ rounds with high probability. In sub-phase II(b) we then show that every simple oriented path of length $\sqrt{\log n/\log\log n}$ has at least one colored node, with high probability, after $O(\sqrt{\log n/\log\log n})$ rounds. Finally, in sub-phase II(c), we show that every node can be colored after a further $O(\sqrt{\log n/\log\log n})$ rounds. In this phase, every node can use a color palette of size $2c_2\log n$.

Analysis for Phase II(a)

For sub-phase II(a), we show the following lemma.

Lemma 4.4.6 *In Phase II(a), in $O(\frac{\sqrt{\log n}}{\log\log n})$ rounds, the number of uncolored neighbors of any node decreases to $\sqrt{\log n}\log\log n$, with high probability. Further, the bit complexity of this sub-phase is $O(\sqrt{\log n})$.*

Proof. Consider any node u. At the end of phase I, it holds that $d_u \leq c_2 \log n$, with high probability. Since the number of colors used by u is $2c_2 \log n$, it also holds that

$$\Pr[\text{node } u \text{ fails to get colored in a given round}] \leq 1/2.$$

Consider any subset A of the uncolored neighbors of u. Let E_A denote the event that all the nodes in A remain uncolored after $r = \frac{4c_2\sqrt{\log n}}{\log\log n}$ consecutive rounds. Then, it holds that

$$\Pr[E_A] \leq (1/2)^{r|A|(1-1/\sqrt{\log n})}$$

using the orientation and the witnessing scheme of Theorem 4.3.1. Notice that as we are guaranteed of $\sqrt{\log n}$–acyclicity we can find a set of at least $|A|(1-1/\sqrt{\log n})$ nodes in conflict with a distinct witness for each color conflict.

Let $E_{u,s}$ denote the event that for node u, there exists a set of s uncolored neighbors at the end of r rounds. Then,

$$\Pr[E_{u,s}] = \Pr[\bigcup_{A\subseteq N_u, |A|=s} E_A] \leq \bigcup_{A\subseteq N_u, |A|=s} \Pr[E_A] = \binom{d_u}{s}\left(\frac{1}{2}\right)^{rs(1-1/\sqrt{\log n})}.$$

Denote by E_u the event that for node u there exist more than $\sqrt{\log n}\log\log n$ uncolored neighbors. Using Boole's inequality,

$$\Pr[E_u] \leq \sum_{s=\sqrt{\log n}\log\log n}^{d_u} \Pr[E_{u,s}] \leq \sum_{s=\sqrt{\log n}\log\log n}^{d_u} \binom{d_u}{s} \cdot (1/2)^{rs(1-1/\sqrt{\log n})} \leq \frac{1}{n^2}.$$

as $rs \geq 4c_2 \log n$ and $d_u \leq c_2 \log n$. Now, denote by E the event that for some node u, the event E_u occurs. Then, $\Pr[E] = \Pr[\bigcup_{u \in V} E_u] \leq 1/n$. Thus, the number of uncolored neighbors of any node decreases to $\sqrt{\log n}\log\log n$ with high probability after $4\sqrt{\log n}/\log\log n$ rounds.

During this phase, each uncolored node exchanges $O(\log\log n)$ bits in each round as the palette size is $2c_2 \log n$. Thus, the bit complexity of this sub-phase is $O(\sqrt{\log n})$. □

Analysis for Phase II(b)

At the end of sub-phase II(a), it holds that the number of uncolored neighbors of any node u is at most $\sqrt{\log n}\log\log n$. Recall that $\Delta_* = \max_u d_u$. After sub-phase II(a), it holds that $\Delta_* \leq \sqrt{\log n}\log\log n$.

Lemma 4.4.7 *In Phase II(b), in $16\sqrt{\log n/\log\log n}$ rounds, in every simple oriented path of length $\sqrt{\log n/\log\log n}$ there is at least one node that gets colored, with high probability. Further, the bit complexity of this sub-phase is $O(\sqrt{\log n \log\log n})$.*

Proof. Consider any simple oriented path P of uncolored nodes of length $\ell = \sqrt{\log n/\log\log n}$. Denote by E_P the event that no node in P gets colored in $16\sqrt{\log n/\log\log n}$ rounds. Since P is oriented and the choices of each node are independent, using a witnessing scheme similar to that in the proof of Theorem 4.3.1, it holds that:

$$\Pr[E_P] \leq \left(\frac{\sqrt{\log n}\log\log n}{2c_2 \log n}\right)^{\ell \cdot 16\sqrt{\log n/\log\log n}} \leq \left(\frac{\log\log n}{2c_2\sqrt{\log n}}\right)^{\frac{16\log n}{\log\log n}} \leq \frac{1}{n^4},$$

if n is sufficiently large. In the above, the first inequality holds since the number of uncolored neighbors is $\sqrt{\log n}\log\log n$ and the number of colors that u can choose from is $2c_2 \log n$.

Let E denote the event that there exists a simple oriented path P of length ℓ such that for path P, the event E_P occurs. The number of simple oriented paths of length $\ell = \sqrt{\log n/\log\log n}$ is at most $n \cdot \Delta_*^\ell$. Thus,

$$\Pr[E] = \Pr[\bigcup_P E_P] \leq \sum_{j=1}^{n\Delta_*^\ell} 1/n^4 \leq \Delta_*^\ell/n^3.$$

The above probability is polynomially small since $\Delta_* \leq \sqrt{\log n}\log\log n$. Thus, along any simple oriented path of length $\sqrt{\log n/\log\log n}$, at least one node gets colored with high probability at the end of $16\sqrt{\log n/\log\log n}$ rounds.

The bit complexity of this sub-phase is easily seen to be $O(\sqrt{\log n \log\log n})$ as in each round, each uncolored node exchanges $O(\log\log n)$ bits. □

This completes the analysis for Phase II(b). In Phase II(c), using arguments similar to that of Lemma 4.4.3, it can be shown that in a further $\sqrt{\log n/\log\log n}$ rounds, every node gets colored, with high probability. The bit complexity of Phase II(c) is $O(\sqrt{\log n \log\log n})$. Putting together everything, we arrive at the following theorem.

Theorem 4.4.8 *Given a $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ of maximum degree $\Delta \geq \log n$, for any constant $\epsilon > 0$, a $(1+\epsilon)\Delta$–vertex coloring of G can be obtained in $O(\log\Delta + \sqrt{\log n \log\log n})$ bit rounds, with high probability.*

Notice that for Theorem 4.4.8 to hold, the input graph only needs to be $\sqrt{\log n/\log\log n}$–acyclic, but we stated the theorem with the $\sqrt{\log n}$–acyclicity assumption for the sake of consistency.

The following corollary can be easily obtained showing that for the case of dense $\sqrt{\log n}$–acyclic oriented graphs, our result on the bit complexity is close to the worst-case optimal.

Corollary 4.4.9 *Given an arbitrary $\sqrt{\log n}$–acyclic oriented graph $G = (V, E)$ with maximum degree $\Delta = \Omega(2^{\sqrt{\log n \log\log n}})$, for any $\epsilon > 0$, a $(1+\epsilon)\Delta$-vertex coloring can be obtained in $O(\log \Delta)$ bit rounds, with high probability.*

4.5 Chapter Summary and Acknowledgements

We presented algorithms for distributed vertex coloring using a simple and natural model. While our results are tight in general, a related question to ask is whether any further conditions on the orientation would result in better bounds or whether certain orientations outperform other orientations. For example, if the orientation or the graph is known to be acyclic, would it be possible to color in fewer bit rounds?

A preliminary version of the results contained in this Chapter appear in [77]. This work was done with Melih Onus, Department of Computer Science, Arizona State University and Christian Schindelhauer, Computer Science Deparmtment, University of Paderborn.

Part II

Peer-to-peer Overlay Networks

Chapter 5

P2P Networks: Deterministic Constructions

5.1 Introduction

In recent years, peer-to-peer overlay networks have become extremely popular for a variety of reasons. For example, the fact that peer-to-peer systems do not need a central server means that individuals can search for information or cooperate without fees or an investment in additional high-performance hardware. Also, peer-to-peer systems permit the sharing of resources (such as computation and storage) that otherwise may sit idle on individual computers. Therefore, it is not surprising that peer-to-peer systems have inspired an enormous amount of research. Despite many advances, fundamental problems have remained open, such as:

1. Is it possible to design deterministic peer-to-peer overlay networks with properties comparable to randomized peer-to-peer systems?

2. How can peers of non-uniform bandwidth be organized in an overlay network?

Why are these problems important and non-trivial? An obvious advantage of a deterministic over a randomized solution is the ability to *locally self-correct* the overlay network so that it not only fulfills the given connectivity rules but also retains certain desirable topological properties such as a high expansion. The property of *self-stabilization* was introduced by Dijkstra in his 1974 paper [32] and is considered an important property in existing peer-to-peer systems [142, 125, 129]. By definition, (pseudo-)random constructions cannot be self-correcting with regard to expansion because the systems can be in a state with a poor expansion although all connectivity rules are fulfilled. Although this may be unlikely to happen if all nodes are honest, adaptive adversarial attacks can make such a situation very likely (see also [34]). Designing scalable, deterministic overlay networks with a high expansion is a highly non-trivial problem. The first such construction just recently emerged, and the construction and its analysis is quite involved [8].

Also, organizing peers of non-uniform bandwidth in a scalable way is an important and non-trivial problem. It is important because in reality, peers have different connections to the Internet with bandwidths that may be several orders of magnitude apart. Also, future peer-to-peer systems will have to allow peers to adjust the bandwidth they want to contribute to it to be acceptable since many peer-to-peer applications may run in a peer at the same time. Thus, a system is needed that can organize peers of non-uniform bandwidth and that can adapt to changing bandwidths in a scalable way. DHT-based peer-to-peer approaches in their basic form cannot take advantage of high bandwidth peers, because their approach of giving every peer the same degree and randomly distributing peers in the system will isolate high bandwidth peers, making them ineffective. A straight-forward solution would be to simply include multiple *virtual peers* for each high-bandwidth peer into the system. This approach, however, does not work well in general, because allowing a peer to have multiple virtual peers in the system reduces its scalability and increases its vulnerability.

It reduces its scalability because frequent bandwidth changes may create a high update cost when using virtual peers, and it increases its vulnerability because when a high-bandwidth peer leaves, many virtual peers will leave with it, potentially creating disruption of service and high repair costs for the overlay network. It would therefore be much better to give every peer just a *single, bounded degree* node in the network while retaining the property that high-bandwidth peers can be utilized well. This is exactly what we solve with the overlay network proposed in this chapter. Before we present our network, we review previous work.

5.1.1 Overlay networks for uniform peers

A large collection of scalable peer-to-peer overlay networks has been proposed in recent years. Among them are Tapestry [155], Chord [142], Pastry [129], CAN [125], Viceroy [105], Koorde [67], and DH graphs [110]. Also generic approaches have recently been presented that allow one to turn general families of static graphs into dynamic graphs. See, for example, [110] and [2]. All of these constructions crucially depend on the fact that nodes are given random IDs (which may either be obtained by a random number generator or with the help of a pseudo-random hash function). Hence, they cannot *guarantee* a good expansion or diameter.

Recently, a number of constructions for overlay networks emerged that allow good topological properties for *arbitrary* node IDs. Among them are skip graphs [6], skip nets [59], and the Hyperring [8]. Whereas skip graphs and skip nets still need a random number generator for the topology, the Hyperring is purely deterministic and the only dynamic overlay network to date that has a guaranteed low diameter and high expansion. However, whereas in the randomized constructions the work for a node insertion and deletion can be made as low as $O(\log n)$, w.h.p., the work for a node insertion and deletion in the Hyperring is $O(\log^3 n)$, where n is the current number of nodes in the system. So an open question has been whether this can be reduced to $O(\log^2 n)$ or even

$O(\log n)$. Also, the Hyperring is rather complicated to maintain and therefore an open question has also been whether simpler approaches exist for organizing peers in a deterministic way. The results of this chapter answers both of these questions in the affirmative.

5.1.2 Overlay networks for non-uniform peers

Peers of high bandwidth are often more reliable and participate in the network for a longer time than low bandwidth peers. Though it uses an unstructured approach, Gnutella has a tendency to integrate long-living peers more tightly into the network than short-living peers and therefore can be seen as a heuristic to take advantage of high bandwidth peers. A more structured approach is the super-peer architecture of Kazaa [72]. It classifies peers into two classes: the strong (high bandwidth) peers and the weak (low bandwidth) peers, and it permits a weak peer to be connected to exactly one strong peer. All queries are routed through strong peers, which are also called super-peers. Super-peer networks are also part of JXTA 2.0 [143].

Publications on various super-peer networks can be found in [111, 151, 156]. Also multi-tier topologies (i.e. topologies with more than two classes of peers) have been proposed (e.g., [139]), where each level consists of peers with approximately the same capabilities. None of these publications have studied in a formal way how well their topologies can handle arbitrary unicast or multicast problems.

5.1.3 Overlay networks for multicasting

There are a number of results on overlay networks for multicasting. Overlay based approaches that just create a network for a single multicast group can be found in [9, 31, 62, 128]. For example, in [62] the authors show how to construct an overlay network of tree topology rooted at the source of the multicast. In [9], the authors create a hierarchical topology based on clustering. In [128], the

authors show how to create a tree topology that can be used to achieve reliable broadcasting.

Approaches that allow multiple multicast groups to be formed over the same overlay network are usually implemented on top of DHT-based systems such as Chord, CAN, Tapestry, or Pastry [20, 22, 91, 126, 141, 157]. The works [141, 91] are based on the Chord [142] overlay network. The scheme of [126] use the CAN proposed in [125] so as to create a sub-CAN consisting of the nodes forming a multicast group. Multicasting in the group is then realized by flooding the sub-CAN with additional rules to minimize duplicates. The work of Bayeux [157] is based on the Tapestry network proposed in [155] and the approach of Scribe [20] is based on the Pastry network [129]. For an evaluation of several of these protocols see [22], for example. All of these approaches are scalable, but they only work well for uniform peers because messages for these multicast groups will be routed through the underlying DHT-based networks.

5.1.4 Our results

We propose a dynamic overlay network, called *Pagoda*, that can handle routing, data management, multicasting, and node insertions and deletions in a scalable and efficient way. In the following, n always denotes the current number of peers or nodes in the system.

Uniform overlay networks, routing, and data management

For the uniform case, i.e. all nodes have the same bandwidth and storage, our main results for the Pagoda network are:

- **Maintenance:** Any isolated node insertion or deletion can be executed in $O(\log n)$ time and work.

- **Routing:** There is a local, randomized routing strategy that routes any set of packets in which each node has at most one packet and every packet has a random destination in $O(\log n)$ time, w.h.p.

- **DHT:** There is a distributed hash table method that can keep data distributed among nodes so that every node is responsible for an expected $O(1/n)$ fraction of the data.

Non-uniform overlay networks and multicasting

Our main results for the non-uniform case are:

- **Maintenance:** Any isolated node insertion or deletion can be executed in $O(\log^2 n)$ time and work.

- **Muliticasting:** The Pagoda network for non-uniform nodes creates a congestion for routing arbitrary concurrent multicast requests that is only by an $O(\Delta + \log n)$ factor larger than the congestion achievable by a *best possible* overlay network of maximum degree Δ *for that particular problem*.

Apart from proving existential results, we also provide local-control strategies for building and maintaining multicast trees so that a performance as predicted by the competitive bound can be achieved. We also show that under certain local admission control scenarios, our network can guarantee that rate reservation requests for multicasting are successful with high probability.

5.1.5 Rest of the Chapter

We start in Section 5.2 with the description of the perfect, static form of the Pagoda, and we prove some basic properties. In Section 5.3 we show how to turn the Pagoda into a dynamic overlay network for the case that all nodes have the same bandwidth. We describe how to efficiently

insert a node into or delete a node from the Pagoda, and we show how to perform routing and data management in an efficient and robust manner. In Section 5.4, we extend the Pagoda network to the case that we have arbitrary non-uniform node bandwidths, and in Section 5.5 we show how this network can be used for efficient multicasting.

5.2 The static Pagoda network

Our overlay network is basically a combination of a complete binary tree and a family of leveled graphs that are similar to the well-known Omega network [92], together with some short-cut edges to keep the diameter low. It is called *Pagoda*. We first define a perfect, static form of the network before describing dynamic constructions. Below we recall the definition of $2-$ary de Bruijn graph (see also Definition 2.4.4).

Definition 5.2.1 *Let $d \in \mathbb{N}_0$. The d-*dimensional de Bruijn *graph, denoted $DB(d)$, is an undirected graph with node set $V = [2]^d$ and an edge set $E = \{\{x, y\} \mid x, y \in [2]^d$ and there are $p, q \in \{0, 1\}$ so that $x = (b_1, b_2, \ldots, b_{d-1}, p)$ and $y = (q, b_1, b_2, \ldots, b_{d-1})\}$.*

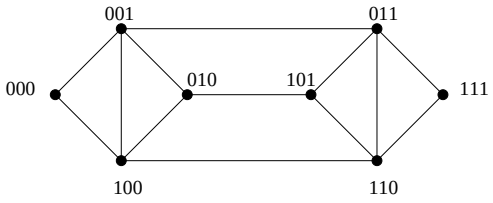


Figure 5.1: The structure of $DB(3)$

Figure 5.1 presents the 3-dimensional de Bruijn, $DB(3)$. It can be seen that $DB(d)$ has 2^d nodes, a maximum degree of 4 and diameter d. Based on the de Bruijn graph, we now define a network called the de Bruijn exchange network (DXN) below.

Definition 5.2.2 (de Bruijn Exchange Network (DXN)) *For $d \in \mathbb{N}_0$, the d-*dimensional de Bruijn exchange network, *$DXN(d)$, is an undirected graph with node set $V = [d+1] \times [2]^d$ and the following edge set:*

$$E = \{\{(j,x),(j+1,y)\} \mid j \in [d], \\ x,y \in [2]^d, \{x,y\} \in E(DB(d)) \text{ or } x = y\}$$

Thus, $DXN(d)$ can be viewed as $d+1$ copies of $DB(d)$ combined together to form a leveled network with edges in the de Bruijn network now going across adjacent levels and each node being connected to its copy in an adjacent level.

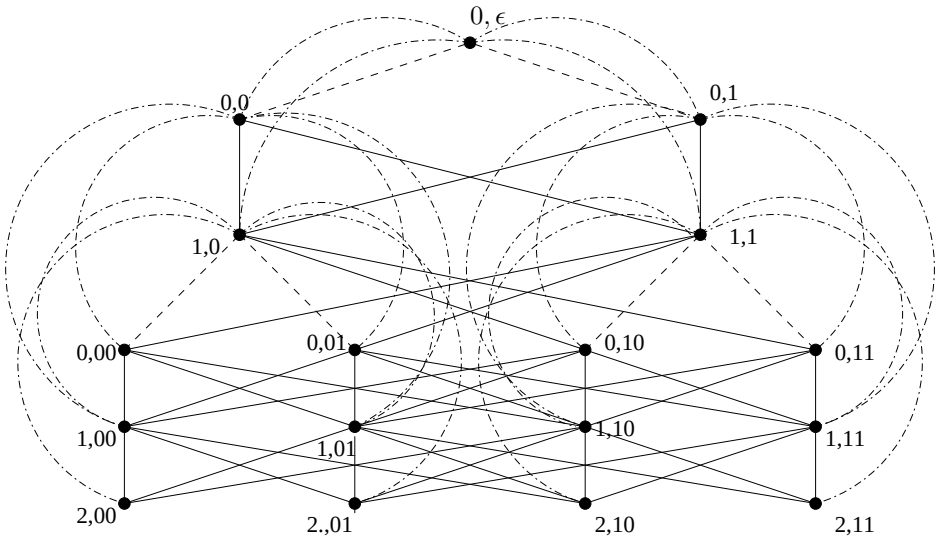


Figure 5.2: The structure of $PG(2)$ consisting of $DXN(0), DXN(1)$ and $DXN(2)$. The tree edges are shown in dashed lines and the shortcut edges are shown in dotted lines.

Definition 5.2.3 *Let $d \in \mathbb{N}_0$. The d-*dimensional Pagoda, *$PG(d)$, is an undirected graph that consists of $d+1$ de Bruijn exchange networks, $DXN(0), \ldots, DXN(d)$, where each node $(i,x) \in [i+1] \times [2]^i$ of $DXN(i)$ has an edge to the nodes $(0,x0)$ and $(0,x1)$ in $DXN(i+1)$ and, additionally, to all nodes $(0,y)$ in $DXN(i+1)$ that have an edge to $(1,x0)$ or $(1,x1)$.*

In addition to this, for every i *and* $j \in \{0, \ldots, i\}$*, every node* (j, x) *in* $DXN(i)$ *has short-cut edges to nodes* $(j, x0)$*,* $(j, x1)$*,* $(j+1, x0)$*, and* $(j+1, x1)$ *in* $DXN(i+1)$*.*

Ignoring the short-cut edges, the Pagoda is a leveled network with the root being at level 0. Levels are consecutively numbered from 0 through $(\sum_{i=0}^{d}(i+1)) - 1$. Given a node at level ℓ, the nodes it is connected to in level $\ell - 1$ are called its *parents*, and the nodes it is connected to in level $\ell + 1$ are called its *children*.

The Pagoda network consists of the following types of edges:

- *column edges* joining node (j, x) to node $(j+1, x)$ in a DXN,
- *tree edges* joining node (i, x) in $DXN(i)$ to nodes $(0, x0)$ and $(0, x1)$ in $DXN(i+1)$,
- *short-cut edges* joining node (j, x) in $DXN(i)$ to nodes $(j, x0)$, $(j, x1)$, $(j+1, x0)$, and $(j+1, x1)$ in $DXN(i+1)$,
- *de Bruijn edges* representing edges of the form (j, x) in $DXN(i)$ to $(j+1, y)$ in $DXN(i)$ with $(x, y) \in E(DB(i))$, and
- *balancing edges* representing all the remaining edges.

We also denote nodes $(0, j)$ in $DXN(i)$ of $PG(d)$ with $j \in [2]^i$ and $0 \leq i \leq d$ as *top* nodes and similarly nodes (i, j) in $DXN(i)$ of $PG(d)$ with $j \in [2]^i$ and $0 \leq i \leq d$ as *bottom* nodes and the rest as *intermediate* nodes.

Each of the above types of edges are important for our protocols to work. Column edges and tree edges allow to keep our protocols simple and efficient, de Bruijn edges allow to perform efficient routing (and deterministic level balancing in the dynamic Pagoda), and short-cut edges keep the diameter and congestion low. The balancing edges are used to minimize congestion during concurrent join/leave operations.

5.2.1 Basic properties

Figure 5.2 shows the 2-dimensional Pagoda $PG(2)$. $PG(d)$ has $\sum_{i=0}^{d}(i+1)2^i \approx (d+1)2^{d+1}$ nodes and maximum degree 20. We start with the following lemma.

Lemma 5.2.4 *$PG(d)$ has $O(d^2)$ levels and a diameter of $O(d)$.*

Proof. The number of levels in $PG(d)$ is $\sum_{i=0}^{d}(i+1) = O(d^2)$ as $PG(d)$ consists of $d+1$ DXN networks, $DXN(0)$, $DXN(1)$, $\cdots$, $DXN(d)$. The claim about the diameter follows easily as $DXN(i)$ has diameter i and distance between nodes u and v in $DXN(i)$ and $DXN(j)$ respectively is thus at most $\max\{i,j\}+d$, using the de Bruijn and shortcut edges. □

The following lemma shows that $PG(d)$ also has a good expansion. Recall that the node expansion is defined as $\alpha = \min_{U:|U|\leq|V|/2}|N(U)|/|U|$ where $N(U)$ is the neighbor set of U.

Lemma 5.2.5 *$PG(d)$ has an expansion of $\Omega(1/d)$.*

Proof. We prove the lemma by showing that every permutation routing problem in the Pagoda can be routed with an expected congestion of $O(d)$.

Consider any permutation routing problem $\pi \in [n]$. In the following we analyze the expected congestion at any node u when using the routing strategy in section 5.3.3 for π.

We start with stage 1. Consider any node in $DXN(i)$. Since there are i nodes in the column of u that may send their packet through u, u is passed by at most i packets. Any bottom node in $DXN(i)$ then gets $2(i+1)$ packets from the nodes in its 2 child columns in $DXN(i+1)$. Since all packets in the bottom nodes in $DXN(i)$ are sent to top nodes in $DXN(i)$ chosen uniformly at random, each node in $DXN(i)$ has an expected congestion of $2(i+1)$. These packets are then forwarded to the bottom nodes of the next higher exchange network, causing an expected congestion of $4(i+2)$ at the bottom nodes of $DXN(i)$. Thus, during stage 1, each node in $DXN(i)$ has an expected congestion

of at most $i+2(i+1)+4(i+2) \leq 7(i+2)$. In stage 2, the maximum number of packets that have to be shipped across $DXN(i)$ is at most the number of nodes in $DXN(i-1)$ and above, which can be easily seen to be at most $2|DXN(i)|$. Thus, each node receives on expectation at most 2 packets during stage 2. Stage 3 is symmetric to stage 1 and therefore creates the same expected congestion as stage 1. Thus, the total expected congestion at any node is $O(d)$.

Suppose now that the node expansion is $o(1/d)$. In this case there must be a set U with $|N(U)| = o(|U|/d)$ and $|U| \leq n/2$. Then consider the permutation π that requires to send all packets in nodes in U to $\bar{U}$ where $\bar{U} = V \setminus U$ with V denoting the set of nodes in $PG(d)$. In this case, the expected congestion must be $\omega(d)$, contradicting our bound above. Thus, the expansion of $PG(d)$ is $\Omega(1/d)$. □

5.2.2 Pagoda vs. existing approaches

Our overlay network construction is closest to the line of papers following the CAN approach [125]. The basic idea behind CAN is to combine an infinite complete binary tree T with a family of graphs $\mathcal{G} = \{G_\ell \mid \ell \in \mathbb{N}_0\}$ with $|V(G_\ell)| = 2^\ell$ so that for every $\ell \geq 0$ the nodes in level ℓ are interconnected according to G_ℓ. Initially, a peer is just stored at the root of the tree. Insertions and deletions of peers are handled so that the invariant is maintained that every path down the tree starting with the root contains exactly one peer. Peers are interconnected according to the edges in $\mathcal{G}$ where a peer inherits all edges to its descendants. To keep the level distribution of the nodes in balance, and therefore their degree low, it was suggested to either use deterministic load balancing along the edges in $\mathcal{G}$ or to choose random positions for newly inserted nodes [2, 3, 125]. However, for any family $\mathcal{G}$ of bounded degree graphs such a deterministic load balancing strategy can result in a very poor expansion (as low as $O(1/n^\epsilon)$ for some constant $\epsilon > 0$), so the CAN approach crucially depends on randomness to be well-connected. In contrast to this, our way of combining a tree with a

family of graphs allows local, deterministic updates while guaranteeing an expansion of $\Omega(1/\log n)$ at any time.

This result is possible because the way we use the Pagoda network in a dynamic setting differs from the CAN approach in two fundamental ways. First of all, in the dynamic Pagoda the invariant is preserved that all parent positions of an occupied node position are occupied, whereas in CAN the invariant is maintained that every path down the CAN tree from the root contains exactly one occupied position, and therefore peers are only at the edge of the CAN tree. Secondly, the Pagoda network uses a DXN network with multiple levels at each tree level and not just single-level connections. If one just used a single de Bruijn graph at each tree level as this was suggested, for example, in [99], then deterministic balancing strategies (to keep nodes with missing children at approximately the same level) would perform poorly (i.e. the expansion can be as low as $O(1/n^{\epsilon})$ for some constant $\epsilon > 0$). This would not only be the case in the CAN approach but also in our approach of having all parent positions occupied.

Also the way the Pagoda network handles non-uniform peers is fundamentally different from previous approaches. Instead of using many virtual nodes or a multi-tier network to incorporate peers of non-uniform bandwidth, every peer is just associated with a single node, and a simple heap property is used to organize the peers in the system: every parent of a peer must have a bandwidth that is at least as large as the bandwidth of that peer. Thus, local, relative rules are used to organize peers instead of the rather global nature of the rules using virtual nodes or multi-tier networks (since an agreement on the minimum bandwidth and bandwidth-to-tier assignments is necessary there).

5.3 The dynamic Pagoda network for uniform nodes

Our basic approach for the dynamic Pagoda network is to keep the nodes interconnected in a network that represents a subnetwork of the static Pagoda network of infinite dimension. In this section, we assume that all nodes have a bandwidth of 1. At any time, the dynamic Pagoda network has to fulfill the following invariant:

Invariant 5.3.1

(a) **Position:** *For any node in the dynamic Pagoda, all of its parent positions are occupied.*

(b) **Consistency:** *For any pair of nodes v and w in the dynamic Pagoda, v and w are connected in the dynamic Pagoda if and only if v and w are connected in the static Pagoda.*

We start with some facts about the dynamic Pagoda network. A node is called *deficient* if it has a missing child along a column or tree edge (i.e. we do not consider missing children reachable via de Bruijn edges).

Lemma 5.3.2 *If Invariant 5.3.1 is true, then in the dynamic Pagoda network with n nodes, the difference between the largest level and the smallest level with deficient nodes is at most* $\log n$.

Proof. Let v be any node of largest level in the Pagoda. Notice that such a node must be deficient. Suppose that v is at position (j, x) in some $DXN(d)$. The fact that every node must have all of its parent positions occupied and the way the *DXN* is constructed ensure that v is connected to at least 2^j nodes at positions $(0, y)$ in $DXN(d)$, where y is either the result of a right shift of x by at most j positions or a left shift of x by at most j positions, padded with arbitrary 0-1 combinations. Thus, if $j = d$, then all positions in row 0 of $DXN(d)$ must be occupied. If $j < d$, then one can easily check that all positions in row j in $DXN(d-1)$ must be occupied. Hence, the difference between

the largest level and the smallest level with a deficient node is at most d. Taking this into account, one can show that $d \leq \log n$, which yields the lemma. □

This lemma has some immediate consequences when combining it with results about the static Pagoda:

Lemma 5.3.3 *If Invariant 5.3.1 is true, then the dynamic Pagoda network with n nodes is a bounded degree network and has $O(\log^2 n)$ levels, a diameter of $O(\log n)$, and an expansion of $\Omega(1/\log n)$.*

Proof. The claim about degree, levels and diameter follow from Invariant 5.3.1, Lemma 5.2.4 and Lemma 5.3.2. Using the same routing strategy as in the proof of Lemma 5.2.5, one can show that every permutation routing problem can be routed with an expected congestion $O(\log n)$. This can then be used to show that the dynamic Pagoda network with n nodes has an expansion of $\Omega(1/\log n)$. □

Next we define local control algorithms that allow nodes to join and leave the system, denoted by the operations JOIN and LEAVE, while preserving Invariant 5.3.1 at any time (under the condition that nodes depart gracefully).

5.3.1 Isolated Join and Leave operations

First, we describe the JOIN and LEAVE protocol for the case that just one node wants to join or leave the system at a time.

The isolated JOIN protocol

The basic strategy of the join protocol is to make sure that every new node is inserted at a place that fulfills Invariant 5.3.1. Suppose that node u wants to join the system. This is done in two stages.

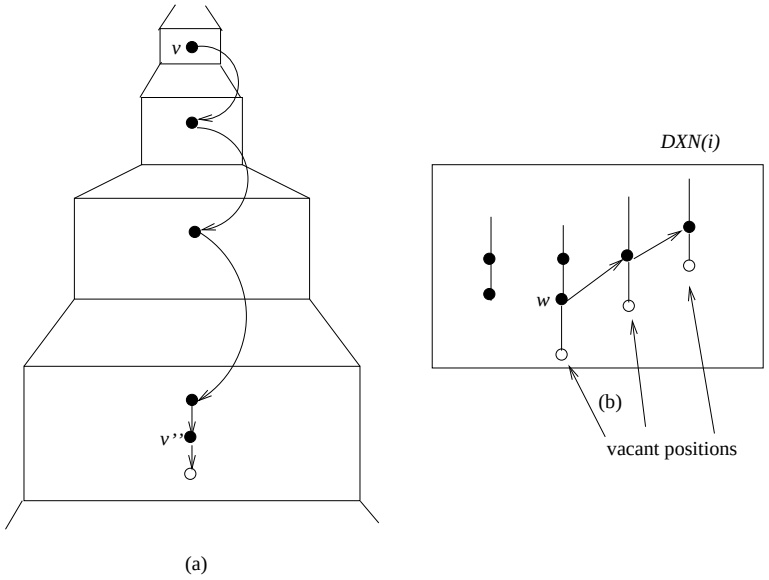


Figure 5.3: Figure (a) shows the operation of stage 1 and (b) shows the operation of stage 2.

Stage 1 Suppose that node v, at position (j, x) in $DXN(i)$, is initiating JOIN(u) to insert node u into the network. If v has a short-cut edge to a node at position $(j, x0)$ in $DXN(i+1)$, then it forwards the request to that node. Let this new node be v'. If v' does not exist then we refer to node v as v'.

We are now at some node v', at position (j', x') in $DXN(i')$. If v' has a short-cut edge to a node at position $(j', x'1)$ in $DXN(i'+1)$ (here the column with suffix 1 is used to ensure an even spreading of JOIN requests), then it forwards the request to that node. Let this node be the new v'. We repeat this until no new v' exists. Call this last node v''.

We are now at some node v'', at position (j'', x'') in $DXN(i'')$. If v'' is not deficient then v'' forwards the request to the node at position $(j''+1, x'')$ in $DXN(i'')$ if $j'' < i''$, and else it forwards the request to the node at position $(0, x''1)$ in $DXN(i''+1)$. This is the new v''. This is repeated until no new v'' exists. Call this last node w. The operation of this stage is shown in Figure 5.3(a) where v transfers the request along the edges shown. At this point stage 1 ends and we proceed with stage 2 on this node.

Stage 2 Initially, the JOIN request must be at some deficient node w. If $w = (i, y)$ in some $DXN(d)$ with $0 < i < d$, then w requests information about the column child (i.e. the child reachable via the column edge) from all parents of w. If all parents report an existing child, w can integrate u as its column child without violating Invariant 5.3.1(a). Otherwise, w forwards the JOIN request for u to any parent w' reporting a missing column child, i.e. node w' is deficient.

If $i = 0$, then w requests information from its parents about each tree child that is a parent of its column child. If all relevant tree children exist, w can integrate u as its column child, and otherwise w forwards the JOIN request to any parent w' reporting a missing tree child.

Finally, if $i = d$, then w picks any of its missing tree children v and requests information from w's parents about each column child that is a parent of v. If all relevant column children exist, w can integrate u at the position of v, and otherwise w forwards the JOIN request to any parent w' reporting a missing column child. Figure 5.3(b) shows the operation of this stage where the insert request is transfered along the edges shown.

This is continued until u can be integrated.

The isolated LEAVE protocol

Suppose that a node u wants to leave the Pagoda. This is also done in two stages. Stage 1 is the same as stage 1 for the JOIN protocol.

Stage 2 Initially, the LEAVE request must be at some deficient node w. If w has a child, then w forwards the request to any one of its children. This is continued until w does not have any children. Once this is the case, w exchanges its position with u so that u can leave the network.

The JOIN and LEAVE protocols above achieve the following result.

Theorem 5.3.4 *Any isolated* JOIN *or* LEAVE *operation can be executed in* $O(\log n)$ *time and with constant topological update work.*

Proof. Consider any JOIN request starting at some node v. From the construction, it can be seen that the request is transferred through at most d short-cut edges until the request reaches a node v' in $DXN(d-1)$ (the second largest *DXN* in the system). From a node in $DXN(d-1)$, at most $O(\log n)$ column or tree edges have to be traversed to reach a deficient node w in $DXN(d)$ or $DXN(d-1)$. From node w on, every time the request is transferred to a deficient node, the level of the node w' receiving the request decreases by one. Hence, it follows from Lemma 5.3.2 that the JOIN request can be transferred along at most $\log n$ deficient nodes. Thus, an isolated JOIN request can be executed in $O(d) = O(\log n)$ time.

Also every LEAVE request is sent along at most d short-cut edges and $O(d)$ column or tree edges until it reaches a deficient node w. From w, it takes at most $\log n$ further nodes to reach a node without children, at which the LEAVE request can be finished. Hence, also any isolated LEAVE request can be executed in $O(d) = O(\log n)$ time.

The bound on the update work (i.e. the number of edge changes) is obvious. □

5.3.2 Concurrent Join and Leave operations

We also study the congestion of concurrent versions of the JOIN and LEAVE protocol. Notice that the bounds are guaranteed.

The concurrent JOIN protocol

Suppose that node u wants to join the system. This can also be done in two stages. Stage 1 is as before, but stage 2 has to be changed to resolve conflicts among multiple JOIN requests.

Stage 2. Initially, the JOIN request must be at some deficient node w at some position (j, x). Each deficient node has a slot in its local memory for every position (j', x) with $j < j' \leq |x|$ (notice that $|x|$ is the dimension of w's *DXN*), every position $(j', x0)$ and $(j', x1)$ with $0 \leq j' \leq |x| + 1$, and every position $(j', x00)$, $(j', x01)$, $(j', x10)$, and $(j', x11)$ with $0 \leq j' \leq |x| + 2$. Now, every deficient node w at position (j, x) does the following in each time step:

- **Receive requests:** Node w receives new JOIN requests and assigns each request to an empty slot (j', y) with smallest possible j' and y. (If w runs out of empty slots, it may buffer the request somewhere else, but in the situations that we will consider, this cannot happen.)

- **Check Slots:** Node w checks for every occupied slot (j', y) with $j' > 0$ whether all the parent positions of (j', y) not managed by w are already occupied, either because there is already a node at those positions, or the deficient node responsible for those positions has already filled this slot. (This can be checked via *DXN* edges.) If not, w sends the JOIN request in (j', y) over to the deficient node w' managing a parent slot to store it there. Furthermore, w checks for every occupied slot $(0, y)$ whether slot $(|y| - 1, y/2)$ is already occupied. This is done by using the balancing edges. If not, w moves the request from $(0, y)$ to $(|y| - 1, y/2)$.

- **Insert:** If the slot (j', y) representing a child of w fulfills $j' > 0$ and is occupied, then w checks whether all the other parent positions of (j', y) are already occupied by a node (which can be done via *DXN* edges). If so, w moves to position (j', y) and inserts the node whose join request is stored in slot (j', y) at its old position. For the case that $j' = 0$, w inserts the new node at position (j', y) and sends to this node all requests in slots relevant for it. If any of the other child positions of w is still empty, w remains a deficient node.

Theorem 5.3.5 *Any set of concurrent* JOIN *requests with at most one request for each old node can be executed with congestion* $O(\log n)$.

Proof. First, we bound the congestion in non-deficient nodes, which is created in stage 1 of the JOIN protocol. Notice that any request starting at column (j, x) will proceed with columns $(j, x0)$, $(j, x01)$, $(j, x011)$, $(j, x0111)$, etc. This guarantees that for any two requests starting at columns x and y with $x \neq y$, they will traverse different columns for any columns of bit length more than $\max\{|x|, |y|\}$, no matter whether these columns are in the same *DXN* or not. Hence, as long as every node has at most one JOIN request, at most $2d$ requests will pass any node of the Pagoda in stage 1, where d is the largest value so that the Pagoda has a node in $DXN(d)$. This completes the congestion bound for non-deficient nodes.

Due to the congestion bound, at most $2d$ JOIN requests will arrive at any deficient node. Hence, the largest slot position a JOIN request will ever occupy is at most $d + d/2$ levels higher than the largest slot position of a deficient node in the Pagoda. Since levels of deficient nodes can only be $d + 2$ levels apart from each other, it is not difficult to see that the slots given to the deficient nodes always suffice to accommodate all JOIN requests in our case. Now, each JOIN request will only move along a fixed sequence of slots. This sequence has a length of at most $O(d)$, and it is the same for all requests in slots of this sequence. Hence, any slot in a deficient node is traversed by at most $O(d)$ requests, and since each deficient node only has $O(d)$ slots, the congestion bound for the deficient nodes follows. □

The concurrent LEAVE protocol

Concurrent LEAVE requests can also be done in two stages. Stage 1 is as before, but stage 2 has to be changed to resolve conflicts among multiple LEAVE requests.

Stage 2. Initially, the LEAVE request must be at some deficient node w at some position (j, x). Each deficient node has a slot in its local memory for every position (j', x) with $0 \leq j' \leq j$, every

position $(j', x/2)$ with $0 \leq j' \leq |x| - 1$ if x's least significant bit is 0, and every position $(j', x/4)$ with $0 \leq j' \leq |x| - 2$ if x's two least significant bits are 0. Now, every deficient node w at position (j, x) does the following in each time step:

- **Receive requests:** Node w receives new LEAVE requests and assigns each request to an empty slot (j', y) with largest possible j' and y. If there is no empty slot left, and y' is the lowest row number of slots in w, forward the request via a to the deficient node responsible for row numbers $y/2$. (If there is no such deficient node, w may buffer the request somewhere else, but in the situations we will consider, this cannot happen.)

- **Check slots:** Then, w checks for every occupied slot (j', y) with $j' < |y|$ whether the all child positions of (j', y) are currently occupied. If so, w checks whether the slot in the deficient nodes responsible for those positions are already filled. If not, w moves the LEAVE request to some deficient node, who will store the request in that slot.

- **Completion:** If the slot (j', x) representing w's position is occupied, then w checks whether all of its child positions are still occupied. If not, and $j' > 0$, w orders the node at position $(j' - 1, x)$ to take the position of the node whose LEAVE request occupies slot (j', x) and moves to position $(j' - 1, x)$. If $j' = 0$, then w checks whether its parent is a deficient node. If not, w does the same as for $j' > 0$. Otherwise, w takes over the position of the node in slot (j', x) and forwards all of its remaining LEAVE requests to its parent.

Notice that whenever a LEAVE request meets a JOIN request, the node that wants to join can take over the position of the node that wants to leave, finishing the two requests immediately.

Theorem 5.3.6 *Any set of at most $n/2$ concurrent* LEAVE *requests can be executed with congestion $O(\log n)$.*

Proof. The proof is very similar to the proof for the JOIN requests above. □

5.3.3 Routing

Suppose that we want to route unicast messages in the Pagoda network. Consider any such unicast packet p with source $s = (j, x)$ in $DXN(i)$ and destination $t = (j', z)$ in $DXN(i')$. First, p picks a random pair of real values $(c, r) \in [0, 1)^2$ (a precision of $\log n$ bits for each is sufficient). Then, p is sent in three stages:

1. **Spreading stage:** First, send p from s along column edges and a tree edge to $(i - 1, x/2)$ in $DXN(i - 1)$. Then, send p upwards to the node $(0, y)$ in $DXN(i - 1)$ with y being the closest prefix of r. From there, forward p to the node $(k, y/2)$ in $DXN(i - 2)$ with $k/(i - 2)$ being closest to c.

2. **Shuttle stage:** Forward p along short-cut edges across nodes (k', y') with k' being closest to c and y' being the closest prefix of r until a node (k', y') in $DXN(i' - 2)$ is reached.

3. **Combining stage:** Perform stage 1 in reverse direction (with s replaced by t) to forward p to t.

Notice that as long as s and t are non-deficient nodes, this strategy is successful even *while* nodes join and leave the system, because the position of every node that is an non-deficient node will be fixed in the Pagoda. Also, whenever a node leaves, the node replacing it can inherit its packets so that no packet gets lost. More general strategies for ensuring reliable communication even while nodes are moving, using the concept of virtual homes, can be found in Section 5.5.6.

With these facts in mind, one can easily design a protocol based on the random rank protocol (see, e.g., [130, Chapter 7]) to show the following theorem.

Theorem 5.3.7 *If every node wants to send at most one packet, the packets have random destinations, and every node being the destination of a packet does not move for $O(\log n)$ steps, the routing strategy above can route the packets in $O(\log n)$ time, with high probability.*

5.3.4 Data management

Finally, we show how to dynamically manage data in Pagoda. We use a simple trick to distribute data evenly among the nodes of the Pagoda so that it is searchable. Suppose that we have a (pseudo-)random hash function mapping each data item to some real vector $(c, r) \in [0, 1)^2$. The current place of a data item d is always the lowest possible position (j, x) in the Pagoda where x is the closest prefix of r and $j/|x|$ is closest to c among all $j'/|x|$ with $0 \leq j' \leq |x|$ ($|x|$ denotes the length of x, and thus the dimension of the *DXN* owning (j, x)).

This strategy implies that if $DXN(d)$ represents the largest exchange network that has occupied positions in the Pagoda, then all data items will be stored at nodes in $DXN(d-2)$, $DXN(d-1)$, or $DXN(d)$. Since every node will at most have to store an $O(1/(d \cdot 2^d))$ fraction of the data and $d \cdot 2^d = \Theta(n)$, we get:

Theorem 5.3.8 *The data management strategy ensures that every node is only responsible for an expected $O(1/n)$ fraction of the data at any time, and this bound even holds with high probability if there are at least $n \log n$ data items in the system.*

Notice that none of the DHT-based systems can achieve the bounds above in their basic form – they only achieve a bound of $O(\log n/n)$. Combining the data management strategy with our routing strategy above, requests to arbitrary, different data items with one request per node can be served in $O(\log n)$ time, w.h.p. The results in Section 5.5 imply that this also holds for cases in which some nodes want to access the same data item, i.e. we have a multicast problem, if requests

can be combined.

5.4 The dynamic Pagoda network for non-uniform nodes

Next we show that the Pagoda network can also be used for arbitrary non-uniform node bandwidths. In this case, we want to maintain the following heap property to allow efficient multicasting, apart from the invariants for the Pagoda network of uniform nodes.

Invariant 5.4.1 *For any node v in the Pagoda,*

(a) **Position:** *all of its parent positions are occupied,*

(b) **Consistency:** *For any pair of nodes v and w in the dynamic Pagoda, v and w are connected in the dynamic Pagoda if and only if v and w are connected in the static Pagoda.*

(c) **Heap:** *the bandwidth of v is at most the bandwidth of any of its parents.*

Similar to the uniform case, we require these invariants to be fulfilled *while* nodes join and leave the system. Because of item (c), we cannot just do a single exchange operation to integrate or remove a node but we have to be more careful. First, we describe the JOIN and LEAVE operations for the isolated case, and then we consider the concurrent case.

5.4.1 Join and Leave operations

For any node u in the Pagoda, *max-child(u)* refers to the child of maximum bandwidth and *min-parent(u)* refers to the parent with minimum bandwidth.

The isolated JOIN protocol

Suppose that node v is executing JOIN(u) to insert a new node u with bandwidth $b(u)$ into the network. This is done in three stages. Stages 1 and 2 are identical to the uniform case. So it remains to describe stage 3 which is similar to inserting a node in a binary heap.

Stage 3 Once the JOIN request for u has reached a deficient node with an empty column or tree child position in which u can be integrated without violating Invariant 5.4.1(a), u is integrated there with *active bandwidth* $a(u)$ equal to the minimum of $b(u)$ and the bandwidth of its min-parent. The active bandwidth is the bandwidth it is allowed to use without violating Invariant 5.4.1(b). Then, u repeatedly compares $b(u)$ with $a(u)$. If $a(u) < b(u)$, it replaces its position with the position of its min-parent and afterwards updates $a(u)$ to $\min\{b(u), b(\text{min-parent}(u))\}$. Once u reaches a position with $a(u) = b(u)$, the JOIN protocol terminates. The process of moving u upwards is called *shuffle-up*.

The isolated LEAVE protocol

Suppose that a node u wants to leave the Pagoda. Then it first sets its active bandwidth to $b(u)$. Afterwards, u repeatedly replaces its position with its max-child and updates its active bandwidth to $a(u) = b(\text{max-child}(u))$ until it reaches a position with no child. At this point, u is excluded from the system so that Invariant 5.4.1 is maintained. The process of moving u downwards is called *shuffle-down*.

Bandwidth changes

If the bandwidth of some node u increases, we use the shuffle-up procedure, and if the bandwidth of some node u decreases, we use the shuffle-down procedure to repair Invariant 5.4.1.

Isolated update requests have the following performance.

Theorem 5.4.2 *Any isolated join operation, leave operation, or bandwidth change of a node needs $O(\log^2 n)$ time and work to repair the invariant.*

Proof. First, consider the insertion of some node u. The process of moving the request of u downwards only needs $O(\log n)$ time. According to Lemma 5.3.3, u is integrated at some level $\ell = O(\log^2 n)$. Hence, the shuffle-up process only requires $O(\log^2 n)$ messages and edge changes because each exchange of positions between u and some parent v to repair Invariant 5.4.1 moves u one level upwards and requires updating only a constant number of edges. Every shuffle operation maintains the invariant for all nodes involved in it. Hence, the total time and work is $O(\log^2 n)$.

For the case of an isolated leave operation of node u, it holds that the leave request would be transferred along $O(\log^2 n)$ levels according to Lemma 5.3.3. Thus, the shuffle-down process requires $O(\log^2 n)$ messages and edge changes. Hence, the total time and work required for an isolated leave operation is $O(\log^2 n)$. Bandwidth changes are handled as either a shuffle-up or shuffle-down and hence the time and work requirements for bandwidth change are also $O(\log^2 n)$.

□

The concurrent JOIN protocol

The only difference between the isolated and concurrent JOIN protocol is that we need to be more careful about exchanging positions. If a node u wants to replace its position with some parent v, then u checks whether v is a node that has not finished its JOIN operation or bandwidth increase operation yet (i.e. $a(v) < b(v)$). If so, u does nothing. Otherwise, u replaces its position with v.

The concurrent LEAVE protocol

Also the concurrent LEAVE protocol is similar to the isolated LEAVE protocol, with the only difference that if some node u in the process of leaving the network wants to replace its position with some child v, u first checks whether v is a node that has not finished its LEAVE operation or bandwidth decrease yet (i.e. $a(v) > b(v)$). If so, u does nothing. Otherwise, u replaces its position with v.

Bandwidth increase or decrease is handled similarly. The next lemma shows that the concurrent operations always terminate with a work that is at most the sum of the work for isolated update operations.

Lemma 5.4.3 *For any set of k concurrent insertions, deletions, and bandwidth changes of nodes, the work and time required to repair Invariant 5.4.1 is $O(k \log^2 n)$.*

Proof. The work bound is obvious. Thus, it remains to prove the time bound.

Consider k concurrent update requests. From the analysis in the uniform case we know that $O(k \log n)$ work is necessary for nodes of JOIN requests to be integrated into the system. Each time step progress is made here until all JOIN requests are integrated.

Afterwards, we mark all nodes with 1 that have not completed their JOIN or bandwidth increase operation yet, all nodes with -1 that have not completed their LEAVE or bandwidth decrease operation yet, and all other nodes with 0. Suppose that there is at least one node marked as 1. Then let v be any of these nodes of minimum level. Since the level of v must be at least 1 (as the root cannot be a 1-node), it can replace its position with its min-parent, thereby making progress.

On the other hand, suppose that there is at least one node marked as -1. Then let v' be any of these nodes of maximum level. If v' does not have any children, then v' can leave, and otherwise it can replace its position with its max-child, thereby making progress in any case.

Hence, we make progress in every time step. Since the total work of the shuffle-up, shuffle-down, and departure operations is bounded by $O(k \log^2 n)$, the time spent for executing these operations is also bounded by $O(k \log^2 n)$. □

5.5 Multicasting

Finally, we study how well the non-uniform Pagoda supports arbitrary concurrent multicasting. We first define the concurrent multicast problem, provide a routing strategy in the Pagoda network to route multicast requests and then finally show that the strategy is competitive with respect to congestion in the best possible network for the given problem.

5.5.1 The concurrent multicast problem

In the concurrent multicast problem, we are given a set of client-server-demand triples called *streams*, (T_k, s_k, D_k), where T_k is a set of client nodes served by a server node s_k and D_k is a demand vector which specifies the flow demanded of s_k by each client node for $k \geq 1$. All the k multicast requests are to be satisfied concurrently and we are interested in the congestion caused in the Pagoda network due to the flows created. We are interested in comparing the congestion created in the Pagoda network to that of an optimal network of degree Δ_{OPT} for the given multicast problem. Notice that our definition of the problem allows for a single multicast stream to have different classes of service based on the demand vector.

5.5.2 Routing strategy

We start by constructing a flow system for one server, s_k, and one client $t \in T_k$. We name this flow system, $f_{k,t}$. We assume that s_k is a node in $DXN(i)$ and t is a node in $DXN(j)$. Our

routing strategy has three stages, called the *spreading* stage, the *shuttle* stage and the *combining* stage, described below.

1. **Spreading stage:** This stage spreads flow originating at s_k in $DXN(i)$ evenly among the nodes in $DXN(i-2)$. This is done in three steps.

   a. Move the flow from s_k along column edges to the top node in $DXN(i)$.

   b. Move the flow upwards to the bottom node in $DXN(i-1)$ along the tree edge connecting the two DXN's. From there, cut the flow into 2^{i-1} flow pieces of uniform size and send piece i upwards to node $(0,i)$ along the unique path of de Bruijn edges representing right shifts.

   c. Move all flow from the top nodes in $DXN(i-1)$ to the bottom nodes in $DXN(i-2)$ along tree edges. Every bottom node in $DXN(i-2)$ sends flow along its column edges so that each node in the column gets the same fraction of flow. That is, at the end every node in $DXN(i-2)$ has a $1/((i-1)2^{i-2})$ fraction of the flow of s_k.

2. **Shuttle stage:** Short-cut edges are used to send the flows forward to $DXN(j-2)$ (which may be upwards or downwards in the Pagoda) so that the flows remain evenly distributed among the nodes in each exchange network visited from $DXN(i-2)$ to $DXN(j-2)$.

3. **Combining stage:** This stage is symmetric to stage 1, i.e. we reverse stage 1 to accumulate all flow in t.

This results in a flow system, $f_{k,t}$, for a source s_k and a destination $t \in T_k$. Let $f_{k,t}(e)$ be the flow through any edge e in this flow system. The procedure is repeated for each client $t \in T_k$. We now construct a flow system, f_k, for the stream k. We lay the flow systems $f_{k,t}$ one on top of the other. The flow through an edge in system f_k is the maximum flow through the same edge in

each $f_{k,t}$. That is, let $f_k(e)$ be the flow through any edge e in flow system f_k. Then we have the following:

$$f_k(e) = \max_{t \in T_k} f_{k,t}(e)$$

Note that we select the maximum flow because if there are two flows of the same stream going through an edge then we simply keep the one with the higher bandwidth (the lower bandwidth stream may be reconstructed from the higher one). We use flow system f_k to route multicast flow for stream k.

5.5.3 Competitiveness

In this section we show that the Pagoda network is $O(\Delta_{\mathrm{OPT}} + \log n)$-competitive with respect to congestion in the best possible network of degree Δ_{OPT} when the multicast problem is posed as a flow problem. Specifically, we prove the following theorem.

Theorem 5.5.1 *The Pagoda network on n nodes of non-uniform bandwidth that satisfies Invariant 5.4.1 has a competitive ratio of $O(\Delta_{\mathrm{OPT}} + \log n)$ for any multicast flow problem compared to the congestion in an optimal network for this problem whose degree is bounded by Δ_{OPT}.*

Proof. Let OPT be a network of degree Δ_{OPT} that routes the given flow system with congestion C_{OPT}. Without loss of generality, we assume that every demand is at most the bandwidth of the source and destination.

Consider any node u in pagoda. Let it be in exchange network $DXN(i)$. We show that the congestion at this node due to the flow system resulting from our routing strategy above is no more than $O(\log n C_{\mathrm{OPT}})$ due to stages 1 and 3 and $O(\Delta_{\mathrm{OPT}} C_{\mathrm{OPT}})$ due to stage 2. We show these bounds in parts. We first bound the congestion at u due to stage 1, $c_1(u)$. The flows through u due to stage 1 are the sum of the flows that originate in $DXN(i)$, $DXN(i+1)$ and $DXN(i+2)$.

Let the congestion due to each of these be $c_{1a}(u)$, $c_{1b}(u)$ and $c_{1c}(u)$ respectively. Clearly, $c_1(u) = c_{1a}(u) + c_{1b}(u) + c_{1c}(u)$. We bound each of these three separately:

Stage 1a: Node u receives flow from nodes that are below it (in the same column) in exchange network $DXN(i)$. We call this set $S_{u,1a}$. The flow is $\sum_k \max_{v \in S_{u,1a}}\{d_k(v)\}$. Note that the max term is used since flows belonging to the same stream are combined, resulting in a flow of largest demand among these. Therefore, the congestion at u is:

$$c_{1a}(u) = \frac{1}{b(u)} \sum_k \max_{v \in S}\{d_k(v)\} \leq \sum_{v \in S} \sum_k \frac{d_k(v)}{b(v)} \leq |S_{u,1a}| \cdot C_{\mathrm{OPT}}$$

The set $S_{u,1a}$ contains at most $\log n$ nodes. Therefore $c_{1a}(u) \leq \log n \cdot C_{\mathrm{OPT}}$.

Stage 1b: Node u receives flow from the bottom nodes of *DXN*(i). Let $f'_k(\cdot)$ be the flow sent up by a bottom node. Thus, each bottom node sends a flow of $f'_k(\cdot)/2^i$ to each top node. Note that f' is purely the spreading caused by stage 1b.

Let $S_{u,1b}$ be the set of bottom nodes with paths crossing u, and let D be the set of top nodes with paths crossing u. We now bound $|S_{u,1b}|$ and $|D|$. Let u be in level h of *DXN*(i). There are 2^i nodes in each level of *DXN*(i), and each node has an address of i bits. Due to the bit-shift routing of the de Bruijn graphs, the number of paths crossing u is $|S_{u,1b}| \cdot |D| = 2^i$. This can be seen by observing that the nodes in $S_{u,1b}$ must have the same $i - h$ most significant bits as u and nodes in D must have the same h least significant bits as u giving $|S_{u,1b}| = 2^{i-h}$ and $|D| = 2^h$.

The flow from each node $v \in S_{u,1b}$ that reaches u is $\frac{|D| \cdot f'_k(v)}{2^i}$, which is the number of nodes in D times the amount of flow destined for each node in the top row of $DXN(i)$. Since $\frac{|D|}{2^i} = \frac{1}{|S_{u,1b}|}$, this becomes $\frac{f'_k(v)}{|S_{u,1b}|}$.

Since flows belonging to the same multicast group merge into one flow equal to the maximum of the two it follows that the flow that reaches u is $\sum_k \max_{v \in S_{u,1b}} \frac{f'_k(v)}{|S_{u,1b}|}$. Assuming v_1 and v_2 are

the two tree children of v, the congestion at u is

$$c_{1b}(u) = \frac{1}{b(u)} \sum_k \max_{v \in S_{u,1b}} \frac{f'_k(v)}{|S_{u,1b}|} \leq \sum_{v \in S_{u,1b}} \frac{\sum_k f'_k(v)}{b(v) \cdot |S_{u,1b}|}$$
$$\leq \frac{1}{|S_{u,1b}|} \sum_{v \in S_{u,1b}} (c_{1a}(v_1) + c_{1a}(v_2)) \leq 2 \log n \cdot C_{OPT}$$

Stage 1c: Node u receives flow from the bottom node in its column. Therefore, the congestion at u, $c_{1c}(u)$ is at most the congestion at the bottom node in the exchange network. The bottom node receives flow from its two descendants in *DXN*$(i+1)$. Note that the two descendants will send up equal flows, let one of them be v. So, $c_{1c}(u) \leq 2c_{1b}(v) \leq 4 \log n \cdot C_{\mathrm{OPT}}$.

We show the bounds for flows due to stage 2 with the help of Lemma 5.5.2. We need to lower bound the congestion that an optimal network can achieve. We do this by showing how an optimal network has limited bandwidth to send flows.

Lemma 5.5.2 *Let E_{OPT} be the set of edges in the optimum network. For any pair of sets X and Y that are subsets of the set of nodes, let $D(X,Y) = \sum_{s_k \in X} max_{v \in T_k \cap Y} \{d_k(v)\}$ and $B(X,Y) = \sum_{(u,v) \in E_{\mathrm{OPT}} \cap X \times Y} min\{b(u), b(v)\}$. Then $C_{\mathrm{OPT}} \geq D(X,Y)/B(X,Y)$.*

Proof. Consider any pair of sets $X, Y \subseteq V$. $B(X,Y)$ as defined in the statement measures the bandwidth between sets X and Y. Note that it is not necessary that X and Y form a cut. Similarly, $D(X,Y)$ is the demand that X asks of Y. The ratio of $B(X,Y)$ to $D(X,Y)$ is the average congestion. The maximum congestion must be at least the average congestion. Therefore $C_{\mathrm{OPT}} \geq \frac{D(X,Y)}{B(X,Y)}$. □

Stage 2: Let U be the set of nodes in the Pagoda which belong to all exchange networks above and including *DXN*$(i+1)$. Let Z be all nodes in exchange network *DXN*$(i+2)$. Let V be all nodes below and including exchange network *DXN*$(i+3)$. Let the collective flow through exchange network *DXN*(i) be f. Any stream whose source is in $U \cup Z$ and has a destination in $V \cup Z$ must go through *DXN*(i) according to our routing strategy. The expression for the flow is:

$$f = \sum_{s_k \in U \cup Z} \max_{v \in V \cup Z} d_k(v)$$

Due to lemma 5.5.2 we bound f as follows:

$$f \leq (|U| \Delta_{\mathrm{OPT}} \max_{i \in V \cup Z} \{b_i\} + |U \cup Z| \Delta_{\mathrm{OPT}} \max_{i \in V} \{b_i\} + |Z| \Delta_{\mathrm{OPT}} \max_{i \in Z} \{b_i\}) \cdot C_{\mathrm{OPT}}$$

The first term accounts for bandwidth between U and $V \cup Z$, the second term for bandwidth between V and $U \cup Z$, and the third term for bandwidth within Z. Hence,

$$f \leq 3\,|U \cup Z| \Delta_{\mathrm{OPT}} \max_{i \in V \cup Z} \{b_i\} \cdot C_{\mathrm{OPT}} \leq 3\,|U \cup Z| \Delta_{\mathrm{OPT}}\, b_u \cdot C_{\mathrm{OPT}}$$

Since the Pagoda spreads tree flow evenly across all nodes in each exchange network, the flow through u is at most $\frac{f}{|DXN(i)|}$. Therefore $c_2(u) \leq \frac{f}{|DXN(i)| \cdot b_u}$. The construction of the Pagoda implies that $|U \cup Z| < 2\,|Z|$, and $|DXN(i)| \geq \frac{|Z|}{12}$. Thus, $c_2(u) \leq 72\ \Delta_{\mathrm{OPT}} \cdot C_{\mathrm{OPT}}$.

The congestion at u due to stage 3 is identical to the congestion due to stage 1 because the two cases are symmetric. Hence, $c(u) = 2\,c_1(u) + c_2(u) \leq (14 \log n + 72\ \Delta_{\mathrm{OPT}}) \cdot C_{\mathrm{OPT}}$. The theorem follows. □

5.5.4 Turning multicast flows into trees

In practice, it may be expensive or impossible to divide and recombine streams. Instead, we choose a pseudo-random hash function h that maps every node v in the Pagoda to a pair of real values $(c, r) \in [0, 1)^2$. Similar to the routing strategy in Section 5.3.3, we can then adapt the multicast scheme in the following way for a source s and target t:

1. **Spreading stage:** This stage has three sub-stages as earlier. Stage (a) is the same as above, but instead of spreading the flow in (b), we route all flow to the node $(0, y)$ in $DXN(i-1)$ with y being the closest prefix of r. From there during stage (c), we forward the flow to the node $(k, y/2)$ in $DXN(i-2)$ with $k/(i-2)$ being closest to c. (In the above, notice that we are comparing a label in $[2]^i$ with a real number r which is done by treating the label in $[2]^i$ as a binary decimal number which then represents a real number in $[0, 1)$ uniquely.)

2. **Shuttle stage:** Forward the flow along short-cut edges across nodes (k', y') with k' being closest to c and y' being the closest prefix of r until a node (k', y') in $DXN(j-2)$ is reached.

3. **Combining stage:** Reverse the spreading stage to send the flow to t.

Multicast flows that belong to the same stream are combined so that for every edge e, the flow for that stream through e is the maximum demand over all flows of targets t that are part of that stream. Using this rule, it is not surprising that the expected congestion of our integral flow scheme is equal to the congestion of the divisible flow scheme above.

Theorem 5.5.3 *The integral multicast flow scheme has an expected competitive ratio of* $O(\Delta_{\mathrm{OPT}} + \log n)$ *compared to an optimal network with degree* Δ_{OPT}.

Proof. The theorem can be shown by following the line of arguments in the proof of Theorem 5.5.1. Here, we just give an intuition of why the theorem is correct. We start with bounding the expected

congestion for stages 1 and 3.

Lemma 5.5.4 *The expected congestion from routing the spreading stage is $O(\log n)$-competitive against an optimal network of degree Δ_{OPT}.*

Proof. Let d_i be the total demand requested by node i across all streams, and let b_i be node i's bandwidth.

Consider the congestion on any node in $DXN(i)$ due to stage (a) of the spreading stage. Since flow is sent up along column edges, the maximum congestion occurs at the top nodes of $DXN(i)$. If $d_{\max}$ is the largest demand of any node in some node v's column, then v must route at most $(i+1) \cdot d_{\max}$ demand, under the worst case assumption that the demands are for different streams and cannot be combined. Since v has at least the bandwidth of every node with demand d_{max}, this is $O(\log n)$-competitive with respect to congestion.

Now consider the congestion at any node in $DXN(i-1)$ caused by stage (b) of the spreading stage. Here, any node $(0,j)$ in $DXN(i-1)$ receives flow along the column edges from node $(i-1,j)$ in $DXN(i-1)$ which receives flow from at most two nodes $(0,j0)$ and $(0,j1)$ in $DXN(i)$. Since the nodes in $DXN(i)$ have flow of at most $(i+1)d_{\max}$ from stage (a), the demand at any node $(0,j)$ in $DXN(i-1)$ is at most $2(i+1)d_{\max}$ which is $O(\log n)$–competitive with respect to congestion. From the bottom nodes in $DXN(i-1)$ the flow reaches a top node in $DXN(i-1)$. Since these top nodes are chosen independently and uniformly at random, the expected demand at any of the top nodes is no more than the demand at the bottom nodes. Further, the bit-shift routing properties of the de Bruijn graph [130] which can be extended to DXN imply that the expected demand at the intermediate nodes in $DXN(i-1)$ would be no more than the demand at the bottom nodes. Putting these arguments together, one can show that the expected congestion at the top nodes in $DXN(i-1)$ is $O(\log n)$–competitive.

Similar arguments apply to stage (c) of the spreading stage and one can show that the expected congestion at any node in $DXN(i-2)$ is $O(\log n)$–competitive. □

The following lemma bounds the expected congestion due to stage 2.

Lemma 5.5.5 *The expected congestion from routing flow in the shuttle stage is $O(\Delta_{\mathrm{OPT}})$–competitive against an optimal network of degree Δ_{OPT}.*

Proof. Consider the boundary between any two DXN networks. The flows crossing this boundary upwards (resp. downwards) along short-cut edges must have a set of sources S and a set of destinations T with $S \cap T = \emptyset$. Hence, there is a cut in the optimal network that all these flows have to cross. Furthermore, since we are sending exactly one copy of the stream across the cut, we are sending no more flow than OPT must send. The same upper bound on the demand across a cut as shown in Lemma 5.5.2 holds as in the divisible flow case. Since the nodes along which the flows travel are randomly selected, the expected congestion at any node would be the total flow divided by the number of nodes in the DXN, which implies that the congestion is $O(\Delta_{\mathrm{OPT}})$-competitive in expectation. □

Combining the two lemmata and noting that the congestion due to stage 3 is no more than the congestion of stage 1, yields Theorem 5.5.3. □

5.5.5 Multicast streaming

Next, we address the issue of how to use the multicasting capabilities for multimedia streaming where peers can enter and leave a multicast stream at any time. To ensure reliable streaming, a mechanism is needed to join and leave a multicast stream, to reserve bandwidth in the nodes along that stream, and to use a local admission control rule for admitting multicast stream requests in a

fair and transparent way.

Joining and leaving a multicast stream

Consider the situation that node u in the Pagoda wants to join a multicast stream S of source s. Node u then prepares a control packet containing the demand d requested by it and sends the control packet to s as described in Section 5.5.4. Along its way, the control packet will try to reserve a bandwidth of d. If it succeeds, it will continue to reserve bandwidth along its way until it reaches a point in which for the stream S a bandwidth of at least d is already reserved.

Every node along the multicast stream will only store for each of its incoming edges the client requesting the stream with the largest demand.

Suppose now that some node u wants to leave a multicast stream S. Then it first checks whether it is the client with largest demand for S that traverses itself by checking its incoming edges. If not, u does not need to send any control packet. Otherwise, u checks whether there is a path of some client v for S into u. If so, u prepares a control packet with the largest demand of these clients. Otherwise, u prepares a control packet with demand 0. This control packet is sent towards the source s of S as in Section 5.5.4. Each time the control packet reaches a node v that is also traversed by other clients to S (that arrive at different incoming edges), the demand of the control packet is updated to the largest demand of these clients. This is continued until the control packet reaches a node v traversed by some client for S with demand larger than the original demand of u.

Rate reservation

For a rate reservation scheme to be transparent and fair, a policy is needed that gives every peer a simple, local admission control rule with the property that if a request is admissible according to this rule, then the rate reservation request should succeed with high probability. We will investigate

two such rules:

Suppose that every node v representing a server in the network offers multimedia streams $s_1^{(v)}, s_2^{(v)}, \ldots$ with rates $r_1^{(v)}, r_2^{(v)}, \ldots$ so that $\sum_i r_i^{(v)} \leq b(v)$. Then consider the following rules for some client v.

- **Admission rule 1:** Admit any multicast request to some server w as long as $b(v) \leq b(w)$ and the total demand of the requests in v does not exceed $\epsilon b(v)/\log n$.
- **Admission rule 2:** Admit any multicast request to some server w as long as v is not belonging to any other multicast group and the demand of the request does not exceed $\epsilon \frac{\min\{b(v),b(w)\}}{\log n}$.

Rule 1 will normally be the case in practice because servers of streams usually have a higher bandwidth than clients, but rule 2 would also allow multicasting if this is not true.

Theorem 5.5.6 *When using admission rule 1 or 2, every request fulfilling this rule can be accommodated in the Pagoda, w.h.p.*

Proof. Recall the integral multicast routing strategy in Section 5.5.4. Consider any multicast problem that fulfills rule 1 or rule 2. Using the proof of Theorem 5.5.1, one can easily show that for any node u in the Pagoda, $c_{1a}(u) = c_{1b}(u) = c_{1c}(u) = O(\epsilon)$ and $c_2(u) = O(\epsilon)$. Hence, the expected total amount of demand traversing u is $O(\epsilon b(u))$. Since any single demand through u can be at most $\epsilon b(u)/\log n$ (demands from or to a node v will always traverse only nodes w with $b(w) \geq b(v)$), and the flows for different servers follow paths chosen independently at random, it follows from the well-known Chernoff bounds that the total amount of demand traversing u is also $O(\epsilon)$ with high probability. Hence, making the constant ϵ small enough, the admission rules 1 and 2 will work correctly with high probability. □

Notice that also a combination of rules 1 and 2 is allowed.

5.5.6 Multicasting in a dynamic setting: virtual homes

Our multicast tree approach above has several problems. First, it requires to know the position of the server in the Pagoda to join a stream from it, and second, it requires to update the multicast stream each time the server or a client moves. Fortunately, this problem has an easy solution: For every node v, let $h(v) \in [0,1)^2$ be chosen *independent* on its position in the Pagoda. For example, $h(v)$ may depend on v's IP address. Then v can treat the node closest to $h(v)$ two DXNs above v as its *personal virtual home* that only has to move if v leaves its current DXN.

Suppose that every node continuously informs its virtual home about its current position and that virtual home responsibilities are exchanged whenever nodes exchange positions. Then v only has to update its connection to the multicast stream if it leaves its current DXN. However, when using the short-cut edges, such an update can be done in constant time so that the disruption of service to v is kept at a minimum. While frequent switches between DXNs could cause frequent update operations, a lazy virtual home update strategy can be used to easily solve this problem.

A third problem with dynamic conditions is that intermediate nodes may change their requested bandwidth. We can use *active* bandwidth restrictions to ensure that the previous invariant continues to hold, so that routing is still valid. Since the invariant continues to hold, congestion remains low and the admission control theorems remain true.

5.6 Chapter summary and acknowledgements

In this chapter we have shown that there exist deterministic constructions of overlay networks that can handle peers of non-uniform bandwidth efficiently. We showed that the resulting network, *Pagoda*, guarantees a logarithmic diameter, bounded degree, and 1/logarithmic expansion. The construction is also deterministic.

Isolated peer join and leave operation can be done using $O(\log^2 n)$ time and work and reduces to $O(\log n)$ time and constant topological update for the special case where all the nodes in the network have same bandwidth. We also investigated the time and work bounds for the case of concurrent join and leave operations.

Apart from the above properties, we also showed that the Pagoda network guarantees good load balancing properties when used as a distributed hash table, in the case where all the nodes in the network offer the same storage.

In addition, we demonstrated that the Pagoda network has a good competitive ratio with respect to congestion for routing concurrent multicast requests (see Theorem 5.5.1).

In the current form, when the nodes have non-uniform storage capacities, the Pagoda network is not easily amenable as a DHT satisfying load balance properties. In fact, distributed data management for uniform storage systems is well understood as most structured overlay networks can be easily used to arrive at a good data management strategy. But for the case of non-uniform storage systems, very little is known. Only recently [133] solutions are proposed.

A preliminary version of this chapter appeared in [14]. This work is done jointly with Ankur Bhargava, Chris Riley and Mark Thober. Qian Li also participated in early stages of this work.

Chapter 6

P2P Networks: Supervised P2P Systems

In the previous chapter, we saw that designing deterministic peer-to-peer networks is a highly non-trivial problem. Though we presented a solution for this, the entire construction is quite complex. On the other hand, it is known that traditional served-based systems can provide guarantees on reliability and are therefore preferable for critical applications that need a high level of reliability. But they are not easily scalable unless special high-cost hardware is employed. However, the advantage of peer-to-peer systems is that they can scale to millions of sites even with low-cost hardware. An interesting question is whether it is possible to marry the two approaches in order to share their benefits without sharing their disadvantages. In this chapter, we propose *supervised peer-to-peer systems* as a possible solution. Our approach also results in deterministic constructions.

6.1 Introduction

A *supervised peer-to-peer system* is a system in which the overlay network is formed by a supervisor but in which all other activities can be performed on a peer-to-peer basis without involving the supervisor. That is, all peers that want to join (or leave) the network have to contact

the supervisor, and the supervisor will then initiate their integration into (or removal from) the network. All other operations, however, may be executed without involving the supervisor. In order for a supervised network to be highly scalable, we propose two central requirements that have to be fulfilled:

1. The supervisor needs to store at most a polylogarithmic amount of information about the system at any time (i.e. if there are n peers in the system, storing contact information about $O(\log^2 n)$ of these peers would be fine, for example), and

2. The supervisor needs at most a constant number of messages to include a new peer into, or exclude an old peer, from the network.

The second condition makes sure that the work of the supervisor to include or exclude peers from the system is kept at a minimum. Still, one may certainly wonder whether supervised peer-to-peer systems are really as scalable as pure peer-to-peer systems on the one hand and as reliable as server-based systems on the other hand. In this chapter, we argue that our approach can result in highly scalable and highly reliable systems.

6.1.1 Motivation

First of all, remember that even pure peer-to-peer systems need some kind of a "rendezvous point", such as a well-known host server [118] or a well-known web-address like gnutellahosts.com, which allows new peers to join the system. The rendezvous point typically does not play any role in the overall topology of the network but just acts as a bridge between new nodes and the existing network. This means that nodes have to self-organize to form an overlay network with good topological properties such as diameter, degree and expansion. In such a scenario, we saw in the last chapter that (a) randomized constructions cannot *guarantee* a good expansion or diameter

and (b) deterministic constructions involve complex balancing schemes [14] to arrive at a good topology.

We show that allowing the supervisor to oversee the topology of the overlay network, apart from working as the rendezvous point, tremendously simplifies the problem of maintaining the above mentioned desirable properties of the peer-to-peer network. Hence, as long as the communication effort of a supervisor for including or excluding a peer is only a low constant, supervised designs should compete well with pure peer-to-peer systems.

Our approach has many interesting applications in the area of grid computing [127, 135, 33], WebTV, and massive multi-player online gaming [49], as outlined in Section 6.7. A supervisor may also serve, for example, as a reliable anchor for code execution rollback, which is important for failure recovery mechanisms such as those used in the Time Warp system [39]. This would make supervised peer-to-peer systems particularly interesting for grid computing [127]. With our concept, supervised peer-to-peer systems can scale to millions of peers without requiring the supervisor to be more powerful than just having a normal workstation with a 100 Mbit/s connection. Also, it is much easier to recover from temporary network partitions with a supervised system than a pure peer-to-peer system. This is useful for systems in which fast recovery is important due to real-time content, such as Internet radio or Internet TV. Finally, though supervised peer-to-peer systems are not as stable as server-based systems with powerful servers, their advantage is that because the supervisor only takes care of the topology but may not be involved at all in peer-to-peer activities, it is from a legal point of view a much safer design than the server-based design.

6.1.2 Our Results

In Section 6.2, we show how to combine known techniques proposed for peer-to-peer systems such as the hierarchical decomposition approach of CAN [125], and the continuous-discrete ap-

proach [110] in a novel way to obtain a general framework for the design of supervised peer-to-peer systems. Our approach requires the supervisor to store a *constant* amount of information about the system at any time and to only send and receive a *low constant* number of messages in order to integrate or remove a peer from the system. We demonstrate our approach by showing how to maintain a supervised hypercube network and a supervised de Bruijn network with it. Our scheme can also be extended to allow concurrent join/leave operations or allow multiple supervisors as outlined in Section 6.4. In order to demonstrate that supervised systems can be made highly scalable, we propose solutions in Section 6.4 that allow a supervisor to serve many join and leave requests concurrently and then extend our basic design to allow multiple supervisors. Afterwards, in Sections 6.5 –6.6 we look at robustness issues. We discuss how our supervised design can be extended to handle random faults. We also present and analyze a simple scheme involving the supervisor so that the resulting network is robust even against adaptive adversarial join/leave attacks, a study recently initiated in [132]. Finally, we discuss in Section 6.7 various applications of our supervised approach.

6.1.3 Related work

Special cases of supervised peer-to-peer systems have already been formally investigated [118, 128, 127], but to the best of our knowledge a general framework for supervised peer-to-peer systems has not been presented yet.

In [118], the authors consider a special node called the *host server* that is contacted by all new peers that join the system. The overlay network maintained by the host server is close to a random-looking graph. As shown by the authors, under a stochastic model of join/leave requests the overlay network can, with high probability, guarantee connectivity, low diameter, and low degree. Alternative designs were later proposed in [128, 127]. In [128] it is shown how to maintain a tree topology using a supervisor for guaranteed broadcasting and in [127] it is shown how to maintain a

supervised overlay network with de Bruijn graph topology for grid computing and load balancing. In this work, we propose a unified model that enables one to create a large class of supervised overlay networks.

Most of the distributed systems are either server-based or peer-to-peer. For example, Napster is rather server-based because all peer requests are handled at a single location. Also systems like SETI@home [135], Folding@home [45], and distributed.net [33] are heavily server-oriented because they do not allow peer-to-peer interactions. Other systems such as the IBM OptimalGrid allow communication between peers but it still uses a star topology and therefore is still closer to being server-based than supervised. Extensive research on computational grids is also done in the Globus Alliance but they do not appear to consider topological designs in their research.

The line of research that is probably closest to our approach is the work on overlay networks in the area of application-layer multicasting. Among them are SpreadIt [31], NICE [9], Overcast [62], and PRM [10], to name a few. However, these systems only focus on specific topologies such as trees, and they do not seem to be generalizable to a universal approach for supervised systems. Other protocols for application-layer multicasting such as Scribe [20], Bayeux [157], I3 [141], Borg [154], SplitStream [21], and CAN-Multicast [126] are rather extensions of a pure peer-to-peer system. For an evaluation of several of these protocols see [22], for example.

6.2 A general framework for supervised peer-to-peer systems

Our general framework for supervised peer-to-peer systems needs several ingredients, including the hierarchical decomposition technique [125], the continuous-discrete technique [110], and the recursive labeling technique. After presenting these techniques we show how to put them together in an appropriate way so that we obtain a universal approach for supervised peer-to-peer systems.

Afterwards, we give some examples that demonstrate how to apply this approach to maintain a supervised hypercubic network and a supervised de Bruijn network.

6.2.1 The hierarchical decomposition technique

Consider any d–dimensional space $U = [0, 1)^d$ for some $d \geq 1$. The *decomposition tree* $T(U)$ of U is an infinite binary tree in which the root represents U and for every node v representing the subcube U' in U, the children of v represent two subcubes U'' and U''', where U'' and U''' are the result of cutting U' in the middle at the smallest dimension in which U' has a maximum side length. Let every edge to a left child in $T(U)$ be labeled with 0 and every edge to a right child in $T(U)$ be labeled with 1. Then the label of a node v, is the sequence of all edge labels encountered when moving along the unique path from the root of $T(U)$ downwards to v. For $d = 2$, the result of this decomposition is shown in Figure 6.1.

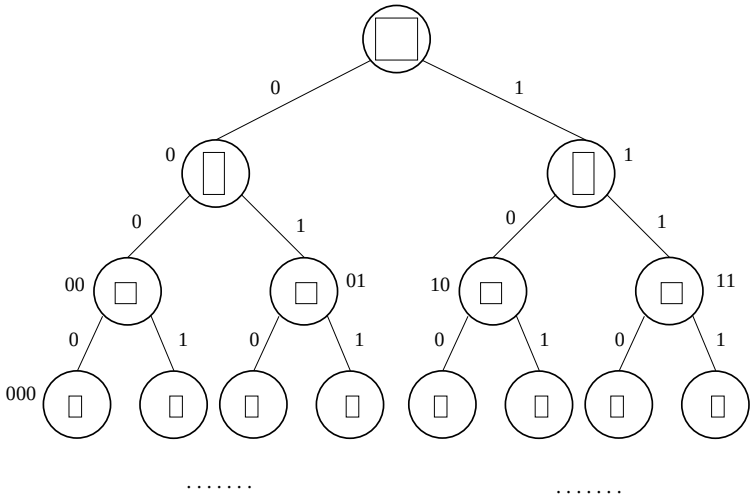


Figure 6.1: The decomposition tree for $d = 2$.

Our goal for the supervised peer-to-peer system will be to map the peers to nodes of $T(U)$ so that

1. the subcubes of the (nodes assigned to the) peers are disjoint,

2. the union of the subcubes of the peers gives the entire set U, and

3. the peers are only distributed among nodes of two consecutive levels in $T(U)$.

The above goals are important for the following reason. Recall the CAN based approach of [125]. The basic idea is to combine an infinite complete binary tree T with a family of graphs $\mathcal{G} = \{\mathcal{G}_\ell | \ell \in \mathbb{N}_0\}$ with $|V(\mathcal{G}_\ell)| = 2^\ell$ for every $\ell \geq 0$. The first two goals are required so that every path down the tree starting with the root contains exactly one peer which is the basic invariant for the CAN-based approach [125]. In order to keep the degree low, a basic goal of the CAN approach is to keep the nodes in as few levels of the tree T as possible. This can be quantized by *level imbalance* being defined as the maximum difference between the levels of the nodes in T. This parameter is called the *global gap* in [2]. The third goal thus asks for an assignment of nodes to levels so that the level imbalance is close to optimal.

Whereas CAN-based peer-to-peer systems usually satisfy the first two properties, they have problems with the third property. For example, using randomized strategies [2, 125] involve advanced techniques such as multiple-choice hashing [107] and result in a level imbalance that is within $O((\log \log n)/\log d)$ for $d \geq 2$. But as we will see, it will be easy for our supervised peer-to-peer approach to also maintain the third property using deterministic strategies.

6.2.2 The continuous-discrete technique

The basic idea underlying the continuous-discrete approach [110] is to define a continuous model of graphs and to apply this continuous model to the discrete setting of a finite set of peers.

Consider any d-dimensional space $U = [0, 1)^d$, and suppose that we have a set F of functions $f_i : U \to U$, $i \geq 1$. Then we define E_F as the set of all pairs $(x, y) \in U^2$ with $y = f_i(x)$ for some i. Given any subset $S \subseteq U$, let $\Gamma(S) = \{y \in U \setminus S \mid \exists x \in S : \ (x, y) \in E_F\}$. We say that (U, E_F)

is *connected* if for any subset $S \subset U$ it holds that $\Gamma(S) \neq \phi$.

Consider now any set of peers V, and let $R(v)$ be the region in U that has been assigned to peer v. Let $G_F(V)$ be the graph with node set V and edge set

$$E(G_F) = \{(v, w) \in V \times V \mid \exists x \in R(v),\ \exists y \in R(w),\ (x, y) \in E_F\}$$

That is, $E(G_F)$ contains an edge (v, w) for every pair of nodes v and w for which there is an edge $(x, y) \in E_F$ with $x \in R(v)$ and $y \in R(w)$. Using the above setting, the following theorem holds:

Theorem 6.2.1 *Suppose that $\cup_{v \in V} R(v) = U$ and (U, E_F) is connected, then also $G_F(V)$ is connected.*

The proof of the above theorem follows from the definitions. Thus, to arrive at a situation where $G_F(V)$ is connected we have to ensure that $\cup_{v \in V} R(v) = U$. But the goals of the hierarchical decomposition technique ensure such an assignment.

Let $\rho = \max_{u,v \in V} |R(v)|/|R(u)|$ be the smoothness [110] of the above assignment scheme. Then, using the properties of the hierarchical decomposition technique it holds that ρ is independent of n and $\rho \leq 2$. Having ρ a constant has nice implications as described in [110] even when considering an arbitrary set F of functions.

6.2.3 The recursive labeling technique

In the recursive labeling approach, the supervisor assigns a *label* to every peer that wants to join the system. The labels are represented as binary strings and are generated in the following order:

$$0, 1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, \ldots$$

Basically, ignoring label 0, when stripping off the least significant bit, the supervisor is first creating all binary numbers of length 0, then length 1, then length 2, and so on. More formally, consider the mapping $\ell : \mathbb{N}_0 \to \{0,1\}^*$ with the property that for every $x \in \mathbb{N}_0$ with binary representation $(x_d \dots x_0)_2$ (where d is minimum possible),

$$\ell(x) = (x_{d-1} \dots x_0 x_d)$$

Then ℓ generates the sequence of labels displayed above. In the following, it will also be helpful to view labels as real numbers in $[0,1)$. Let the function $r : \{0,1\}^* \to [0,1)$ be defined so that for every label $\ell = (\ell_1 \ell_2 \dots \ell_d) \in \{0,1\}^*$, $r(\ell) = \sum_{i=1}^{d} \frac{\ell_i}{2^i}$. Then the sequence of labels above translates into

$$0,\ 1/2,\ 1/4,\ 3/4,\ 1/8,\ 3/8,\ 5/8,\ 7/8,\ 1/16,\ 3/16, \dots$$

Thus, the more labels are used, the more densely the $[0,1)$ interval will be populated. When using the recursive approach, the supervisor aims to maintain the following invariant at any time:

Invariant 6.2.2 *The set of labels used by the peers is $\{\ell(0), \ell(1), \dots, \ell(n-1)\}$, where n is the current number of peers in the system.*

The above invariant is useful for our approach as shown in Section 6.2.4 so that all the three goals of the hierarchical decomposition technique are met.

This invariant is preserved when using the following simple strategy:

- Whenever a new peer v joins the system and the current number of peers is n, the supervisor assigns the label $\ell(n)$ to v and increases n by 1.
- Whenever a peer w with label ℓ wants to leave the system, the supervisor asks the peer with

currently highest label $\ell(n-1)$ to take over the role of w (and thereby change its label to ℓ) and reduces n by 1.

6.2.4 Putting all pieces together

Now we are ready to put the pieces together. We assume that we have a single supervisor for maintaining the overlay network. In the following, the label assigned to some peer v will be denoted as ℓ_v. Given n peers with unique labels, we define the *predecessor* of peer v, denoted $\mathrm{pred}(v)$, as the peer w for which $r(\ell_w)$ is closest from below to $r(\ell_v)$. We define the *successor* of peer v, denoted $\mathrm{succ}(v)$, as the peer w for which $r(\ell_w)$ is closest from above to $r(\ell_v)$ (viewing $[0,1)$ as a ring in both cases). Given two peers v and w, we define their *distance* as

$$\delta(v,w) = \min\{(1 + r(\ell_v) - r(\ell_w)) \bmod 1,\ (1 + r(\ell_w) - r(\ell_v)) \bmod 1\}$$

In order to maintain a doubly linked cycle among the peers, we simply have to maintain the following invariant:

Invariant 6.2.3 *Every peer v in the system is connected to $\mathrm{pred}(v)$ and $\mathrm{succ}(v)$.*

Now, suppose that the labels of the peers are generated via the recursive strategy above. Then we have the following properties:

Lemma 6.2.4 *Let n be the current number of peers in the system, and let $\bar{n} = 2^{\lfloor \log n \rfloor}$. Then for every peer $v \in V$, $|\ell_v| \leq \lceil \log n \rceil$ and $\delta(v, \mathrm{pred}(v)) \in \{1/(2\bar{n}), 1/\bar{n}\}$.*

So the peers are approximately evenly distributed in $[0,1)$ and the number of bits for storing a label is almost as low as it can be without violating the uniqueness requirement.

Now, recall the hierarchical decomposition approach. The supervisor assigns every peer p to the unique node v in $T(U)$ at level $\log(1/\delta(p, \mathrm{pred}(p)))$ with ℓ_v being equal to ℓ_p (padded with 0's to the right so that $|\ell_v| = |\ell_p|$). As an example, if we have 4 peers currently in the system, then the mapping of peer labels to node labels is

$$0 \to 00,\ 1 \to 10,\ 01 \to 01,\ 11 \to 11$$

With this strategy, it follows from Lemma 6.2.4 that all three demands formulated in the hierarchical decomposition approach are satisfied.

Consider now any family F of functions acting on some space $U = [0,1)^d$ and let $C(p)$ be the subcube of the node in $T(U)$ that p has been assigned to. Then the goal of the supervisor is to maintain the following invariant at any time.

Invariant 6.2.5 *For the current set V of peers in the system it holds that*

1. *the set of labels used by the peers is $\{\ell(0), \ell(1), \ldots, \ell(n-1)\}$, where $n = |V|$,*
2. *every peer v in the system is connected to $\mathrm{pred}(v)$ and $\mathrm{succ}(v)$, and*
3. *there are bidirectional connections $\{v, w\}$ for every pair of peers v and w for which there is an edge $(x, y) \in E_F$ with $x \in C(v)$ and $y \in C(w)$.*

6.2.5 Maintaining Invariant 6.2.5

Next we describe the actions that the supervisor has to perform in order to maintain Invariant 6.2.5 during an isolated join or leave operation. For simplicity, we assume that all nodes are reliable and trustworthy and also that peers depart gracefully i.e., they announce their departure to the supervisor. (Non-graceful departures and untrustworthy nodes are treated in Section 6.5). We

also assume that the supervisor can in each round send a message that can contain up to a constant amount of information. We start with the following important fact which can be easily shown.

Fact 6.2.6 *Whenever a new peer v enters the system, then $\mathrm{pred}(v)$ has all the connectivity information v needs to satisfy Invariant 6.2.5(3), and whenever an old peer w leaves the system, then it suffices that it transfers all of its connectivity information to $\mathrm{pred}(w)$ in order to maintain Invariant 6.2.5(3).*

The first part of the fact follows from the observation that when v enters the system, then the subcube of $\mathrm{pred}(v)$ splits into two subcubes where one resides at $\mathrm{pred}(v)$ and the other is taken over by v. Hence, if $\mathrm{pred}(v)$ passes all of its connectivity information to v, then v can establish all edges relevant for it according to the continuous-discrete approach. The second part of the fact follows from the observation that the departure of a peer is the reverse of the insertion of a peer.

Thus, if the peers take care of the connections in Invariant 6.2.5(3), the only part that the supervisor has to take care of is maintaining the cycle. For this we require the following invariant.

Invariant 6.2.7 *At any time, the supervisor stores the contact information of $\mathrm{pred}(v)$, v, $\mathrm{succ}(v)$, and $\mathrm{succ}(\mathrm{succ}(v))$ where v is the peer with label $\ell(n-1)$.*

We now describe how to maintain Invariant 6.2.5 during any join or leave operation.

Join: If a new peer w joins, in order to satisfy Invariant 6.2.7, the following actions are performed. In the following, S denotes the supervisor.

- S informs w that $\ell(n)$ is its label, $\mathrm{succ}(v)$ is its predecessor, and $\mathrm{succ}(\mathrm{succ}(v))$ is its successor.
- S informs $\mathrm{succ}(v)$ that w is its new successor.

- S informs $\mathrm{succ}(\mathrm{succ}(v))$ that w is its new predecessor.
- S asks $\mathrm{succ}(\mathrm{succ}(v))$ to send its successor information to the supervisor, and
- S asks v which is now $\mathrm{pred}(w)$ to send the connectivity information according to F to node w.
- S sets $n = n + 1$.

Leave: If an old node w leaves and reports ℓ_w, $\mathrm{pred}(w)$, and $\mathrm{succ}(w)$ in order to maintain Invariant 6.2.5(3), the following actions are performed. In the following, S denotes the supervisor. Recall that we are assuming graceful departures.

- S informs v (the node with label $\ell(n-1)$) that ℓ_w is its new label, $\mathrm{pred}(w)$ is its new predecessor, and $\mathrm{succ}(w)$ is its new successor.
- S informs $\mathrm{pred}(w)$ that its new successor is v and $\mathrm{succ}(w)$ that its new predecessor is v.
- S informs $\mathrm{pred}(v)$ that $\mathrm{succ}(v)$ is its new successor and $\mathrm{succ}(v)$ that $\mathrm{pred}(v)$ is its new predecessor.
- S asks $\mathrm{pred}(v)$ to send its predecessor information to the supervisor and to ask $\mathrm{pred}(\mathrm{pred}(v))$ to send its predecessor information to the supervisor.
- S asks node v to transfer all of its connectivity information according to F to $\mathrm{pred}(v)$, and
- S sets $n = n - 1$.

Thus, the supervisor only needs to handle a constant number of messages for each arrival or departure of a peer. In fact, at most 8 messages suffice for each operation, and each message is very small. If we assume, for example, that the supervisor has a 100 Mbit/s connection, each message

has a size of 64 bytes, we have 1,000,000 peers in the system, and each peer stays in the system for a minute (on average), then the bandwidth of the supervisor is in principle high enough to handle all of the arrivals and departures (though this would need a high parallelization of the handling of join and leave requests, as discussed in Section 6.4). Moreover, a peer can join and leave the supervised system with a constant number of communication rounds. Hence, our join method is much faster than in pure peer-to-peer systems where the join request of a peer first has to be forwarded to the right location, which usually takes $\Omega(\log n)$ time.

6.3 Examples

In this section we show some examples to illustrate the power of the supervised approach. We show how to maintain dynamic variants of two well- known network topologies, namely the hypercube and the de Bruijn network.

6.3.1 Dynamic Hypercube Network

Consider the d-dimensional Hypercube network with nodes labeled $(x_1, x_2, \ldots, x_d) \in \{0,1\}^d$ and nodes u and v are neighbors if and only if their labels differ in exactly one position.

For a supervised hypercubic network, let $U = [0,1)$ and select F as the family of functions $F_H := \{f_i^-, f_i^+ : U \to U | i \in \mathbb{N}\}$ with

$$f_i^+(x) = (x + (1/2^i)) \bmod 1 \quad \text{and} \quad f_i^-(x) = (x - (1/2^i)) \bmod 1$$

Using the above family of functions, then the neighbors of point $x \in U$ are defined as $\{y | y = f_i^-(x), x_i = 1\} \cup \{y | y = f_i^+(x), x_i = 0\}$.

Using our framework, the following lemma holds by Invariant 6.2.5.

Lemma 6.3.1 *A supervised dynamic hypercubic network can be maintained with $O(1)$ time and work for each (isolated) join and leave request.*

In addition to the above lemma, we now show that the dynamic hypercube has the following topological properties.

Theorem 6.3.2 *At any time, using the supervised framework, a dynamic hypercube can be maintained so that:*

- *the network has a degree of $O(\log n)$*
- *the network has a diameter $O(\log n)$ and,*
- *the network has an expansion $\Omega(1/\log n)$.*

Proof. The bound on the degree follows as each peer v is mapped to an interval $R(v)$ of size in at most $2/\overline{n}$ where $\overline{n} = 2^{\lfloor \log n \rfloor}$ and each function $f \in F_H$ maps an interval I to another interval $f(I)$ of length same as that of interval I. Moreover, once $i \geq 1 + \log \overline{n}$, the interval $f_i(I)$ is contained in the region $R(\mathrm{pred}(v)) \cup R(v) \cup R(\mathrm{succ}(v))$. Hence, the degree of any peer is $O(\log n)$.

For the diameter, we note that for any two points $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ when using F_H, it takes at most k edge traversals to adjust $(x_1, x_2, \ldots, x_k)$ to $(y_1, y_2, \ldots, y_k)$ following the standard bit-flipping scheme of the hypercube. Since x and y may differ in at most $O(\log n)$ bit positions, it follows that the diameter of the dynamic hypercube is $O(\log n)$.

For the expansion, recall that the supervised approach maps peers to at most two consecutive levels of the decomposition tree. If all the peers are mapped to a single level of the tree then the $G_{F_H}(V)$ has a d–dimensional hypercube as a subgraph and hence has an expansion of $\Omega(1/\log n)$. If the peers are mapped to two consecutive levels, then $G_{F_H}(V)$ has a $\log \overline{n}$–dimensional hypercube as a subgraph and hence has an expansion $\Omega(1/\log \overline{n}) = \Omega(1/\log n)$. □

Notice that the expansion we can guarantee using the supervised approach is better than what pure hypercubic peer-to-peer systems like Chord [142] can achieve.

6.3.2 Dynamic de Bruijn Network

Recall the definition of the d–dimensional de Bruijn network where nodes are labeled $(x_1, x_2, \ldots, x_d) \in \{0,1\}^d$ and a node with label $(x_1, x_2, \ldots, x_d)$ has nodes $(0, x_2, \ldots, x_d)$ and $(1, x_2, \ldots, x_d)$ as neighbors.

For a supervised de Bruijn network, consider the space $U = [0,1)$ and select $F_D := \{f_0, f_1 : U \to U\}$ as the family of functions on U with

$$f_0(x) = x/2 \text{ and } f_1(x) = (1+x)/2$$

The family F_D of functions approximate the de Bruijn edges. Hence, when using the supervised framework and maintaining Invariant 6.2.5, the following lemma holds.

Lemma 6.3.3 *A supervised dynamic de Bruijn network $G_{F_D}(V)$ can be maintained with $O(1)$ time and work for each (isolated) join and leave request.*

Moreover, the following theorem can be shown along the lines of Theorem 6.3.2. Notice that the expansion that can be guaranteed is also better than known peer-to-peer systems that are based on the de Briujn network [67, 110].

Theorem 6.3.4 *At any time, using the supervised framework, a dynamic de Bruijn network can be maintained so that:*

- *the network has a degree of $O(1)$*
- *the network has a diameter $O(\log n)$ and,*

- *the network has an expansion* $\Omega(1/\log n)$.

As these examples show, choosing the family of functions F appropriately, various topologies are possible.

6.4 Concurrency

In this section we extend our approach to concurrent join and leave operations and also provide a way to allow multiple supervisors.

6.4.1 Concurrent Join/Leave Operations

In order to be able to handle d join or leave requests in parallel, Invariant 6.2.5 just needs to be extended by one more rule given below. In the following, $\mathrm{pred}_i(v)$ (resp. $\mathrm{succ}_i(v)$) denotes the i^{th} predecessor, (resp. successor), of v on the cycle of nodes. That is, $\mathrm{pred}_0(v) = \mathrm{pred}(v)$ and $\mathrm{pred}_i(v) = \mathrm{pred}(\mathrm{pred}_{i-1}(v))$.

4. Every peer v in the system is connected to its dth predecessor and its dth successor

In addition to this, given that v is the node with label $\ell(n-1)$, Invariant 6.2.7 needs to be extended to:

Invariant 6.4.1 *At any time, the supervisor stores the contact information of v, the $2d$ successors of v, and the $3d$ predecessors of v.*

These invariants can be preserved as follows:

Concurrent Join Operation In the following, let v be the node with label $\ell(n-1)$. Let the d new peers be $w_1, w_2, \ldots w_d$. Then the supervisor integrates w_i between $\mathrm{succ}_i(v)$ and $\mathrm{succ}_{i+1}(v)$ for every $i \in \{1, \ldots, d\}$. As is easy to check, this will violate rule (4) for the $2d$ closest successors of v and the $d-2$ closest predecessors of v. But since the supervisor knows all of these nodes, it can directly inform them about the change. In order to repair Invariant 6.4.1, the supervisor will request information about the dth successor from the d furthest successors from v and will set v to w_d. Thus, we obtain the following result:

Claim 6.4.2 *The supervisor needs at most $O(d)$ work and $O(1)$ time (given that the work can be done in parallel) to process d join operations.*

Concurrent Leave Operation Let the d peers that want to leave the system be $w_1, w_2, \ldots, w_d$. For simplicity, we assume that they are outside of the peers known to the supervisor, but our strategy below can also be easily extended to these cases. The strategy of the supervisor is to replace w_i by $\mathrm{pred}_{2(i-1)}(v)$ for every i. As is easy to check, this will violate rule (4) for the d closest successors of v and the $3d$ closest predecessors of v. But since the supervisor knows all of these nodes, it can directly inform them about the change. In order to repair Invariant 6.4.1, the supervisor will request information about the dth predecessor from the d furthest predecessors from v and their dth predecessors and will set v to $\mathrm{pred}_{2d}(v)$. Thus, we obtain the following result:

Claim 6.4.3 *The supervisor needs at most $O(d)$ work and $O(1)$ time (given that the work can be done in parallel) to process d leave operations.*

6.4.2 Multiple Supervisors

In this section, we show multiple supervisors can work together in maintaining a single supervised peer-to-peer system. We assume that the number of supervisors it not too large so that it is

reasonable to connect them in a clique.

In a network with k supervisors $S_0, S_1, \cdots S_{k-1}$, the $[0,1)$-ring is split into the k regions $R_i = [(i-1)/k, i/k)$, $i \in \{1, \ldots, k\}$, and supervisor S_i is responsible for region R_i. The supervisors are assigned distinct labels s_i which is equal to the binary representation of i using $\lceil \log_2 k \rceil$ bits. For example, with 4 supervisors, the labels of the supervisors are $00, 01, 10$ and 11. Every supervisor manages its region as described for a single supervisor above, with the exception of the borders of its region. The borders are maintained by communicating with the neighboring supervisors on the ring.

Each time a new node wants to join the system via some supervisor S_i, S_i forwards it to a random supervisor who will integrate it into the system. To generate labels for the nodes in the system, supervisor S_i prepends its own label to the labels generated according to section 6.2.3 with the modification that label 1 is the first label to be generated. Thus in a system with 4 supervisors, where supervisor 2 has label $s_1 = 01$ supervisor S_2 generates 0111 as the label for the third node to join the system under S_2. To formalize the above discussion, let n_i be the number of nodes that are being managed by supervisor S_i currently with $\sum_{i=1}^{n} n_i = n$ being the total number of nodes in the system currently. Then, supervisor S_i maintains the following invariant:

Invariant 6.4.4 *The set of labels generated by supervisor i S_i, $i \in [k]$ is*

$$\{s_i \cdot 1, s_i \cdot 01, s_i \cdot 11, s_i \cdot 001, \ldots\}$$

where the $\cdot$ operator denotes binary concatenation.

The above sequence of labels is generated by stripping of the s_i most significant bits and the least significant bit, then supervisor S_i is enumerating all binary numbers of length 0, followed by

length 1 and so on. The mapping $\ell : \mathbb{N}_0 \to \{0,1\}^*$ from section 6.2.3 can then be easily provided.

Supervisor S_i also maintains the invariant that when n_i nodes are in the system managed by S_i then the set of labels used is $\{\ell(0), \ell(1), \ldots, \ell(n_i - 1)\}$. Using techniques from section 6.2.1–6.2.3, it can be shown that supervisor S_i has to only take care of maintaining the doubly linked cycle of peers and the peers have to maintain the connections according to Invariant 6.2.5(3).

Each time a node v under some supervisor S_i wants to leave the system, S_i contacts a random supervisor (which may also be itself) to provide a node that can replace v.

Thus, the join rule provides a random distribution of the peers among the supervisors and it is not too difficult to verify that the leave rule preserves this random distribution. Hence, when using the Chernoff bounds we get the following claim. In the following claim, the phrase high probability refers to a probability that is at least $1 - 1/k^c$ for a constant c.

Claim 6.4.5 *Let n be the total number of nodes in the system. Then it holds for every $i \in [k]$ that the number of nodes currently placed in R_i is in the range $n/k \pm O(\sqrt{(n/k)\log k} + \frac{\log k}{\log\log k})$, with high probability.*

This implies that if n is sufficiently large compared to k, all properties formulated above for a peer-to-peer system with one supervisor can be preserved, including the property that the peers are only distributed among nodes of two consecutive levels of the decomposition tree.

6.5 Robustness against Random Faults

So far we have assumed that the peers announce their departure to the supervisor and thus are said to be graceful. In reality, however, such an assumption cannot be justified as peers may depart ungracefully. In this section, we show how to handle ungraceful departures under the assumption

that ungraceful departures are uniformly distributed among the nodes in the $[0, 1)$ interval and the rate of ungraceful departures is low enough so that the supervisor can handle.

6.5.1 The Random Fault Model

We now introduce the *random fault model* under which we want to show robustness guarantees for the supervised overlay network. To state the model more formally, we let each node in the system have a probability of failure p_i that is independent of any other node in the system. We require that the average failure probability $\overline{p} = \sum_{i=1}^{n} p_i$ be such that $\overline{p}$ is a constant between 0 and 1. Thus, our model does not require that all peers have the same failure probability. Additionally, we require that the failure probabilities are also spread uniformly. Nodes can depart at any time without informing the supervisor about their departure. We say that the network (system) is in a *valid* state if it holds that for m available peers in the network, the m peers occupy positions with labels $\ell(0)$ through $\ell(m-1)$ and all the invariants are met. The goal of the supervisor can be stated as arriving at a valid state in a finite amount of time after all the faults (ungraceful departures) have occurred and in this case we say that the supervisor has been able to *recover* the network.

Towards this goal, the supervisor now maintains the following invariants for $k = c \log n$, for some large constant c. We start with the following notation.

Definition 6.5.1

- *For any $v \in V$, we let* $N_v := \{v\} \cup \{\mathrm{pred}_i(v) | i = 1, 2, \cdots, k\} \cup \{\mathrm{succ}_i(v) | i = 1, 2, \cdots, k\}$.
- *For any $V' \subseteq V$, we let* $R(V') := \cup_{v \in V'} R(v)$.

Finally, recall from Section 6.2.2 that $\Gamma(s)$ for any $S \subseteq U$ refers to the neighbors in $U \setminus S$ of nodes in S according to E_F.

Invariant 6.5.2 *Every node v is connected to:*

- $\mathrm{pred}_i(v)$ *and* $\mathrm{succ}_i(v)$ *for* $i \in \{1, 2, \ldots k\}$*, and*
- *all nodes w such that* $\Gamma(R(N_v)) \cap \Gamma(R(N_w)) \neq \phi$*.*

The above connectivity rules introduce a k–wise redundancy in the system as each node maintains connections to nodes in its k-neighborhood. The supervisor stores the contact information according to the following invariant.

Invariant 6.5.3 *The supervisor maintains the following connections.*

- **Join connections:** *These are to the $2k$ successors and $3k$ predecessors of the node v with label $\ell(n-1)$. These connections are similar to the connections specified by Invariant 6.4.1.*
- **Repair connections:** *These are to some peer w, the k closest predecessor positions of w, and the k closest successors positions of w. By the predecessor (successor) positions of a node w we mean the positions in the unit interval that precede (*resp. *succeed) node w and may or may not be occupied by any peer currently in the system.*

The join and leave operation are now extended as follows. To insert a new w into the system, the supervisor assigns a label to w and proceeds according to a normal join operation in section 6.2.5 and also satisfying invariant 6.5.2. To maintain Invariant 6.5.3, the supervisor updates its join connections accordingly by requesting relevant information. The leave operation of a gracefully departing w now follows similarly to that of a basic leave operation by the supervisor reversing the last join operation.

Thus the supervisor has to maintain only a logarithmic amount of information. The cost of join and leave operation increases to $O(\log n)$ from a constant. As the size of the network increases

or decreases by a factor of 2, the supervisor updates the value of k accordingly by a factor of $\pm c$. The supervisor also updates its repair connections if w or any successor/predecessor of w departs by choosing the closest successor/predecessor position.

The above operations along with the invariants have the following property under the random fault model. We start with the following lemma.

Lemma 6.5.4 *Under the random fault model, consider any set S of k consecutive positions $q_1, q_2, \cdots, q_k$ where peer v is chosen for position i independently and uniformly at random from the n peers in the network. Then, the probability that either all positions are occupied or none of the positions are occupied by a peer is polynomially small.*

Proof. Let X_i be a random variable defined so that $X_i = 1$ if and only if the peer in position q_i fails and 0 otherwise. It holds that:

$$E[X_i] = \Pr[\text{peer in position } q_i \text{ fails}] = \sum_{j=1}^{n} \Pr[\text{peer } j \text{ is in position } q_i] \cdot p_j = \overline{p}$$

Let the random variable $X := \sum_{i=1}^{k} X_i$. As the X_i's may not be independent, we may not use Chernoff bounds on X. However, it can be shown that the random variables X_i's are negatively correlated as follows.

$$\Pr[X_i = 1 | X_1 = 1, X_2 = 1, \cdots, X_{i-1} = 1] \leq \frac{n}{n-k} \cdot \overline{p} \leq \left(1 + \frac{k+1}{n}\right) \overline{p} =: \overline{p}_u$$

The above probability represents the increase in the average failure probability of peer assigned to position q_i in the set S under the worst-case scenario that all the previous positions in the set S have failed peers.

Thus, it holds that for any subset $S' \subseteq \{1, 2, \cdots, k\}$,

$$E[\Pi_{i \in S'} X_i] = \Pr[\wedge_{i \in S'} X_i = 1] = \Pi_{i \in S'} \Pr[X_i = 1 | X_1 = 1, X_2 = 1, \ldots, X_{i-1} = 1] \leq \Pi_{i \in S'} \overline{p}_u$$

Hence, the random variables X_i's can be seen as negatively correlated with probability $\overline{p}_u$ and we can use the Chernoff bounds on X to show that the event that all peers in S fails, i.e. the event $\{X \geq (1+\delta)k\overline{p}_u\}$, has a polynomially small probability since $\overline{p}$ is bounded from above by a constant.

Similarly one can show that the random variables X_i's are positively correlated as follows:

$$Pr[X_i = 1 | X_1 = 1, X_2 = 1, \cdots, X_{i-1} = 1] = \frac{n}{n-k} \cdot \overline{p} \geq (1 - (k-1)/n)\, \overline{p} =: \overline{p}_\ell$$

For any $S' \subseteq \{1, 2, \cdots, k\}$, it then holds that

$$E[\Pi_{i \in S'} X_i] = \Pi_{i \in S'} \Pr[X_i = 1 | X_1 = 1, X_2 = 1, \ldots, X_{i-1} = 1] \leq \Pi_{i \in S'}\, \overline{p}_\ell$$

This means that the random variables $X_1, X_2, \cdots X_S$ are positively correlated with probability $\overline{p}_\ell$ allowing one to use Chernoff bounds on X (cf. [131, Lemma 1.41]) to show that the the event that none of the peers in S fail, i.e., the event $\{X \leq (1-\delta)k\overline{p}_\ell\}$, has a polynomially small probability, since $\overline{p}$ is bounded from above by a constant. □

Theorem 6.5.5 *When considering the random fault model, the supervisor can recover the network in a finite amount of time after all the faults have occurred.*

Proof. To prove the theorem, first notice that the supervisor always knows the positions in the cycle that should be occupied as this is precisely the positions with a label among $\ell(0)$ to $\ell(n-1)$. From

the above lemma it holds that in any region R, with high probability at least one position has a failed peer and not all positions have failed peers. Thus, the supervisor can always make progress during the repair process by the way the repair connections are updated. Also, by performing one complete tour of the unit interval, the supervisor can bring the network to a valid state. □

For the dynamic setting that faults may occur while the supervisor is in the middle of a repair phase, we argue that in any time interval T large enough so that the supervisor can complete a entire tour around the $[0, 1)$ interval and so that the average error probability stays as a constant between 0 and 1 and the faults are evenly distributed around the unit interval. Then, during any such interval T, according to the above lemma, the supervisor never encounters the situation that it cannot make any progress while processing ungraceful departures.

6.6 Robustness against Adaptive Adversarial Attacks

In this section we describe a simple scheme to guarantee robustness against even adaptive adversarial join/leave attacks. Due to the presence of supervisor, our scheme for providing robustness under an adaptive adversary is surprisingly simple.

While the results of the previous section guarantee that the system is robust to random node failures, the system is not robust against adaptive adversarial attacks. Such attacks take the form of adversarial nodes that can join and leave the system as many times as they wish. In our system, adaptive adversarial attacks can easily disconnect the supervisor from the rest of network by taking positions that the supervisor is connected to. This would then make it difficult for new peers to join the system. The adversary can also place nodes at critical positions so that routing in the network is disrupted by not forwarding the packets or forwarding them to the wrong location or by injecting lots of packets destined for other adversarial nodes so that the network would be heavily

congested. These type of attacks are recently studied in [132] by showing how to maintain a robust ring network of nodes under the presence of such a powerful adversary. While mechanisms for other network topologies are not known, using the supervised approach we show how to extend our basic scheme to provide robustness guarantees for any overlay network under the presence of an adaptive adversary.

Formally, we allow the adversary to own up to ϵn of the n nodes in the system for some sufficiently small constant $\epsilon > 0$. These nodes are also called *adversarial* nodes and the rest are called *honest* nodes. The supervisor and the honest nodes are oblivious to adversarial nodes, i.e., there is no mechanism to distinguish at any time whether a particular node is honest or not. To achieve robustness in the presence of an adaptive adversary, we use the following scheme.

In the following, a *region* is an interval of size $1/2^i$ in $[0, 1)$ starting at an integer multiple of $1/2^i$ for some $i \geq 0$, and a node v belongs to a region R if $r(\ell_v) \in R$. Recall that $\overline{n} = 2^{\lfloor \log n \rfloor}$. The supervisor organizes the nodes into regions so that each region contains between $c \log \overline{n}$ and $2c \log \overline{n}$ nodes for some constant $c > 1$. Whenever these bounds are violated in a region, the supervisor splits it or merges it with a neighboring region. The n nodes are also organized into 5 sets S_1 to S_5 and the following invariant is maintained for these sets.

Invariant 6.6.1 *At all times,*

1. S_1 *has* $\bar{n}/8$ *nodes with labels* $\ell(0), \ell(1), \ldots, \ell(\overline{n}/8 - 1)$.

2. S_2 *has* $\overline{n}/8$ *nodes with labels* $\ell(\overline{n}/8), \ell(\overline{n}/8 + 1), \ldots, \ell(\overline{n}/4 - 1)$.

3. S_3 *has* $\overline{n}/4$ *nodes with labels* $\ell(\overline{n}/4), \ell(\overline{n}/4 + 1), \ldots, \ell(\overline{n}/2 - 1)$.

4. S_4 *has* $\overline{n}/2$ *nodes with labels*$\ell(\overline{n}/2), \ell(\overline{n}/2 + 1), \ldots, \ell(\overline{n} - 1)$.

5. S_5 *has the remaining* $n - \overline{n}$ *nodes with labels* $\ell(\overline{n}), \ell(\overline{n} + 1), \ldots, \ell(n - 1)$.

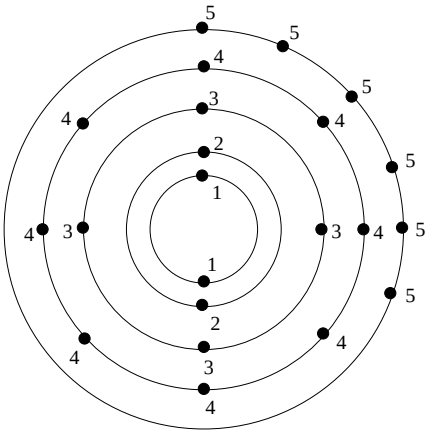


Figure 6.2: Logical organization of nodes into five sets. The number against node position indicates the set to which the node belongs to.

The following invariant describes the connections maintained by the nodes in the various sets and the connections maintained by the supervisor. To simplify notation, for a real number $x \in [0, 1)$, $R(x)$ is the region that x belongs to and $S_i(R)$ is the set of S_i-nodes belonging to R. For every region R, let $S_R = S_1(R) \cup S_2(R)$ and $\bar{S}_R = S_3(R) \cup S_4(R) \cup S_5(R)$ if R precedes $R(r(\ell(n)))$ and otherwise, $S_R = S_1(R)$ and $\bar{S}_R = S_2(R) \cup S_3(R) \cup S_4(R) \cup S_5(R)$. For every region R let $M_R = S_1(R) \cup S_2(R) \cup S_3(R)$.

Invariant 6.6.2 *For all regions R, every S_R-node is connected to all nodes in $S_R \cup \bar{S}_R$. Every S_R-node is also connected to all nodes in the predecessor and successor regions of R, denoted* $\mathrm{pred}(R)$ *and* $\mathrm{succ}(R)$*, and for every $u \in S_R$ that has a connection to a node $v \in S_{R'}$ according to Invariant 6.2.5(3), all S_R-nodes are connected to all $S_{R'}$-nodes.*

The supervisor is connected to all the nodes in S_R in the regions $R(r(\ell(n-1)))$, $\mathrm{pred}(R(r(\ell(n-1))))$ *and* $\mathrm{succ}(R(r(\ell(n-1))))$*. The above connections are called* join connections*.*

The supervisor is connected to all the nodes in M_R for some region R, and to a special node $v^* \in M_R$. These connections are called mixing connections*.*

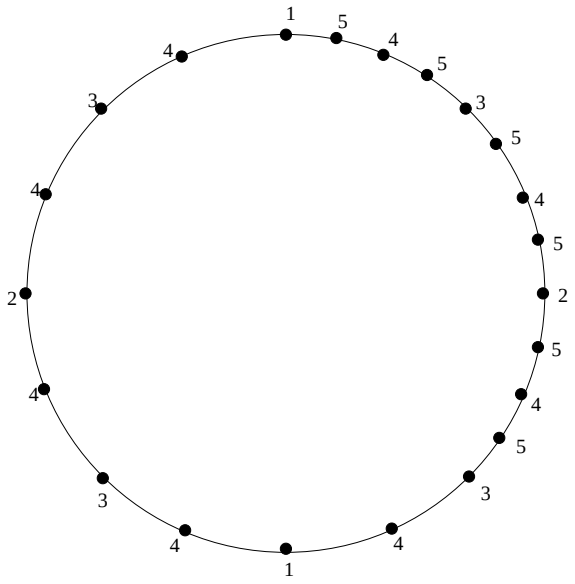


Figure 6.3: Physical organization of nodes into five sets.

Figure 6.2 shows the logical organization of the nodes and the sets S_1 through S_5 and Figure 6.3 shows the physical organization. Our organization of the nodes ensures that in a constant fraction of the network, the adversarial nodes cannot influence the network behavior.

The set S_1 is also referred to as the *stable* set. The goal of the supervisor is to have the honest nodes in the majority in every set $S_1(R)$ of size $c \log n$ nodes, with high probability. The reason for this goal are stated shortly.

The set S_2 is in a stage called the *split-and-merge* stage because S_2-nodes are merged into the stable set or removed from it as nodes join or leave the system. The set S_3 is in a stage called *mixing* stage in which the supervisor performs transpositions according to a uniformly chosen permutation to ensure that the nodes are well-mixed before being integrated into the stable set.

The set S_4 is in a *reservoir* stage. S_4 is used to fill departed positions in the sets S_1 to S_3 by selecting random nodes in S_4 and filling their positions with the last nodes in S_5. Finally, the set S_5 is in a *filling* stage where new nodes are added by assigning them the label $\ell(n)$.

The join and leave operation have to be extended so that the supervisor can ensure the majority condition at all times. We first describe the modifications to the join/leave operations.

Join:

The supervisor assigns to the new node the label $\ell(n)$ and integrates it so that the Invariants 6.6.1 and 6.6.2 are satisfied.

Afterwards, the supervisor updates v^* to be successor of v^* among the nodes in M_R. If there is no such node, then M_R is updated to the $M_{R'}$ where R' is the region succeeding region R and v^* is taken to be the first node in $M_{R'}$. Suppose that v^* belongs to the set S_i in M_R for a region R. Then the supervisor picks a node w with position between v^* and 1 (exclusive) uniformly at random and exchanges the positions of w and v^*. This is realized by the supervisor informing all nodes in $S_1(R(\ell(n)))$ the positions of nodes v^* and w so that this is reliably done without involving the supervisor.

Each time a new node causes the supervisor to switch from a region R to $\mathrm{succ}(R)$, the nodes in $S_2(R)$ are merged into $S_1(R)$ as prescribed by Invariant 6.6.2.

Observe that during the join operation, nodes in M_R undergo transpositions that has the effect of permuting the nodes according to a permutation chosen uniformly at random from the set of all permutations of size $|M_R|$. This is crucial to guarantee robustness as shown in the following result. Thus, once a pass has been made through all positions of $S_1 \cup S_2 \cup S_3$, the positions in $S_i, i \in \{1, 2, 3\}$ form a random permutation.

Leave:

If a node v leaves with $v \in S_4 \cup S_5$, the supervisor simply replaces it by the last node in S_5. Otherwise, the supervisor replaces v by a random node in S_4 and fills the position of that random

node with the last node in S_5. This is followed by performing a mixing operation similar to that done during a join operation. (The supervisor initiates the leave operation for v only if a majority of S_1-nodes in v's region notify it about that. In this case, the supervisor has the necessary information to correctly initiate the replacement.) Each time a departure causes the supervisor to switch from a region R to $\mathrm{pred}(R)$, the nodes in $S_2(\mathrm{pred}(R))$ are split away from $S_1(R)$ as prescribed by Invariant 6.6.2.

Majority Condition

The goal of the above scheme of the supervisor is to ensure that in any region of R of logarithmic size, the honest nodes are in a majority with high probability. This condition is referred as the *majority condition* and is useful in the following way. Suppose the majority condition holds. Then quorum strategies can be used to wash out adversarial behavior as follows. Consider any set T of $c \log n$ nodes in S_1. According to Invariant 6.6.2, all the honest nodes in T are connected to all nodes in T. To perform any network operation such as finding the node with a given label, all the nodes in T perform majority voting. The outcome of the operation is determined uniquely if a majority of the nodes in T agree on the outcome. This means that for adversarial nodes to have any influence on the outcome of a network operation, they should be in a majority in the set T of $c \log n$ nodes since we assume that honest nodes act honestly.

6.6.1 The Semi-adaptive Model

Before we proceed further, we outline the way in which the nodes join/leave the system. We start by considering a model similar to that [132] where honest nodes do not leave the system and only adversarial nodes may join/leave the system in an adaptive manner. Certainly this is a simple model but is very illustrative.

Theorem 6.6.3 *For a sufficiently small constant $\epsilon > 0$ it holds that as long as the adversary owns at most ϵn nodes, the above scheme guarantees that in every region R of size $c \log n$ for $c \geq 1$, the honest nodes are in the majority in $S_1(R)$, with high probability.*

Proof. We first consider the following random experiment. Consider m balls placed in m bins such that there is exactly one ball in each bin. The balls are labeled uniquely from 1 through m and the bins are numbered from 1 through m. Now, the ball in bin 1 is switched with the ball in a bin chosen uniformly at random from bins 1 through m. This is followed by switching the ball in bin 2 with the ball in a bin chosen uniformly at random from the bins 2 through m. This is continued until we visit the ball in bin $m - 1$. Also, the choice of bin at any time is independent of the previous choices. This random experiment creates a random permutation of m balls in the bins as it holds that:

- Every permutation is an outcome of the random experiment, i.e., any permutation of the balls assigned to bins can be produced by the above experiment, and
- Any permutation, or outcome of the random experiment, is equally likely with a probability of $1/m!$.

Consider the basic model where only adversarial nodes may join/leave the system. After n join/leave operations the effect of the mixing operations is the same as that of choosing a random permutation of size $|S_1 \cup S_2 \cup S_3|$. It then follows that as nodes in S_1, S_2 and S_3 are permuted during the mixing operations R, we arrive at a situation where given any position in S_i, for $i \in \{1, 2, 3\}$, the probability that the node at that position is an adversarial node is at most $\frac{\epsilon n}{\sum_{i=1}^{3} |S_i|} \leq 4\epsilon/3$.

We prove the theorem in this case by considering any fixed set $T = S_1(R)$ of $c \log n$ positions in a region R. Given that any position in T is occupied by an adversarial node with probability close to ϵ, the probability that a majority of the positions in T are occupied by adversarial nodes is

at most:

$$\sum_{k=|T|/2}^{|T|} \binom{|T|}{k} (4\epsilon/3)^k \leq 2^{|T|-1}(4\epsilon/3)^{|T|/2} \leq (16\epsilon/3)^{c\log n/2} \leq 1/n^4$$

if $c \geq 3$ and $\epsilon < 1/6$. Since there are at most $n/c\log n$ regions of size $c\log n$, the probability that for some such region the majority condition is violated is at most $1/n^3$ using Boole's inequality. □

We now extend the model to allow also honest nodes to leave the system but so that the leave operations of the honest nodes are spread uniformly around the $[0,1)$ interval. This means that the adversary cannot issue join/leave requests for honest nodes adaptively. The same proof extends easily to the case where honest nodes may also leave the system but such leave operations are spread uniformly in the $[0,1)$ interval. In this case, it holds that in any region R, the expected number of leave operations required so that a majority of the honest nodes leave the system is $\Theta(n)$. During each such leave operation the probability that the position is replaced by an adversarial node is $O(\epsilon)$. Since after $O(n)$ operations, the nodes in R are replaced by the transpositions, having ϵ low enough will ensure that the majority condition holds for every region R with high probability.

6.6.2 The Fully Adaptive Model

Finally, consider the situation where the adversary can force honest nodes to leave in a non-uniform or adaptive manner. We call this the *fully adaptive model*. In this case it is possible that from a given region, a majority of the honest nodes are made to leave. This case captures the most difficult scenario for distributed systems as it becomes difficult to ensure uniform spreading of adversarial nodes. The reason for this is that using the above scheme, the following sequence of join/leave operations violate the majority condition for some region R. Fix any region R and assume that R has no adversarial nodes at this time. Now the adversary can force honest nodes from R to leave successively. During each such leave operation, the probability that an adversarial

node enters R is $\Theta(\epsilon)$. So after $\Theta(\log n)$ leave operations of honest nodes from R, the expected number of adversarial nodes in R is $\Theta(\epsilon|R|)$. By repeating the scheme for $\Theta(\log n)$ times, where during these operations no adversarial node leaves R, the majority condition for R can be violated. Note that during these $\Theta(\log^2 n)$ nodes in R may be part of exchange operations initiated outside of R. But the probability of this is only $\Theta(\log n/n)$, which is small enough so that the above scheme is not affected.

What led to the failure of the existing scheme is that the supervisor does not have a chance to make any transpositions in region R until $\Theta(n)$ join/leave operations have occurred. This presented a window for the adversarial nodes to gain a majority in region R as the honest nodes leave en-masse.

The Modified Leave Operation

We proceed as follows during the leave operation. Recall that the leave operation of node v with a position in $S_1 \cup S_2 \cup S_3$ involves replacing the position of v with a node w chosen from S_4 uniformly at random. The supervisor now maintains a special position p^* in every region R. During every leave operation of a node with a position in R, the supervisor also exchanges the node in position p^* with that of another node w' chosen independently and uniformly at random from the nodes in S_4. Position p^* is then updated to be the successor of position p^* among the positions in the region R. If there is no successor position of p^* in region R, then $p^*$is taken to be the first node in R.

With this modified leave operation we now show the following theorem similar to Theorem 6.6.3.

Theorem 6.6.4 *In the fully adaptive model, for a sufficiently small constant $\epsilon > 0$ it holds that as*

long as the adversary owns at most ϵn nodes, the above scheme guarantees that in every region R of size $c \log n$ for $c \geq 1$, the honest nodes are in the majority in $S_1(R)$, with high probability.

Proof. In addition to the proof of Theorem 6.6.3, we also consider the case that honest nodes may leave from any given region R.

Consider any region R. Denote by the *age* of node v in R as the number of leave operations from R during which node v is not replaced. Upon entering the region R, node v has age 0 and during every time step that v is still in R, the age of v increases by 1. It then holds that the age of any node in R while using the modified leave operation is at most $|R|$. The reason for this is that after $|R|$ more leave operations, node v is certainly replaced by another node via an exchange operation.

It also holds that during any leave operation from R, the probability that the node in position p^* is replaced by an adversarial node is at most $\frac{\epsilon n}{|S_4|} \leq 2\epsilon$. It then follows that the expected number of adversarial nodes in R due to $|R|$ leave operations is at most $2\epsilon|R|$. Since the actions of leave operations are independent, one can use Chernoff bounds to show that the majority condition holds for region R with high probability when ϵ is sufficiently small. □

Notice that the modified leave operation is actually replacing nodes in $S_1 \cup S_2 \cup S_3$ with nodes in S_4. This may sound artificial but the following counter-example suggests that replacing the node at position p^* with that of another node at a position chosen uniformly at random from the positions in $S_1 \cup S_2 \cup S_3$ still allows the adversary to gain a majority in some region R as follows. Fix a region R. It holds that there will be on expectation $\Theta(n/\log n)$ regions, excluding R, where the position p^* is currently occupied by an adversarial node. Now the adversary issues a leave request from one of these regions. The probability that an adversarial nodes then enters the region R is at least $\Theta(1/n)$. But, using the scheme repeatedly, while not disturbing the region R, the adversary

can gain a majority in R. Hence, we have to exchange the node at position p^* with a node from S_4. This attack can be seen as an indirect attack on region R.

It is worth noting that such a strong guarantee can be provided with a simple scheme. The amount of information the supervisor has to maintain is only logarithmic. The analysis is also not as complicated as that of [132] and the presence of a supervisor limits the ability of the adversary even with adaptive join/leave attacks. The simplification results from the fact that nodes in S_1, S_2 and S_3 are isolated from the node join/leave operations allowing the supervisor to permute the nodes before integrating them into the existing network.

6.7 Applications

We now discuss some applications of the supervised overlay networks that arise in the area of distributed computing.

6.7.1 Grid Computing

Recently, many systems such as SETI@home [135], Distributed.net [33] have been proposed for distributed computing. A main drawback of such systems is that the topology of the system is a star graph with the central server maintaining a direct connection to each client. Such a topology imposes heavy demands on the central server. Instead, we can use the basic approach of Section 6.2 to design a overlay network for distributed computing. Peer-to-peer connections allow subtasks to be spawned without the involvement of the supervisor so that the demands on the server can be significantly reduced. This is particularly interesting for distributed branch-and-bound computations as was discussed in [127].

6.7.2 WebTv

Our approach can also be used in Internet applications such as WebTv. In such an application, there are typically various channels that users can browse or watch while being connected to the Internet. The number of channels ranges in the scale of hundreds while the number of users can range in the scale of millions. Such a system should allow users to quickly zap through channels. Hence, such a system should allow for rapid integration and be scalable to large number of users. Our supervised overlay networks can easily achieve such a smooth operation. Suppose that every channel has a supervisor, each supervisor maintains its own broadcast network, and the supervisors form a clique. Then it follows from our supervised approach, which can handle join and leave operations in constant time, that users browsing through channels can be moved between the networks in a very fast way, comparable to server-based networks, so that users only experience an insignificant delay.

6.7.3 Massive Multi-player Online Gaming

Distributed architectures for massive multi-player online gaming (MMOG) are being studied recently (see e.g., [49]). The basic requirements of such a system includes authentication, scalability, and rapid integration. Traditionally, such systems have been managed by a central server that takes care of the overall system with limited communication between the users. As can be seen, such a system will not be scalable and also might experience heavy congestion at the central server. Hence, distributed architectures are required at a certain scale. A supervised overlay network naturally satisfies the requirements. Authentication of entities can be done by the supervisor (or multiple supervisors) and the system stays highly scalable because of the relatively low load on the supervisor. Rapid integration is also possible since the supervisor can handle integration of new peers

(players in this setting) with a constant number of communication rounds.

Typically in a MMOG, it is possible to partition the virtual world into what are called *locales*. In an architecture based on supervised overlay networks then it would be possible to have one (or more) supervisor to be responsible for each such locale. Whenever a player moves between locales, the supervisor can coordinate the join/leave of the player quickly. Also, based on number of players based at various locales, the supervisors can be distributed so that the load at the supervisors stays balanced.

6.8 Chapter Summary

In this chapter, our goal is to design highly scalable and highly reliable systems. We proposed a method, supervised peer-to-peer systems, that inherits the advantages of both peer-to-peer systems and centralized server-based systems. Our unified scheme for building a supervised peer-to-peer system from a large class of topologies is not very complicated.

We also showed that robustness guarantees under a strong adversarial model can also be provided with small modifications to the basic design. Our construction falls under the category of *pro-active* approaches to providing robustness guarantees, as opposed to *reactive* approaches which take some action only when certain conditions are met. Compared to the approach of [132], the scheme presented in Section 6.6 is much simpler owing to the presence of the supervisor.

We note that our supervised peer-to-peer system can be easily extended to function as a distributed hash table (DHT) as shown in [127]. While [127] focuses on networks with a de Bruijn topology, the results can be easily extended for any topology. Also, our design guarantees that the load among the peers is balanced up to a low constant factor in expectation, when using known techniques [69] of hashing data using a (pseudo)-random hash function.

A preliminary version of the results in this chapter appeared in [79].

Part III

Wireless Ad hoc Networks

Chapter 7

Wireless Ad Hoc Networks: Model and Spanner

In this chapter, we consider the problem of designing overlay networks for wireless ad hoc networks. While many local control algorithms have been already suggested in the literature, most of them are based on an oversimplified model of wireless communication. We first suggest a model that is much more general than previous models. It allows the path loss of transmissions to significantly deviate from the idealistic unit disk model and does not even require the path loss to form a metric. Also, our model is apparently the first proposed for algorithm design that does not only model transmission and interference issues but also aims at providing a realistic model for physical carrier sensing. Physical carrier sensing is needed so that our protocols do not require *any* prior information (not even an estimate on the number of nodes) about the wireless network to run efficiently.

Based on this model, we propose a local-control protocol for establishing a constant density spanner among a set of mobile stations (or *nodes*) that are distributed in an arbitrary way in a 2-dimensional Euclidean space. More precisely, we establish a backbone structure by efficiently electing cluster leaders and gateway nodes so that there is only a constant number of cluster leaders and gateway nodes within the transmission range of any node and the backbone structure satisfies

the properties of a topological spanner.

Our protocol has the advantage that it is locally self-stabilizing, i.e., it can recover from *any* initial configuration, even if adversarial nodes participate in it, as long as the honest nodes sufficiently far away from adversarial nodes can in principle form a single connected component. Furthermore, we only need constant size messages and a constant amount of storage at the nodes, irrespective of the distribution of the nodes. Hence, our protocols would even work in extreme situations such as very simple wireless devices (like sensors) in a hostile environment.

7.1 Introduction

An important problem for wireless ad hoc networks has been to design overlay networks that allow time- and energy-efficient routing. Many local-control strategies for maintaining such overlay networks have already been suggested, but mostly high-level wireless models have been used for their analysis. However, since mobile ad-hoc networks have many features that are hard to model in a clean way, it is not clear how well these strategies may actually perform in practice. Major challenges are how to model wireless communication and how to model mobility. Here, theoretical work is still in its infancy. So far, people in the algorithms community have mostly looked at *static* wireless networks (i.e. the wireless stations are always available and do not move). Even in such static situation, modeling wireless networks is difficult due to the following reasons.

- The area over which a wireless node can transmit messages successfully, called the *transmission range* and denoted by r_t, can be of arbitrary shape due to the characteristics of the environment, transmission power, and other factors. This makes it very difficult to model the concept of message transmission accurately.

- Wireless devices are prone to interference problems. This can be caused by wireless devices

transmitting simultaneously or due to external factors. Just like the transmission range, the area over which a wireless node can cause interference, called the *interference range* and denoted by r_i, can also be of arbitrary shape. Usually, the interference range is bigger than the transmission range.

- Wireless devices often use their ability to sense the carrier before transmitting. This is called *physical carrier sensing*. Physical carrier sensing is needed so that protocols do not require *any* prior information (not even an estimate on the number of nodes) about the wireless network to run efficiently. However, modeling physical carrier sensing has not been done in the theoretical community.

A variety of models are proposed in the literature for wireless networks. Below, in Section 7.2 we first review the models so far proposed and point out some of their short-comings. We then introduce our new model for wireless networks in Section 7.3. In Section 7.4 we define the spanner problem and make some initial observations. Section 7.5 discusses related work for overlay constructions for wireless networks. In Section 7.6 a brief overview of our entire protocol is presented. In Sections 7.7–7.8 we present and analyze our three phase protocol. The chapter ends with a summary and acknowledgements.

7.2 Models of Wireless Networks

7.2.1 Unit Disk Graph (UDG) model

By far the easiest model of wireless models is called the unit disk graph model (see e.g., [47, 146, 138, 85]). In this model, all wireless nodes are assumed to have the same transmission range R. The neighborhood of any node thus consists of all other nodes that are within a distance

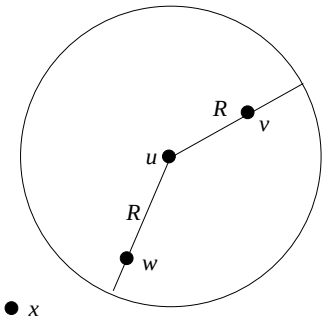


Figure 7.1: Neighborhood of node u according to UDG model.

R and all links are bidirectional. When the transmission range is scaled to 1, we get the following definition for unit disk graph.

Definition 7.2.1 (Unit Disk Graph) *The graph $G = (V, E)$ with V being a set of wireless stations located in a 2-dimensional Euclidean plane and for any $u, v \in V$, the edge $(u, v) \in E$ if and only if the Euclidean distance between u and v is at most 1 is called the unit disk graph corresponding to V.*

Transmission of messages are said to *interfere* at a node if at least two of its neighbors transmit at the same time. A node can only receive a message if it does not interfere with any other message. Figure 7.1 shows the neighborhood of node u according to the UDG model where node u has node v and w as neighbors but not node x, and node u can interfere at nodes v and w.

7.2.2 Packet Radio Network (PRN) model

In the packet radio network model (see e.g., [30, 80, 82, 81]), the network is modeled as a graph and the wireless units, or nodes, form the vertex set, and two vertices are connected by an edge if and only if the corresponding wireless nodes are within the transmission range of each other. Thus, this model removes the assumption that all nodes have the same transmission range and also does not rely on Euclidean distances to model the transmission range. The interference model is

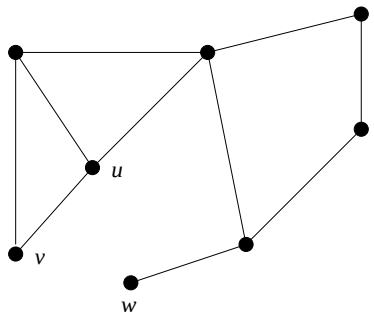


Figure 7.2: Neighborhood of node u according to PRN model.

the same as that of the UDG model. Figure 7.2 shows the graph obtained by using the PRN model where node u has node v as a neighbor but not node w, and node u can interfere at node v but not node w.

The packet radio network model is a simple and clean model that allows one to design and analyze algorithms for overlay networks with a reasonable amount of effort. However, since it is a high-level model, it does have some serious problems with certain scenarios in practice. In reality, the transmission range of a message is not the same as its interference range and for the network in Figure 7.2, it is possible that node w may still interfere at node u even though they are not neighbors. There are other serious problems due to interference that this model suffers from, which we discuss in Chapter 8.

7.2.3 Transmission, Interference Model

One of the drawbacks of the PRN model is that it models the interference range to be the same as that of the transmission range. However, in reality, the interference range of a node is usually bigger than its transmission range. There are a limited number of papers that use a model that differentiates between the transmission range and the interference range , see e.g,. [4, 55, 56, 57, 83]. In these models, in general, the interference range is taken to be a constant factor bigger than that of the transmission range. This is shown in Figure 7.3 where node u has nodes v and w as neighbors,

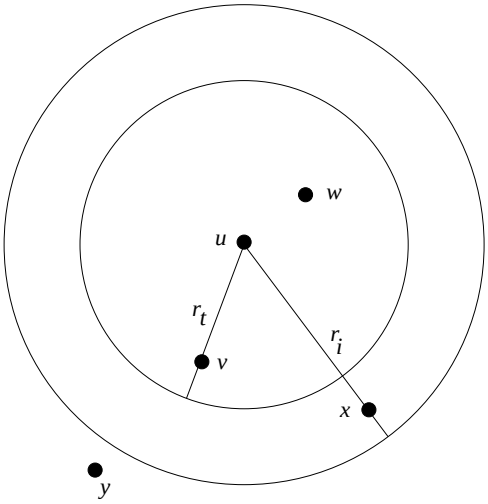


Figure 7.3: The general transmission, interference model.

node u can interfere at node x and node u cannot interfere at node y. As shown, these models still assume a disk model in a sense that the transmission range and interference range can be modeled by two distance values that hold irrespective of the position of a node. Thus, this model can be seen as an extension of the UDG model to handle a bigger interference range. We propose a more general model.

7.3 A new model for wireless communication

In order to motivate our model, we first review some commonly used transmission techniques in wireless communication. We will concentrate here on the IEEE 802.11 standard because IEEE 802.11-based radio LANs are currently dominating the market and will most probably do so also in the future. The IEEE 802.11 standard distinguishes between a Physical (PHY) layer and a Medium Access Control (MAC) layer for the transmission of messages. The 802.11 MAC protocols are based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).

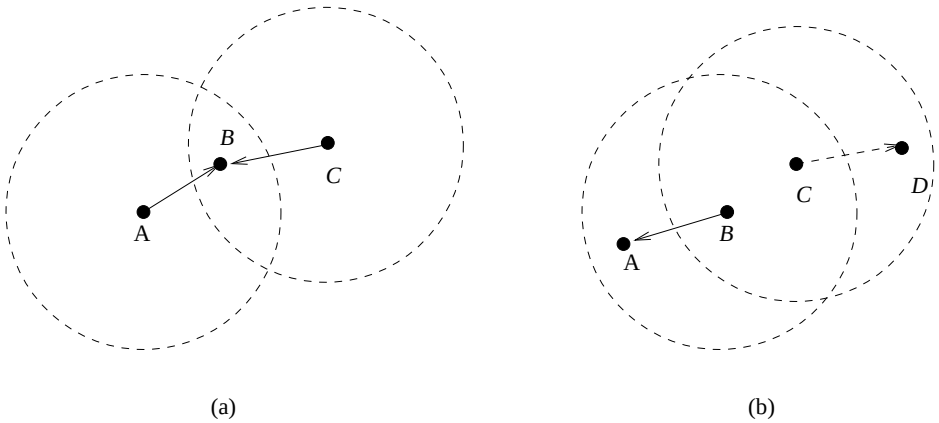
7.3.1 Carrier sensing

The basic approach of the CSMA/CA scheme is as follows. Whenever a node has a packet to be transmitted, it first listens to the channel to ensure that no other node is transmitting. If the channel is clear, it transmits the packet. Otherwise, it uses an exponential back-off scheme until it either finds a time point in which the channel is clear so that it can transmit its packet or aborts the transmission due to too many failed attempts. (The idea of back-off is that whenever a node that has a packet to transmit experiences a busy channel, it retries at a subsequent time with a reduced probability of transmission. In exponential back-off, the retransmission probability is reduced exponentially.)

In wireless devices, there is usually just one antenna for both sending and receiving, and hence the nodes are not able to listen while sending. For this and other reasons there is no collision detection capability like in the Ethernet. Therefore, acknowledgment packets (ACK) have to be sent from the receiver to the sender to confirm that packets have been correctly received.

In wireless ad hoc networks that rely on a carrier-sensing random access protocol, such as IEEE 802.11, the wireless medium characteristics generate complex phenomena such as the well-known *hidden-node problem* and the *exposed-node problem* as shown in Figure 7.4. In order to handle these problems, the MAC layer uses *physical* and *virtual carrier sensing* techniques.

The physical-carrier-sensing part of the CSMA scheme is provided by a Clear Channel Assessment (CCA) circuit. This circuit monitors the environment to determine when it is clear to transmit. It can be programmed to be a function of the Receive Signal Strength Indication (RSSI) and other parameters. The RSSI measurement is derived from the state of the Automatic Gain Control (AGC) circuit. Whenever the RSSI exceeds a certain threshold, a special Energy Detection (ED) bit is switched to 1, and otherwise it is set to 0. By manipulating a certain configuration register, this threshold may be set to an absolute power value of t dB, or it may be set to be t dB above the



(a) (b)

Figure 7.4: Figure in (a) shows the hidden node problem where nodes A and C cannot send to B at the same time and (b) shows the exposed node problem where C cannot sent packets to D while B is sending to A as C senses busy medium though A is out of the transmission range of C.

measured noise floor, where t can be set to any value in the range 0-127. The ability to manipulate the CCA rule allows the MAC layer to optimize the physical carrier sensing to its needs.

Virtual carrier sensing is usually achieved by using two control packets, Request-To-Send (RTS) and Clear-To-Send (CTS), which are exchanged before the data transmission is taking place. Virtual carrier sensing has been added to 802.11 to mitigate the hidden node problem. More precisely, before transmitting a data frame, the source node sends an RTS packet to the receiving node announcing the upcoming frame transmission. Upon receiving the RTS packet, the destination replies by a CTS packet to indicate that it is ready to receive the data frame. Upon receiving the CTS packet, the sender initiates transmitting the actual data. Both the RTS and CTS packets contain the total duration of the transmission, i.e. the overall time needed to transmit the data frame and the related ACK, so that other nodes within the transmission range of either the source or the destination stay silent until the transmission is complete.

7.3.2 Transmission range, interference range, and physical carrier sensing range

Every data transmission mechanism has a minimum signal-to-noise ratio (SNR) at which a data frame can still be transmitted with a reasonably low frame error rate. The minimum SNRs for 802.11b, for example, are 10dB for 11Mbps, 8dB for 5.5Mbps, 6dB for 2Mbps, and 4dB for 1Mbps, and for 802.11a, 23dB is usually the minimum SNR for 54Mbps. In the 802.11a standard [115], the minimum dB values are defined as the received signal strength level at which the frame error rate (FER) of a 1000-octet frame is less than 10%.

The SNRs above specify the *transmission range*, TX_RANGE, of the data transmission mechanism, i.e. the maximum range within which data frames can still be received correctly. The transmission range is highly dependent on the environment. A reasonable model for determining the transmission range is the log-normal shadowing model [89, 122]. In this model, the received power at a distance of d relative to the received power at a reference distance of d_0 is given in dB as

$$-10\ \theta\ \log_{10}\ (d/d_0) + X_\sigma$$

where θ is the path loss coefficient and X_σ is a Gaussian random variable with zero mean and standard deviation σ (in dB) that models the influence of the background noise. θ usually ranges from 2 (free space) to 5 (indoors) [124]. For example, if the received power at a distance of 100 meters from the transmitting node is 40 dB, then at a distance of 200 meters, the received power would be $40 - 10\ \theta + X_\sigma$ dB $= 20 + X_\sigma$ dB when we use $\theta = 2$.

When using forward error correction mechanisms as proposed in the IEEE 802.11e MAC standard currently under development, the transition between being able to correctly receive a data frame with high probability and not being able to correctly receive a data frame with high probability is very sharp. As shown in [25], it can be less than 1 dB. Thus, in an ideal environment the

transmission range is an area with a relatively sharp border that in reality, however, may be blurred due to environmental effects.

A limitation of the shadowing model is that it is only applicable in uniform environments. In non-uniform environments, the signal strength can exhibit a non-monotonic behavior. For example, it can happen that the sender position A has a smaller distance to a position B than to a position C and yet the strength of the signal from A received at B is lower than the signal strength received at C. This can even happen if B and C are close by.

The PCS_RANGE is the range within which a node can detect a busy channel. As explained earlier, this range can be set through the CCA circuit. The IF_RANGE is the range within which a transmitting node can cause interference at other nodes. Thus a transmission from node u can interfere a receiving node v if v is in the IF_RANGE of u. Normally, IF_RANGE is larger than the TX_RANGE and a good approximation is to use the range over which the signal strength is above a certain constant fraction of the white Gaussian noise.

From the above, it can be said that for the interference and physical carrier sensing ranges there does not seem to be a commonly accepted definition in practice. So we will use a conservative model for these ranges to make sure that our results in this model are meaningful in practice.

7.3.3 Formal model

In our model, we assume that we are given a set V of mobile stations, or *nodes*, that are distributed in an arbitrary way in a 2-dimensional Euclidean space. For any two nodes $v, w \in V$, let $d(v, w)$ be the Euclidean distance between v and w. Furthermore, consider any cost function c with the property that there is a fixed constant $\delta \in [0, 1)$ so that for all $v, w \in V$,

- $c(v, w) \in [(1 - \delta) \cdot d(v, w),\ (1 + \delta) \cdot d(v, w)]$ and

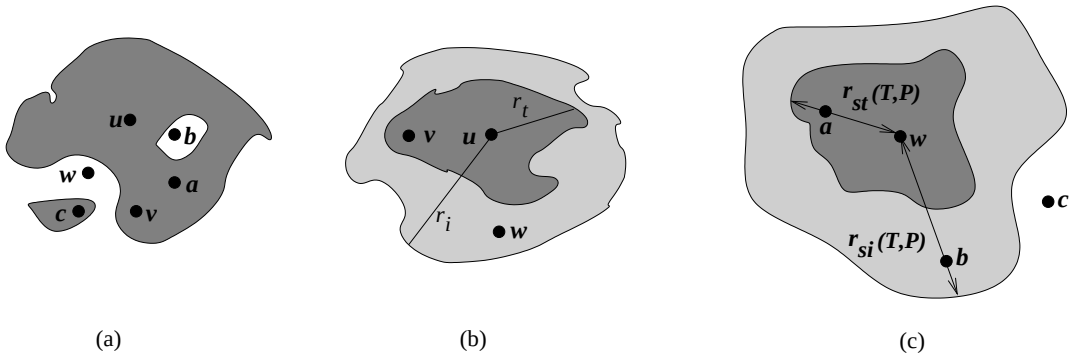


Figure 7.5: Properties of the new model for wireless communication.

- $c(v, w) = c(w, v)$, i.e., c is symmetric.

c determines the transmission and interference behavior of nodes and δ bounds the non-uniformity of the environment. Notice that we do not require c to be monotonic in the distance or to satisfy the triangle inequality. This makes sure that our model even applies to highly irregular environments. In Figure 7.5(a), for example, the distance between u and v is greater than the distance between u and w. Yet, the cost of communicating between u and w, $c(u, w)$, is bigger than $c(u, v)$. In Figure 7.5(a), node u can communicate directly with nodes v, a, and c but not with nodes b and w. Similar cost functions were also used in [87].

We assume that the nodes use some fixed-rate power-controlled communication mechanism over a single frequency band. When using a transmission power of P, there is a transmission range $r_t(P)$ and an interference range $r_i(P) > r_t(P)$ that grow monotonically with P. The interference range has the property that every node $v \in V$ can only cause interference at nodes w with $c(v, w) \leq r_i(P)$, and the transmission range has the property that for every two nodes $v, w \in V$ with $c(v, w) \leq r_t(P)$, v is guaranteed to receive a message from w sent out with a power of P (with high probability) as long as there is no other node $u \in V$ with $c(v, u) \leq r_i(P')$ that transmits a message at the same time with a power of P'. Figure 7.5(b) shows the above ranges for a node v.

For simplicity, we assume that the ratio $\rho = r_i(P)/r_t(P)$ is a fixed constant greater than 1 for all relevant values of P. This is not a restriction because we do not assume anything about what happens if a message is sent from a node v to a node w within v's transmission range but another node u is transmitting a message at the same time with w in its interference range. In this case, w may or may not be able to receive the message from v, so any worst case must be assumed in the analysis. The only restriction we need, which is important for any overlay network algorithm to eventually stabilize, is that the transmission range is a sharp threshold. That is, beyond the transmission range a message cannot be received any more (with high probability). This is justified by the fact that when using modern forward error correction techniques, the difference between the signal strength that allows to receive the message (with high probability) and the signal strength that does not allow any more to receive the message (with high probability) can be very small (less than 1 dB).

Nodes can not only send and receive messages but also perform physical carrier sensing, which has not been considered before in models proposed in the algorithms community. Given some sensing threshold T (that can be flexibly set by a node) and a transmission power P, there is a *carrier sense transmission (CST) range*, denoted $r_{st}(T, P)$, and a *carrier sense interference (CSI) range*, denoted $r_{si}(T, P)$, that grow monotonically with T and P. The range $r_{st}(T, P)$ has the property that if a node v transmits a message with power P and a node w with $c(v, w) \leq r_{st}(T, P)$ is currently sensing the carrier with threshold T, then w senses a message transmission (with high probability). The range $r_{si}(T, P)$ has the property that if a node v senses a message transmission with threshold T, then there was at least one node w with $c(v, w) \leq r_{si}(T, P)$ that transmitted a message with power P (with high probability). More precisely, we assume that the monotonicity property holds. That is, if transmissions from a set U of nodes within the $r_{si}(T, P)$ range cause v

to sense a transmission, then any superset of U will also do so. The two sensing ranges are shown in Figure 7.5(c) shows the sensing ranges. When all nodes use a transmission power P and node w uses a threshold of T, in this example, node w can *always* sense transmissions of node a while it *may* sense transmissions of node b and *can never* sense transmissions of node c.

For simplicity, we will assume in the following that for the carrier sense ranges, for all relevant values of T and P, $r_{si}(T,P)/r_{st}(T,P) = r_i(P)/r_t(P)$.

7.4 Our contributions

Our contributions are two-fold: apart from the new model for wireless networks, we demonstrate how to develop and analyze algorithms on top of this model by presenting self-stabilizing local-control algorithms for building constant density dominating sets and spanners.

In our algorithms, the nodes do not require *any* a-priori knowledge about the other nodes, not even an estimate on their total number. Also, fixed identification numbers of any form are not required so that our protocols may even be applicable to the important field of sensor networks. It is sufficient for us if the nodes choose identification numbers so that there are no local conflicts (which can be easily achieved with random, local-control coloring strategies). In this case, we also say that the labels are *locally distinct* meaning that the label of a node u is different from the label of any node v that is within the transmission range of u. We only require that the mobile hosts can synchronize in rounds of constant length. This can be done, for example, with the help of GPS signals or any form of beacons (that are sufficiently far apart in time for a round of our protocols to complete).

In order to obtain a constant density spanner under an arbitrary distribution of nodes, we proceed in two stages. First, we show that there is a simple, distributed protocol to obtain a constant

density dominating set, and then we show how to extend this protocol in order to also obtain a constant density spanner.

It is worth noting that our protocols only need a constant amount of storage at each node, irrespective of the distribution of the nodes. The constant only depends on the δ in our model. Moreover, our protocols can self-stabilize even if some of the nodes show *arbitrary* adversarial behavior. We only require the honest nodes that are outside a certain range of the adversarial nodes to be placed so that they can in principle form a single connected component. In this case, the protocol would then arrive at a constant density spanner in a finite amount of time. So our protocols would even work for very primitive devices in hostile environments.

7.4.1 Constant density dominating set

We start with the following definitions for dominating set and maximal independent set.

Definition 7.4.1 (Dominating set) *Given an undirected graph $G = (V, E)$, a subset $U \subseteq V$ is called a* dominating set *if all nodes $v \in V$ are either in U or have an edge to a node in U. A dominating set U is called* connected *if U forms a connected component in G. The* density *of a dominating set U is the maximum over all nodes $v \in U$ of the number of neighbors that v has in U.*

Definition 7.4.2 (Maximal independent set (MIS)) *Given an undirected graph $G = (V, E)$, a subset $U \subseteq V$ is called an* independent set *if for any pair of nodes $v, w \in U$, there is no edge between v and w in G. If an independent set U has the property that every node $v \in V \setminus U$ is a neighbor of at least one node in U, then U is called a* maximal independent set.

Given an arbitrarily distributed set V of nodes in a 2-dimensional Euclidean space, let the graph $Q_r = (V, E_r)$ contain all edges $\{v, w\}$ with $d(v, w) \leq r$. Suppose that we select a maximal

independent set U in Q_r. Then this is also a dominating set of constant density because in the 2-dimensional Euclidean space a node can have at most five neighbors within a distance of r that are part of an independent set in Q_r [144]. Note that a constant density dominating set is also a constant factor approximation of a minimum dominating set, a well-studied problem in the algorithms and wireless networking community.

Now, let us consider the graph $G_r = (V, E'_r)$ that contains all edges $\{v, w\}$ such that $c(v, w) \leq r$. Since $c(v, w) \leq (1 + \delta)\ d(v, w)$, it follows from [144]:

Fact 7.4.3 *Every node v can have at most five neighbors within a Euclidean distance of $r/(1 + \delta)$ that are part of an independent set in G_r.*

Otherwise, there must be a pair $v, w \in V$ with $c(v, w) \leq (1+\delta)d(v, w) \leq (1+\delta)\cdot r/(1+\delta) = r$ that are in an independent set in G_r, a contradiction. Furthermore, because $c(v, w) \geq (1-\delta)d(v, w)$, a node can only be connected in G_r to nodes up to a Euclidean distance of $r/(1 - \delta)$. Hence, it is easy to see that for every node v there is a set C_v of neighbors of v in G_r of constant size so that for every neighbor w of v in G_r there is a neighbor $w' \in C_v$ with $d(w, w') \leq r/(1 + \delta)$. Combining this with Fact 7.4.3, we get:

Fact 7.4.4 *For any independent set in G_r it holds that every node v in G_r can have at most a constant number of neighbors in this set, where the constant depends on δ.*

Now, recall that any maximal independent set in a graph G_r is also a dominating set in G_r, and according to the fact above, any maximal independent set in G_r has a constant density (i.e., every node only has a constant number of neighbors in that set). Hence, in order to obtain a dominating set of constant density, it suffices to design an algorithm that constructs a maximal independent set in G_r. It turns out that constructing such a set is quite tricky, given the uncertainties in our model, but we can construct something close to that so that the following result holds.

Theorem 7.4.5 *For any desired transmission range r and any initial situation, the dominating set protocol generates a constant density dominating set in G_r in $O(\log^4 n)$ communication rounds, with high probability.*

Hence, our protocol self-stabilizes within $O(\log^4 n)$ rounds. Interestingly, this result is only possible because our protocol uses physical carrier sensing. It is known that if physical carrier sensing is not available and the nodes have no estimate of the size of the network, then it takes $\Omega(n)$ steps on expectation for a single message transmission to be successful [66] in any protocol.

7.4.2 Constant density spanner

We start with the definition of a spanner.

Definition 7.4.6 (Spanner) *A subgraph H of a graph G is called a* (topological) t-spanner *of G if for every pair of nodes v, w in G there is a path in H from v to w whose length is at most t times the minimum length of a path from v to w in G. In this case, t is also called the* stretch factor *or the* spanning ratio *of H.*

Notice that a connected dominating set forms a topological spanner. If the connected dominating set has constant density, then we say that the resulting topological spanner also has constant density. We thus extend the dominating set protocol by additional protocols that connect the nodes in the dominating set via so-called gateway nodes so that the following result holds.

Theorem 7.4.7 *For any desired transmission range r and any initial situation, the spanner protocol generates a constant density spanner in G_r in $O(D \log^2 n + \log^4 n)$ communication rounds, with high probability, where D is the maximum number of nodes that are within the transmission range of a node.*

All of our protocols can self-stabilize even under adversarial behavior as long as the nodes outside a range of $r' = \Theta(r)$ of adversarial nodes form a connected component in G_r.

7.5 Related work

The problem of finding a minimum dominating set is an important restriction of the more general set cover problem. The minimum dominating set problem has been proven to be NP-complete in [48, 71]. If we take all nodes as dominating set then this will include less than $(\Delta + 1)$ times the size of optimal minimum dominating set, where Δ is the maximum degree in the graph, so an $O(\Delta)$ approximation is trivial. The greedy algorithm first takes a node with maximum degree and continues taking the node which covers maximum number of uncovered nodes until every node is covered. This algorithm achieves a $\log \Delta$ approximation [65, 100, 137] and Feige [40] proved that the approximation ratio achieved by the greedy algorithm is best possible, unless NP has $n^{O(\log \log n)}$ time deterministic algorithms.

The problem of finding a minimum dominating set has been shown to be NP-complete even when restricted to unit disk graphs [27] and, hence, approximation algorithms are of interest. Recent research focused on developing distributed (rather than centralized) algorithms for finding good approximations of minimum dominating sets in arbitrary graphs. A simple and elegant distributed approximation algorithm was proposed by Luby [102].

Distributed algorithms for small size of dominating sets is extensively studied. Liang and Haas [97] presented distributed implementation of the greedy algorithm. However, the runtime of the algorithm in [97] can be polynomial in the number of nodes in the network. Jia et al. [63] extended this algorithm to obtain a local randomized greedy algorithm which works in $O(\log n \log \Delta)$ time. The algorithm of [63] is also a $\log \Delta$ approximation algorithm but the resulting dominating set is

not guaranteed to be a connected dominating set.

Kuhn and Wattenhofer [84] proposed an algorithm based on LP relaxation techniques which achieves $O(k\Delta^{2/k}\log\Delta)$ approximation in $O(k^2)$ time for an arbitrary number k. Recently, Dubhashi et al. [36] presented a $\log\Delta$ approximation algorithm for minimum connected dominating set problem which works in polylogarithmic time. The key observation in [36] is to sparsify a given graph as the resulting graph has only a linear number of edges but still stays conected. It was also shown in [36] that no such distributed algorithm that runs in $o(n)$ rounds exists where as their randomized algorithm runs in $O(\log n)$ time.

Wu and Li [150] presented an algorithm finds an initial connected dominating set and removes redundant nodes from this set. This algorithm requires two hop information. This algorithm has time complexity $O(\Delta^2)$ and message complexity $O(n\Delta)$.

Alzoubi et al. [5] presented the first constant approximation algorithm for the minimum connected dominating set problem in unit-disk graphs with $O(n)$ time and $O(n\log n)$ message complexity, respectively. Cheng et al. [23] proposed a polynomial time approximation scheme for the connected dominating set problem in unit-disk graphs.

Huang et al. [60] formally analyze a popular algorithm used for clustering in ad-hoc mobile network scenarios. They show that this algorithm actually gives a 7-approximation for the minimum dominating set problem in unit-disk graphs, while adapting optimally to the mobility of the nodes in the network.

Recently, Kuhn et. al. [83] presented a distributed algorithm that computes a constant factor approximation of a minimum dominating set in $O(\log^2 n)$ time without needing any synchronization but it requires that nodes know an estimate of the total number of nodes in the network. In [119], Parthasarathy and Gandhi also present distributed algorithms to compute a constant factor

approximation to the minimum dominating set. The running time of their algorithm depends on the amount of information available to the nodes, and nodes have to know an estimate of the size of the network. Both papers extend the unit disk model taking into account signal interference.

Spanners

Suppose that we have a set of nodes V that are distributed in an arbitrary way in a Euclidean space. For $v, w \in V$, let $d(v, w)$ denote the Euclidean distance between v and w. The goal of the geometric spanner problem is to find a graph $G = (V, E)$ so that for each pair of nodes $v, w \in V$ there is a path in G from v to w whose length is at most $t \cdot d(v, w)$ for some fixed constant t. In this case, G is called a *geometric t-spanner* of G where t is the stretch factor.

Bose et. al. [16] proposed a $O(n \log n)$ time centralized algorithm that constructs a planar t-spanner, for $t \leq 10.02$, such that the degree of each node is at most 27. This is the first algorithm that constructs a planar spanner of bounded degree.

For constructing geometric spanners, several structures have been proposed. It is known that Delaunay triangulation is a planar t-spanner for $t \leq 4\sqrt{3}\pi/9$ [73]. Hence constructions based on the Delaunay triangulation are studied e.g., [47, 93, 147]. While the spanner constructed in [47, 93] is planar, the node degree is not bounded. Wang and Li [147] proposed an efficient localized algorithm that constructs a bounded degree planar t-spanner. This spanner has spanning ratio $t = \max\{\frac{\pi}{2}, \pi \sin\frac{\alpha}{2} + 1\} \cdot C_{\text{del}}$ and each node has degree at most 25 where $0 < \alpha \leq \pi/3$ where $C_{\text{del}} \leq 4\sqrt{3}\pi/9$ is the spanning ratio of the Delaunay triangulation.

Spanners based on the Yao graph [152] and the Gabriel graph [46] are presented in [94, 148, 138]. Of these, the results of [94, 95] guarantee constant degree and constant spanning ratio but are not guaranteed to give a planar spanner. Song et. al. [138] construct a planar low degree spanner combining the constructions of both the Gabriel graph and the Yao graph.

For topological spanners, Dubhashi et. al. [36] presented a spanner with logarithmic stretch factor. Alzoubi et. al. [5] presented a spanner with constant stretch factor of 5 where the protocol is very similar to ours but uses a high-level model for wireless networks. Our protocol for selecting gateway nodes also has similarities to the protocols presented in [146, 47]. However both these papers are based on high-level wireless models.

7.6 Overview of spanner protocol

In the following, r_t denotes the desired transmission range and G_{r_t} represents the graph with node set V and edge set E_{r_t} containing all edges $\{v, w\}$ with $c(v, w) \leq r_t$.

Our spanner protocol for G_{r_t} consists of 3 phases:

- Phase I: The goal of this phase is to construct a constant density dominating set in G_{r_t}. This is achieved by extending Luby's algorithm [102] to our more complex model. Since the dominating set resulting from Phase I may not be connected, we need further phases to obtain a constant density spanner.

- Phase II: The goal of this phase is to organize the nodes of the dominating set of Phase I into color classes that keep nodes with the same color sufficiently far apart from each other. Only a constant number of different colors is needed for this, where the constant depends on δ. Every node organizes its rounds into time frames consisting of as many rounds as there are colors, and a node in the dominating set only becomes active in Phase III in the round corresponding to its color.

- Phase III: The goal of this phase is to interconnect every pair of nodes in the dominating set that is within a hop distance of at most 3 in G_{r_t} with the help of at most 2 gateway nodes,

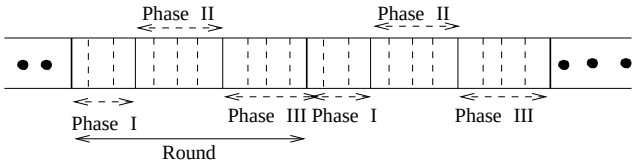


Figure 7.6: Two consecutive rounds of the spanner protocol.

using the coloring determined in Phase II to minimize interference problems. Constructions using gateway nodes were also presented in [47, 146] but assuming a higher level model of wireless networks.

Each phase has a constant number of time slots associated with it, where each time slot represents a communication step. Phase I consists of 3 time slots, Phase II consists of 4 time slots, and Phase III consists of 4 time slots. These 11 time slots together form a *round* of the spanner protocol (see also Figure 7.6). We assume that all the nodes are synchronized in rounds, that is, every node starts a new round at the same time step. As mentioned earlier, this may be achieved via GPS or beacons.

The spanner protocol establishes a constant density spanner by running sufficiently many rounds of the three phases. All of the phases are self-stabilizing. More precisely, once Phase I has self-stabilized, Phase II will self-stabilize, and once Phase II has self-stabilized, Phase III will self-stabilize. In this way, the entire algorithm can self-stabilize from an arbitrary initial configuration.

It is not difficult to see that our spanner protocol results in a 5-spanner of constant density: Consider any pair of nodes s and t in G_{r_t} and let $p = (s = v_0, v_1, \ldots, v_k = t)$ be the shortest path from s to t in G_{r_t}. Then we can emulate p via the connected dominating set by first going to a leader ℓ_0 of s, then (possibly via gateway nodes) to a leader ℓ_1 of v_1, then to a leader ℓ_2 of v_2, and so on, until we reach a leader ℓ_k of t, and finally to t. The length of this path is at most $3k + 2 \leq 5k$ for every $k \geq 1$. Combining this with the time bounds shown for the various phases in the sections

below results in Theorem 7.4.7.

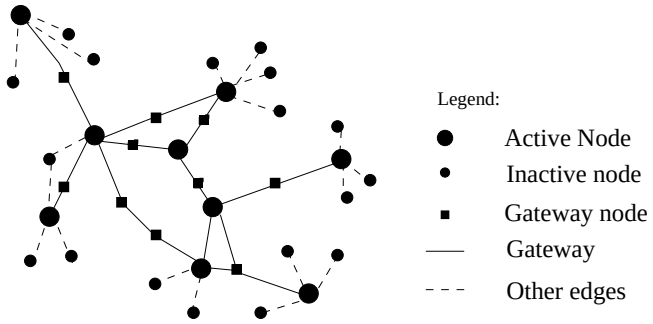


Figure 7.7: The spanner of the original network.

An important feature of our protocol is that all messages sent are of constant length and the nodes only have to have a constant amount of storage, irrespective of the density of the network. We just need the assumption that a storage unit is large enough to store the ID of any node. Hence, our protocol can be used with very simple devices such as sensors.

7.7 Phase I: dominating set

Let P be some fixed transmission power with transmission range r_t and interference range r_i for which we want to construct a dominating set of constant density. That is, given any set of nodes V, we want to find a subset $U \subset V$ of nodes so that every node $v \in V$ has at least one node $w \in U$ with $c(v, w) \leq r_t$ and at most some constant number of nodes $w \in U$ with $c(v, w) \leq r_t$.

As mentioned earlier, if we want to reach the goal above in a sub-linear number of steps without physical carrier sensing, then a good approximation of $\log n$ is needed, where $n = |V|$. Since our goal is to arrive at a dominating set without using any prior knowledge of the network topology, physical carrier sensing has to be used, which complicates the design as it has uncertainties (see our model). To handle these uncertainties, we use a distributed coloring strategy together with two different sensing ranges.

In our protocol, nodes can either be *active* or *inactive*. The active nodes are the candidates for the dominating set. The nodes use two different sensing thresholds, depending on their state. The sensing threshold T_a has a CSI range of r_t and the sensing threshold T_i has a CST range of r_i. To distinguish between these ranges, we speak about an aCST/aCSI-range whenever we mean T_a and iCST/iCSI-range whenever we mean T_i.

Each node cuts the time into *time frames* of k *rounds* each for some constant number k that is the same for every node. The rounds are synchronized among the nodes but we do not require the frames to be synchronized.

Initially, all nodes are inactive. Afterwards, each node executes the following protocol in each round. In this protocol, each active node has exactly one, fixed active round in a frame and a signal is just a very simple message. Each item represents a communication step.

1. If v is active and in its active round, then v sends out an ACTIVE signal.

   If v is inactive and v did not sense any ACTIVE signal for the last k rounds using a sensing threshold of T_a, v senses with threshold T_i, and if it does not sense anything, it becomes active and declares the current round number as its active round. If v did sense some ACTIVE signal in one of the last k rounds, it just performs sensing with threshold T_a and records the outcome.

2. If v is active and is in its active round, then with some fixed probability p, to be determined later, v sends out a LEADER message containing its ID. If v decides not to send out a LEADER message but it either senses a LEADER message with threshold T_a or receives a LEADER message, v becomes inactive.

In the following, let $H_{r,k} = (V, E)$ be an undirected graph that contains an edge between two nodes v and w if and only if v and w are active and use the same active round (or color) k and $c(v, w) \leq r$. A node v is called a *leader* if it is active and there is no other active node w of the

same color with $c(v, w) \leq r_t$. Since inactive nodes sense with an iCST range of r_i before they become active, none of the inactive nodes w with $c(v, w) \leq r_i$ will become active in the active round of v. Hence, we get:

Fact 7.7.1 *At any time, the set of leader nodes forms an independent set in $H_{r_t,k}$ that is disconnected from all other active nodes in $H_{r_t,k}$.*

In addition, a leader node uses an aCSI range of r_t and will therefore not be affected by nodes outside of a range of r_t. Hence, we arrive at the following fact.

Fact 7.7.2 *Once a node becomes a leader, it will stay a leader as long as the cost function c does not change.*

Furthermore, an inactive node v can only become active if in the previous k rounds there was no active node w with $c(v, w) \leq r_s$, where r_s is the CST range for threshold T_a, because otherwise v would have sensed the ACTIVE signal of w in one of these rounds. Hence, we also get:

Fact 7.7.3 *There cannot be two leaders v and w with $c(v, w) \leq r_s$.*

Since r_t/r_s is a constant, the facts above and Fact 7.4.4 imply that the leaders must form a set of constant density in G_{r_t}. On the other hand, the following lemma is true.

Lemma 7.7.4 *In any situation in which all active nodes are leaders but the leaders do not form a dominating set with respect to G_{r_t}, at least one inactive node will eventually become active.*

Proof. From Facts 7.4.4 and 7.7.3 it follows that there can be at most some constant number k' of leaders within the iCSI range of any node. Hence, if $k > k'$ then for every inactive node that does not yet have a leader within its transmission range there must be at least one round s in which there is no leader within its iCSI range. Because the inactive node will continue to explore potential

active rounds in a round-robin fashion as long as it senses a transmission with threshold T_i, it will eventually arrive at round s and become active (unless some other inactive node close to it becomes active before that). □

On the other hand, the following result is easy to check.

Lemma 7.7.5 *Every connected component of active nodes in $H_{r_t,k}$ results in at least one leader.*

Thus, the algorithm eventually arrives at a situation where there is no inactive node that does not have a leader within its transmission range. At that point, the leaders must form a superset of a maximal independent set in G_{r_t}. Thus, according to Facts 7.4.4, 7.7.3, and 7.7.2 the leaders eventually form a static dominating set of constant density. It remains to prove how much time is needed to reach such a state.

Theorem 7.7.6 *If all nodes are initially inactive, after $O(\log^4 n)$ rounds of the algorithm, with high probability, the leaders form a static dominating set of constant density with respect to G_{r_t}.*

Proof. The next two lemmata state important properties of connected components of active nodes in $H_{r_t,k}$. Notice that a leader always represents a connected component by itself.

Lemma 7.7.7 *At any time step t, $H_{r_t,k}$ consists of connected components of active nodes where all nodes in a connected component were reactivated at the same round.*

Proof. Suppose that there are two adjacent nodes, v and w, in some active, connected component in $H_{r_t,k}$ that were not reactivated at the same round. W.l.o.g. let v be the first node that became active. Then w could not have become active because v is in its iCST range, leading to a contradiction. □

For the next lemma, given an active node v, we define $ls(v)$ as the bit sequence in which the ith bit is 1 if and only if v sent out a LEADER message in round i since it joined its current component. $ls(v)_i$ denotes the first i bits of $ls(v)$.

Lemma 7.7.8 *Every connected component of active nodes in $H_{r_t,k}$ needs at most $O(\log n)$ rounds, w.h.p., until every node in it either becomes inactive or becomes a leader.*

Proof. Consider any connected component C of active nodes in $H_{r_t,k}$ at some time point t_0, and let C' be the union of the connected components of active nodes in $H_{r_t,k}$ that have at least one node within the interference range of a node in C.

Whenever a node becomes active after t_0, it cannot interfere with the remaining nodes in C because it will be guaranteed to be outside of their interference range (and therefore also of their aCSI range). Hence, we only need to focus on the remaining active nodes in $C \cup C'$.

We prove the lemma in two steps. First, we show that it only takes $O(\log n)$ rounds, w.h.p., until there are no two active nodes v and w in $C \cup C'$ where w is within the aCST range of v or vice versa. Then we show that it only takes $O(\log n)$ further rounds, w.h.p., until there are no two active nodes v and w in C that are within the transmission range of each other.

The probability that for any two fixed, active nodes v and w it holds that $ls(v)_i = ls(w)_i$ is equal to p^i. Hence, if $i = c \log_{1/p} n$, then the probability that there are two nodes v and w in $C \cup C'$ with $ls(v)_i = ls(w)_i$ that are within their aCST range is at most $n^2/p^{c \log_{1/p} n} = n^{2-c}$. Thus, the probability that after $c \log_{1/p} n$ rounds there are still two nodes within the aCST range in $C \cup C'$ that are both active is polynomially small in n for $c > 2$.

Hence, after $O(\log n)$ rounds, there can only be at most some constant number d of active nodes within the interference range of any active node in C, where d depends on the ratio between the interference range and the aCST range. Thus, when choosing $p = 1/d$, then the probability that exactly one of the active nodes within the interference range of an active node v in C is transmitting a LEADER message in a round is $\Theta(p)$. Therefore, it takes at most $O((1/p) \log_{1/p} n) = O(d \log_d n)$ rounds until for every node v in C that is still active there is no other active node in the transmission

range of v, with high probability. □

Next we give a lower bound on the number of leaders that emerge from a connected component of active nodes in $H_{r_t,k}$. For the rest of the proof, we assume w.l.o.g. that $r_t = 1$ and $r_i = 1 + \alpha$ for some constant $\alpha > 0$. We define the area covered by an active node v as the area that is within the transmission range of v.

Lemma 7.7.9 *For any time step in which the currently existing connected components of active, non-leading nodes cover an area of $A = \Omega(\log^3 n)$, the number of leaders emerging from these components is $\Omega(A/\log^2 n)$, w.h.p.*

Proof. Consider any set C of connected components of active, non-leading nodes that cover an area of A. Given any node v, let $\Gamma(v)$ denote the set of nodes $w \in C$ with $c(v, w) \leq 1$ and let $\gamma(v) = |\Gamma(v)|$. Let H be the directed graph resulting from C by connecting two active nodes v and w by an edge (v, w) if and only if $c(v, w) \leq 1$ and $\gamma(w) \geq 2\gamma(v)$. A node is called a *sink* if it does not have any outgoing edges. H has the following important property:

Claim 7.7.10 *Every node v in H has a directed path to a sink s of length at most $\log n$.*

Proof. First of all, H cannot contain a directed cycle. Thus, every directed path must eventually end in a sink. Suppose now that some node v has a directed path p to a sink s of length more than $\log n$. Because of the definition of the edges, it follows that $\gamma(s) \geq 2^k \cdot \gamma(v) > n \cdot \gamma(v)$, which cannot happen because there are only n nodes in the system. □

Recall that our cost function must satisfy $c(v, w) \in [(1 - \delta)\ d(v, w), (1 + \delta)d(v, w)]$. Thus, if we consider disks of radius $(1 + \log n)/(1 - \delta)$, around the sinks of H, then the complete area A of active, non-leading nodes is covered. To extract out of all sinks a set of sinks useful for our analysis

below, we consider these sinks one by one. For each sink s that has not already been eliminated, eliminate all sinks s' that are of distance at most 4 from s and add s to a set S. At the end, we arrive at a set S of sinks of pairwise distance at least 4 such that disks of radius $r = (5 + \log n)/(1 - \delta)$ around these sinks cover the entire area A. Thus, the area A can be decomposed into areas of size at most $a = \pi r^2$ each containing a sink in S, and therefore $|S| \geq |A|/a$. It is not difficult to show that these sinks have the following property:

Claim 7.7.11 *For any sink $s \in S$, the expected number of active nodes in $\Gamma(s)$ that become a leader is $\Theta(1)$.*

For any sink s, let the random variable X_s denote the number of active nodes in $\Gamma(s)$ that become leaders and let $X = \sum_s X_s$. From Claim 7.7.11 it follows that $\mathrm{E}[X] \geq \alpha|S|$ for some constant $\alpha > 0$, and because the distance between any two sinks in S is at least 4, the X_s variables are independent. Thus, we can use Chernoff bounds to obtain

$$\Pr[X \leq (1 - \epsilon)\alpha|S|] \leq e^{-\epsilon^2 \alpha |S|/2}$$

for any $\epsilon > 0$. This is polynomially small if $\epsilon = 1/2$ and $|S| = \Omega(\log n)$ is sufficiently large. Hence, in this case,

$$\Pr\left[X \leq \alpha|S|/2\right] = \Pr\left[X \leq \frac{\alpha|A|}{2\pi r^2}\right]$$

is polynomially small, which completes the proof of the lemma. □

Now, let us call a node *unfinished* if it is active but not a leader or it is inactive and it does not have a leader within its transmission range. We know that an unfinished node is either active or must have at least one node within its iCSI range, r_{ii}, that was active within the previous k rounds (because otherwise it would become active). Hence, when drawing disks of radius $r_{ii}/(1 - \delta)$

around all nodes that were active in at least one of the k previous rounds, the entire area that the nodes can transmit messages to is covered.

Let A_0 be the area covered by the transmission ranges of all the nodes in the system. If $A_0 = \Omega(\log^3 n)$, then Lemma 7.7.8 and Lemma 7.7.9 imply that after $O(\log n)$ rounds, the area covered by the unfinished nodes is at most

$$A_0 - c \cdot \frac{A_0}{\log^2 n} = \left(1 - \frac{c}{\log^2 n}\right) A_0$$

for some constant c, with high probability. Thus, after k stages of $O(\log n)$ rounds each, the area covered by the unfinished nodes is at most

$$\left(1 - \frac{c}{\log^2 n}\right)^k A_0 \leq e^{(c \cdot k)/\log^2 n} A_0 \ ,$$

with high probability. The right hand side is less than $\log^3 n$ if $k \geq (\log A_0)(\log^2 n)/c$. Once an area of size $O(\log^3 n)$ is reached, it follows from Lemma 7.7.5 that it takes only $O(\log^3 n)$ more stages of $O(\log n)$ rounds each until there are no unfinished nodes any more. Since $A_0 = O(n)$, it follows that the total runtime needed for the set of active nodes to stabilize is $O(\log^4 n)$. □

The dominating set algorithm can be easily extended so that it self-stabilizes [32] and it is robust against malicious behavior. Self-stabilization means that it can recover from *any* initial configuration.

7.7.1 Self-stabilization

An extra rule is necessary to provide self-stabilization because if the protocol above starts in a configuration violating Fact 7.7.3, it may not succeed in establishing a dominating set.

Consider adding a third step to each round of the protocol above. In this step, every active node sends a leader message with probability p and a transmission power so that its transmission range is only equal to the aCST range. Adding now the rule that whenever an active node receives a leader message in that step for a round different from its active round, then it becomes inactive, we do not have to assume anything about how the nodes are initially activated in order to satisfy Fact 7.7.3. So we get:

Corollary 7.7.12 *for any initial situation, the extended protocol needs at most $O(\log^4 n)$ rounds to arrive at a static dominating set of constant density with respect to G_{r_t}, w.h.p.*

7.7.2 Robustness

Our dominating set algorithm is also highly robust against adversarial nodes. For any node v, let the $r_1 \oplus r_2$-range of v be defined as the union of the r_2-ranges of all the nodes within the r_1-range of v. Given any distribution of nodes, let A be the area covered by the $r_{ii} \oplus r_t$-ranges of adversarial nodes, where r_{ii} is the iCSI range of a node. Because in our protocol adversarial nodes can directly influence only nodes within their iCSI range, nodes beyond the r_t range of these nodes can only have leaders outside of A, and leaders outside of A will stay leaders forever, one can show:

Corollary 7.7.13 *If the honest nodes outside A are connected in G_{r_t}, then after $O(\log^4 n)$ rounds, the active honest nodes outside A form a dominating set of constant density with respect to G_{r_t}, w.h.p.*

7.8 Constant density spanner

In the next two subsections, we describe Phases II and III in detail. We use the following notation. The constant d_1 refers to the number of active nodes that are within the interference range

r_i of any node. The constant d_2 refers to the number of active nodes that are within the $r_i \oplus r_i$-range of any node, and the constant g refers to the maximum number of required gateway connections for any active node. Finally, D refers to the density of the network, i.e. the maximum number of nodes within the transmission range of a node.

7.8.1 Phase II - Distributed Leader Coloring

Similar to Phase I, each node organizes the time into time frames consisting of cd_1 rounds for some constant c that is the same for every node. Also here, the rounds are synchronized but frames do not have to be synchronized among the nodes. We again assign active nodes to distinct rounds using a coloring mechanism. While the coloring in Phase I was done with respect to G_{r_t}, we now need a coloring of the active nodes with respect to $G_{r_i \oplus r_i}$, that is, we need the active nodes to be at least $r_i \oplus r_i$ apart in order to receive the same color.

Every active node from Phase I tries to own one of the rounds. An active node u is said to *own* a round if no other active node within its $r_i \oplus r_i$ range is using that round. Active nodes are in one of the states $\{\text{owner}, \text{volatile}\}$. An active node is in owner state if it already owns a round and is in volatile state if it is still trying to own a round. Active nodes in owner state always send their ID in the first time slot of their round. Initially, every active node is volatile. Active nodes in volatile state choose an active round from the cd_1 possible rounds uniformly at random. Active nodes in owner state use a sensing threshold T_o with CST range r_i and active nodes in volatile state use a sensing threshold T_v with a CST range being equal to the CSI range of T_o, r_{ii}.

Active nodes do the following repeatedly. Every time a node re-activates, it sets its time stamp to 0. This time stamp is used by active nodes in Phase III to compare entries. Each item below represents a communication step.

1. Every active node in owner state that is in its active round sends out a LEADER message containing its ID and its current time stamp and increases its time stamp by one afterwards.

2. Every active node in owner state that is in its active round decides with probability $1/2$ to send out an OWNER message either in the first or second substep of step 2.

3. Every inactive node that sensed a LEADER message with threshold T_v sends out a BUSY signal. Every active node in volatile state that senses a BUSY signal in its active round chooses a new active round uniformly at random.

4. Every inactive node that sensed OWNER messages in both substeps of step 2 with threshold T_o sends out a COLLISION signal.

   If an active node in owner state senses a COLLISION signal and sent an OWNER message in the second substep, it changes into volatile state and chooses a new active round uniformly at random.

   If an active node in volatile state did not sense a BUSY or COLLISION signal in its active round, it becomes an owner.

With the above protocol we can arrive at the following result.

Theorem 7.8.1 *Once a stable set of active nodes is available, it holds: If $c \geq 4$, then all active nodes will be in owner state after $O(\log n)$ rounds of the protocol, w.h.p.*

Proof. We proceed as follows. There are two reasons due to which active nodes have to look for a new round. First, active nodes in volatile state may encounter collisions because of other volatile active nodes choosing the same round. This causes them to keep trying for another round. Secondly, active nodes in owner state may receive or sense a collision due to some volatile active

node choosing the same round. Here node u that owns a round may get a COLLISION signal if v chooses the same round and u and v are not within the interference range of each other and chose to send the OWNER message in the second sub-step. So node v during its listening period does not know about node u.

We now try to bound the probability that some volatile active node continues to be in volatile state after $O(\log n)$ attempts. The probability that two volatile nodes choose the same round is at most $\frac{1}{cd_1}d_1 \leq \frac{1}{c} \leq 1/4$ since $c \geq 4$. Thus after $2\log n$ attempts, the probability that there is still some active node in volatile state is at most $n\frac{1}{4^{2\log n}} = 1/n^3$.

Similarly, we can compute the probability that an owner active node has to change its round due to some volatile or owner node. The probability that some owner node has to change its round due to a volatile node is at most $\frac{1}{cd_1}d_1(1/2) \leq 1/c \leq 1/8$ if $c \geq 4$. In the above calculation, the owner node will have to change its rounds only if it chose to transmit the OWNER signal in the second sub-step of the second step. Thus after $2\log n$ attempts, the probability that there is still some active node in volatile state, that has to still look for a round, is at most $1/n^4$.

We can treat these two types of collisions as failure events and bound the probability of failure due to any event to be less than $O(1/n^3)$ and hence w.h.p. after $O(\log n)$ attempts, all active nodes become owners when $c \geq 4$. □

Without the two types of signals BUSY and COLLISION and the two different sensing thresholds the coloring achieved may fail to be $r_i \oplus r_i$ distinct. For any active node ℓ in volatile state, the threshold T_v and the BUSY signal helps to identify the presence of active nodes in owner state with the same active round so that active nodes in owner state without another active node in owner state within the $r_{ii} \oplus r_{ii}$-range will also keep this property in the future and are therefore safe from becoming volatile again. The COLLISION signal is necessary to resolve conflicts among close by

active nodes in owner state with the same active round, which can happen if volatile nodes become an owner in the same round, or this may be part of the initial state when looking at self-stabilization. In any case, the monotonicity assumption on the sensing in our model is important to make sure that there will either never be a conflict among owner nodes or immediately a conflict when a collision is detected.

The theorem implies that after $O(\log n)$ rounds, all active nodes have chosen rounds so that for any two active nodes ℓ and ℓ' with the same round and any inactive node v within the interference range of ℓ, ℓ' is outside of the interference range of v. Hence, ℓ can transmit messages to nodes within its transmission range without interference problems, and one these nodes can transmit messages to ℓ without causing interference problems at ℓ. Both properties are important for Phase III to work correctly.

Another implication of the above theorem is that if any inactive node listens to the channel for a period of cd_1 time steps, then it can gather the ID's of all the active nodes in its transmission radius. This follows because once Phase II stabilizes, all the active nodes are in owner state and hence continue to send their ID messages in the first slot of the round they own. As these messages are free of any collisions, they can be received by the inactive nodes. This characterization is useful especially in Phase III where inactive nodes need this information to organize their data structures.

7.8.2 Phase III - Gateway Discovery

In this section we describe the protocol for Phase III. The goal of this phase is for the active nodes from Phase I to discover gateway connections to other leaders that are within a hop distance of at most 3 in G_{r_t}.

During this phase, the active nodes use an aCST range of r_t. The active nodes use the rounds reserved in Phase II to achieve interference-free communication with the inactive nodes within

their transmission range. Each round consists of four time slots for communication in Phase III, where each time slot represents a communication step as shown in Figure 7.6. In the first time slot, inactive nodes send CLIENT messages and in the second time slot the active node sends a response accordingly; in the third and fourth time slots, an inactive node u may broadcast to its (active and inactive) neighbors all the information it has regarding possible gateways between the leader owning the reserved round and other leader nodes it has heard about. For simplicity, we assume that all active nodes are reactivated at the same time and hence that we can directly compare the time stamps with respect to the different active nodes. In reality, each inactive node u would keep track of the offsets of the (constant number of) time stamps it receives (in the corresponding slots allocated to the different leaders in Phase II) and use these offsets when comparing time stamps from different leaders.

We first describe the data structures that are maintained during this Phase. Each inactive node u maintains a cache, called $\mathcal{P}_u$, which has entries of the form (ℓ, v, t_ℓ) where ℓ is an active node, v is an inactive node (with $u = v$ possibly), and t_ℓ is the time stamp with respect to ℓ at which the entry (ℓ, v) is added to $\mathcal{P}_u$. When comparing entries in the cache, a $*$ acts as a wild card that matches any value. The operation *enqueue*(ℓ, v, t_ℓ) on $\mathcal{P}_u$ is used to add the new entry (ℓ, v, t_ℓ) to $\mathcal{P}_u$. *Enqueue* performs the following checks before actually adding the new entry to $\mathcal{P}_u$. When adding a new entry (ℓ, v, t_ℓ), any entry of the form $(\ell, *, t')$ with $t' < t_\ell$ is evicted. If no such entry exists and $\mathcal{P}_u$ is full, then the least recently added entry $(*, *, t')$, that is $t' = \min\{t | t < t_\ell \text{ and } (*, *, t) \in \mathcal{P}_u\}$, is evicted to make room for the new entry. The cache $\mathcal{P}_u$ has space enough to store a constant, d_2, number of entries. Inactive nodes also maintain a state that is either *awake* or *asleep* with respect to each active node that is within their transmission range. The *asleep* nodes just listen the channel and becomes *awake* when they receive a FREE or a ACK message.

Each active node ℓ maintains a list, called $\mathcal{G}_\ell$, and each entry in $\mathcal{G}_\ell$ contains two fields. The first field has gateways represented as quadruples of the form (ℓ, u, v, ℓ') where $\ell' \neq \ell$ and $u = v$ possibly, ℓ' is an active node and u, v are inactive nodes. The second field contains the time stamp t_ℓ at which the entry was added to $\mathcal{G}_\ell$. The operation *enqueue* on $\mathcal{G}_\ell$ is used to add a new entry $((\ell, u, v, \ell'), t_\ell)$ to $\mathcal{G}_\ell$. Before adding the new entry $((\ell, u, v, \ell'), t)$ to $\mathcal{G}_\ell$, any entry of the form $((\ell, *, *, \ell'), t')$ is evicted from $\mathcal{G}_\ell$ for $t' < t$. If the list $\mathcal{G}_\ell$ is full, then the entry corresponding to t' such that $t' = \min\{t'' | t'' < t \text{ and } ((\ell, *, *, \ell'), \mathrm{t}'') \in \mathcal{G}_\ell\}$, that is the entry of $\mathcal{G}_\ell$ with smallest time stamp, is deleted to make room for the new entry. (Similar to *enqueue* on $\mathcal{P}_u$ for inactive node u). The list $\mathcal{G}_\ell$ has space enough to store a constant, g, number of entries.

In the following, ℓ refers to the ID of the active node that owns the current slot and u is an inactive node that received the ID message from ℓ and the state of u is with respect to ℓ. Each item below represents a communication step.

1. If u is awake then u sends out a CLIENT message of the form $\langle \mathrm{CLIENT}, \ell, u \rangle$ with probability $1/2$.

2. Node ℓ responds with a reply in the next time slot which can be of three forms. If ℓ receives a CLIENT message from node u then ℓ adds u to N_ℓ by calling *enqueue*(u) and also sends an acknowledgment containing the ID of u as $\langle \ell, \mathrm{ACK}, u \rangle$. If ℓ only senses a busy channel but does not receive any message, then ℓ sends a collision message of the form $\langle \ell, \mathrm{COLLISION} \rangle$. If ℓ does not receive any message and also does not sense a busy channel, the ℓ sends a free channel message of the form $\langle \ell, \mathrm{FREE} \rangle$.

   If u is awake and decided not to send a CLIENT message in the previous slot and receives a collision message then u goes to asleep state. If u is asleep and receives a free channel or an acknowledgement message then u becomes awake.

3. If u is awake and receives an acknowledgment containing the ID of u then u will store (ℓ, u, t_ℓ) in $\mathcal{P}_u$, t_ℓ being the current time stamp associated with ℓ, by calling *enqueue*(ℓ, u, t_ℓ).

   Node u also deletes any entries of the form $(*, \ell)$ from $\mathcal{P}_u$ (since ℓ is no longer inactive). Node u then broadcasts, in the third time slot, a message $\langle \text{ADV}, \ell, u, t_\ell \rangle$ to its neighbors. The ADV message is sent with a probability p, to be determined later.

4. Node u builds one GATEWAY message containing all quintuples of the form $((\ell, u, v_j, \ell_j), t)$ for each j such that $\ell_j \neq \ell$ with $(\ell_j, v_j, t_j) \in \mathcal{P}_u$, where $t = \min\{t_\ell, t_j\}$, and sends the message to its neighbors. The GATEWAY message is sent, with probability p, in the fourth time slot.

   If v is not active and received an ADV message $\langle \text{ADV}, \ell, u, t_\ell \rangle$ then it calls *enqueue*(ℓ, u, t_ℓ) on $\mathcal{P}_v$. Node v also deletes any entries of the form $(u, *)$ or $(*, \ell)$ from $\mathcal{P}_v$ (as u is no longer an active node nor is ℓ inactive).

   If ℓ is active and receives a GATEWAY message containing $((\ell, u, v, \ell'), t)$, then ℓ stores $((\ell, u, v, \ell'), t)$ in $\mathcal{G}_\ell$ by calling *enqueue*$((\ell, u, v, \ell'), t)$.

Before we analyze the protocol, we start with the following fact, which follows from the observation that a necessary condition for an inactive node u to transmit in step 3 and step 4 is to receive an ACK from an active node in step 2.

Fact 7.8.2 *During steps 3 and 4 of the protocol there are at most a constant number d_1 of nodes that are transmitting any message.*

Using this fact, we can prove the following theorem.

Theorem 7.8.3 *In $O(D \log^2 n)$ rounds, with high probability, each active node learns about a gateway to each of the currently active nodes in its 3-neighborhood with respect to G_{r_t}.*

Proof. We prove the convergence of Phase III to a set of valid gateway connections in $O(D \log^2 n)$ rounds after Phase I and Phase II have reached a stable state. Since, at that point the active nodes have reserved rounds that are distinct within the $r_i \oplus r_i$ range, we can treat the actions of active nodes independent of each other.

Let (v, ℓ) be an inactive node-active node pair such that v has to send a CLIENT message to ℓ. Node v has at most $O(D)$ inactive nodes in its interference range sending a CLIENT message to some leader node. If more than one node is in awake state, with respect to ℓ, decides to send a CLIENT message, then ℓ will send a collision message. Since the collision message will be received by the inactive nodes, within r_t range of ℓ, awake nodes that decided not to send a CLIENT message to ℓ in the previous slot will go to asleep state.

Consider time to be partitioned into groups of consecutive rounds such that each group ends with a round where the active node ℓ sends either an ACK message or a FREE message. (A group ending with an ACK message signifies a successful group and a group ending with a FREE message is a failed group). Notice that at the end of every group, whether successful or not, all the inactive nodes within the r_t range of ℓ go to awake state (by step 2 of the protocol).

It can be shown that the number of rounds in each group, successful or failed, is $O(\log n)$ and any group is successful with constant probability as follows. Consider any group. Firstly, for an inactive node u in `awake` state, the probability that u stays in the `awake` state in the current round of the group is $1/2$, provided the group does not end in this round. Thus, after r rounds, the probability that there is still a set of $c \geq 2$ nodes that are still in `awake` state is at most:

$$\sum_{j=c}^{D} \binom{D}{j} (1/2)^{rj} \leq \sum_{j=c}^{D} (eD/j2^r)^j \leq 1/n^2$$

when $r = 2 \log n + \log D = O(\log n)$. We now say that each group consists of $r + 1 = O(\log n)$

rounds, with high probability.

Consider the round $r+1$. The probability that the group is successful is at least $\sum_{j=2}^{c} j(1/2)^j \geq 1/2$, as during the $(r+1)$st round, if there are only j `awake` nodes then the probability that only one of them sends a CLIENT message is $1/2^j$. Thus, each group is successful with a constant probability.

Due to symmetry reasons any inactive node is equally likely to be send a CLIENT message in a successful group. Thus, during any successful group, for a given pair (v, ℓ),

$$\Pr[\ v \text{ sends a CLIENT message successfully to } \ell] \geq 1/2D.$$

Using Chernoff bounds, for any given pair (v, ℓ) the probability that it takes more than Dk groups so that v sends a CLIENT message to ℓ successfully will be polynomially small for $k = 3\log n$. Thus any node v requires at most $O(D\log^2 n)$ rounds to send a CLIENT message to ℓ successfully w.h.p.

To proceed further, let ℓ and ℓ' be active nodes, with $d(\ell, \ell') \leq 3$ and let (ℓ, u, v, ℓ') be a gateway between ℓ and ℓ'. Notice that once ℓ and ℓ' receive CLIENT messages from u and v respectively, ℓ and ℓ' can establish a gateway connection between them as successful CLIENT messages are followed by ADV and GATEWAY messages in the next time slots reserved for this phase. Without loss of generality, we assume that u sends the ADV message which when received by v results in v adding the entry (ℓ, u, v, ℓ') to the GATEWAY message that v sends. Along with Fact 7.8.2 it holds that during every successful group the probability that u gets an ACK message and sends the ADV message is at least $1/c'D$ for a constant $c' = 2ed_1$ when we set $p = 1/d_1$. And similarly the probability that in a successful group v gets an ACK message from ℓ' and sends a GATEWAY message is at least $1/c'D$.

Thus, in each group,

$$\Pr[\ell \text{ and } \ell' \text{ discover a gateway connection}] \geq 1/c''D$$

for some constant c'' where c'' depends on c', and $p = 1/d_1$. Using Chernoff bounds again, it holds that ℓ and ℓ' can establish a gateway connection in $3c''D \log n$ groups with high probability. Thus, for ℓ and ℓ' to establish a gateway connection $O(D \log^2 n)$ rounds suffice with high probability.

Note that, after Phase II stabilizes and after we let Phase III run for $O(D \log^2 n)$ rounds, time stamping will be enough to guarantee that we will always keep information received at a leader node ℓ about a valid gateway connection between leader nodes ℓ and ℓ', if at least one such connection exists (since we have at most a constant number of leader nodes within cost $3r_t$ from any given leader node, and since we have at most a constant number of leader nodes adjacent to any inactive node, constant size $\mathcal{P}_u$ and $\mathcal{G}_\ell$ lists at inactive nodes u and active nodes ℓ respectively will suffice).

□

Note that this gateway connection may actually use another node u' adjacent to ℓ in G_{r_t}, in case u received an ADV message from u' later than that of v and before sending the GATEWAY message to ℓ, but that does not affect our calculations in the above proof as they were done for a generic gateway connection.

7.9 Chapter Summary and Acknowledgements

We have introduced a new and more realistic model for wireless communication in this chapter. As will be shown in the subsequent chapters, we can develop time- and energy-efficient protocols for other problems on top of our constant density spanner (e.g. broadcasting and service discov-

ery). In particular, we address the question of how to design protocols that can self-stabilize under adversarial influence using our spanner construction.

A preliminary version of the results in this chapter appear in [75]. This work is done jointly with Melih Onus and Andrea Richa, from the Department of Computer Science, Arizona State University.

Chapter 8

Wireless Ad Hoc Networks: Broadcasting and Gathering

8.1 Introduction

This chapter considers the problem of broadcasting and information gathering in wireless ad-hoc networks. Broadcasting is the problem of sending a packet from a source node in the network to all other nodes in the network. Information gathering is the problem of sending ≥ 1 packets from a subset of the nodes to a single sink node in the network. Broadcasting is one of the most important primitives in wireless networks and it has therefore been extensively studied both in theory and in practice. Information gathering is also an important communication primitive for wireless networks which arises in many contexts such as sensor networks.

Most of the proposed theoretical wireless network models oversimplify wireless communication properties. Such simple models can have a serious effect on the practical efficacy of the proposed algorithms. We will use our model from Chapter 7 that takes into account that nodes have different transmission and interference ranges, and we propose algorithms in this model that achieve a high time- and work-efficiency. Our algorithms have the advantage that they are very simple and self-stabilizing, and would therefore even work in a dynamic environment. Also, our algorithms only require a constant amount of storage at any node. Thus, our algorithms can be used in wireless

systems with very simple devices, such as sensors.

8.2 Motivation

Broadcasting is a basic communication primitive for wireless networks, and it has therefore been heavily studied both in the systems and in the theory community. Though broadcasting itself appears to be an easy problem, it is actually quite hard to realize in an efficient and reliable way in a mobile ad-hoc network. The main problem concerning theoretical investigations is that mobile ad-hoc networks have many features that are hard to model in a clean way. So far, people in the theoretical community have mostly looked at static wireless systems (i.e. the mobile units are always available and do not move).

Broadcasting in wireless networks has been the study of several papers. Recently, the focus is on designing algorithms that assume no knowledge of the topology of the network except possibly the size of the network. Wireless communication is usually modeled using the Packet Radio Network (PRN) model, described in Section 7.2.2.

The PRN model is a simple and clean model that allows to design and analyze broadcast algorithms with a reasonable amount of effort. However, since it is a high-level model, it does have some serious problems with certain scenarios in practice. For example, in reality it is not true that the transmission range, r_t, of a node is the same as its interference range, r_i. Instead, the interference range of a node is usually at least twice as large as its transmission range. Not taking this into account may result in broadcasting algorithms that cannot handle certain scenarios well, although efficient on paper.

When both these assumptions are removed, it is quite challenging to design efficient broadcasting algorithms. The problem becomes more acute as it has been observed in [112] that uncontrolled

additional retransmissions of the message by nodes hinders the actual receipt of message by some nodes due to interference and excessive channel contention. This phenomena is called *broadcast storm* problem in [112].

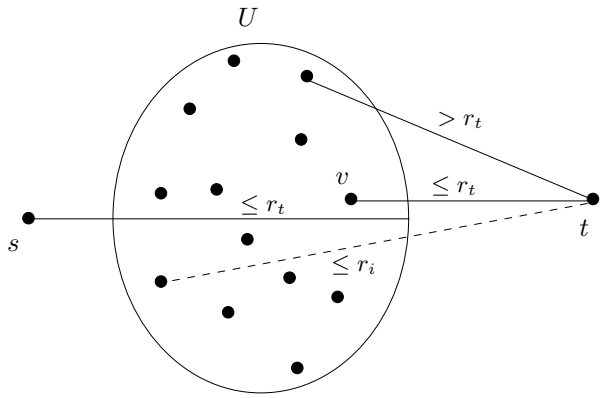


Figure 8.1: An example network with node s the source of the broadcast.

Consider a simple broadcast algorithm for wireless networks where every node that received the message retransmits the message with a probability of $1/2$ during every time step. Now consider for example, a network of n nodes where two nodes s and t and a set U of $n-2$ nodes. Node s is the source of the broadcast message. All nodes in U are within the transmission range of s but only node $v \in U$, is within the transmission range of t as in Figure 8.1. In this situation, when a bigger interference range is not taken into account, node t receives the message when node v transmits after the source node s sends the message. Thus, the broadcast algorithm has a runtime of $O(1)$ in expectation and $O(\log n)$ with high probability.

Let us reconsider the above example when the interference range r_i is bigger than the transmission range r_t. Further let all the nodes in U except v have the property that node t is within their interference range and outside of their transmission range as shown in Figure 8.1. Thus, only node v can successfully deliver the message to t whereas any simultaneous transmission of nodes in U would just result in interference at t. For the above simple broadcast scheme, node t can receive the

message precisely when only node v transmits and rest of the $n-3$ nodes in U do not transmit in a given time step. In this scenario, $\Omega(2^n)$ steps are required in expectation. It could be argued that the above scheme is very simple and hence has poor performance when $r_i > r_t$. However, such a scheme is typical of many algorithms for broadcasting in wireless networks [90, 81, 30] where during every time step there is a set of *active* nodes that have the same probability of transmitting the message. In the above *active* nodes are those that already received the message. Such protocols are also called *uniform* protocols in [66] and the runtime of such protocols in the above example would be $o(n)$ when we consider the situation that $r_i > r_t$.

The assumption that the size of the network or a linear estimate of the size of the network is available to the nodes in the network also over-simplifies the problem. Without an estimate of the size of the network it was shown in [66] that for a single message to be sent successfully $\Omega(n)$ time units are required in expectation, if physical carrier sensing is not available. The reason for this simply is that without knowledge of the size and no physical carrier sensing, when using algorithms that rely on an exponentially decaying transmission probability, nodes do not know when a round ends.

Thus, it is required that algorithms for broadcasting in wireless networks handle the above problems. One way to minimize interference problems is to reduce redundant broadcasts. Heuristics for reducing redundant broadcasts are studied in [112] but a rigorous theoretical analysis is not presented. There is a limited number of papers that use a model that differentiates between the transmission range and interference range [4, 55, 57], but they assume that nodes are distributed in an ideal space so that the transmission range and interference range of every node can be specified in terms of Euclidean distance independent of the position of the node.

We will use our much more general model that is presented in detail in Chapter 7 for designing

self-stabilizing algorithms for wireless overlay networks for broadcasting and information gathering. In this context, self-stabilization means that the algorithm terminates in a finite amount of time at the end of which it also produces a valid output. Our algorithms work without knowledge of size or a linear estimate of size of the network and also can handle interference problems in wireless networks. Our algorithms even work under the condition that the node labels are only locally distinct.

8.3 Related work

Broadcasting in wireless ad-hoc networks has been extensively studied in the literature, especially in the more applied ad-hoc networking community. See [149] for a survey. All of the works on the broadcast problem cited below assume a static network scenario where the transmission and interference ranges of a node are the same and wireless communication is modeled using the PRN or UDG model.

In an early work, Chlamtac and Weinstein [24] presented a deterministic centralized broadcast protocol which assumes complete knowledge of the network topology and which runs in $O(D \log^2 n)$ time, where n is the number of nodes and D is diameter of the network. Bar-Yehuda et al. [11] were the first to present a distributed algorithm for the broadcasting problem in ad-hoc wireless networks. In [11] and in all of the follow-up work cited below, no topological knowledge of the network is assumed. Bar-Yehuda et al. [11] present a randomized protocol, based on a "decay procedure" that mandates when nodes should attempt to retransmit a broadcast message, which has expected completion time $O(D \log n + \log^2 n)$. Later, Kushilevitz and Mansour [90] proved a lower bound of $\Omega(D \log (n/D))$ on the running time of any randomized broadcast protocol, showing that the algorithm of Bar-Yehuda et al. is in fact almost optimal. This was later improved for the case of

symmetric networks (i.e., a node u can directly communicate with a node v if and only if v can also directly communicate with u) by Kowalski and Pelc [81] to obtain a randomized broadcast algorithm with expected time complexity $O(D \log (n/D) + \log^2 n)$. Czumaj and Rytter [30] extended the method by Bar-Yehuda et al. [11] to obtain an optimal randomized algorithm that completes the broadcast in $O(D \log (n/D) + \log^2 n)$ time w.h.p. while not requiring that the network be symmetric.

Adler and Scheideler [4] present approximation algorithms for the unicast problem in wireless ad-hoc networks under the assumption that the transmission and interference ranges are not the same. This model is described in Section 7.2.3. But they still assume a simplified disk model based on Euclidean distances. Their unicast algorithm also does not translate directly into an efficient broadcasting algorithm.

The problem of information gathering in wireless networks is studied mostly in the context of wireless sensor networks. In [61], the authors show a way of constructing a tree on which gathering and aggregation can be performed. However, they do not deal with the problems specific to wireless networks such as channel contention and interference and also do not provide theoretical bounds on time and work. Data gathering in sensor networks where the sensor nodes are placed at the vertices of a 2-dimensional grid with the sink node at the center of the grid is studied in [38]. In [78], the authors study gathering in simple topologies such as the line network and the cycle network. An online algorithm is presented where the buffer overhead only needs to be by a logarithmic factor higher than that of an optimal offline algorithm in order to achieve the same throughput. Also in [35], the authors study the throughput achievable in a wireless sensor network for data gathering. Information gathering and aggregation has been studied experimentally in [61, 104, 153, 58] but a rigorous formal analysis for wireless ad hoc networks has not been presented.

8.4 Our results

We consider two important communication problems in wireless ad hoc networks, namely, broadcasting and information gathering. In the following, we first formally define the problems considered in this chapter and then outline some of the key properties that our algorithms achieve. We also note that our algorithms do not require many of the assumptions that are commonly made in the existing literature.

Broadcasting

The problem of broadcasting arises in many scenarios where information has to be disseminated to all the participants. It is thus not surprising that efficient broadcasting in wireless ad hoc networks has attracted considerable attention.

Broadcasting: Given a static connected wireless network of n nodes, minimize the total time and work to send $m \geq 1$ broadcast messages originating from a source node s to all the nodes in the network.

Isolated Broadcasting Given any fixed node distribution with some source node s and any fixed transmission range r_t, let $D(s)$ denote the maximum distance (in number of hops) of a node from s with respect to transmission range r_t. We achieve the following results with respect to the case of broadcasting a single message, i.e., $m = 1$:

Theorem 8.4.1 *Given a constant density spanner as in Chapter 7, the isolated broadcast algorithm needs $O(D(s) + \log n)$ time w.h.p., to send a broadcast message from s to all the nodes in the network.*

Theorem 8.4.2 *Given a constant density spanner as in Chapter 7, the isolated broadcast algorithm needs $O(W(s))$ work to send a broadcast message to all nodes in the system, where $W(s)$ is the optimal work required to send a broadcast message from s to all the nodes.*

Broadcasting multiple messages We also show how to extend the isolated broadcast algorithm to achieve the following result for the case where the source node s has to broadcast m messages.

Theorem 8.4.3 *Given a constant density spanner as in Chapter 7, the concurrent broadcast algorithm needs $O(D(s) + m + \log n)$ time steps, with high probability, to deliver m broadcast messages to all nodes.*

Theorem 8.4.4 *Given a constant density spanner as in Chapter 7, the multiple broadcast algorithm needs $O(W(s, m))$ work, where $W(s, m)$ is the optimal work required to send m from s to all the nodes.*

Information Gathering

Information gathering is another important communication primitive in wireless networks. The problem has applications in many scenarios such as data gathering in sensor networks [61, 153, 38] and maintaining connectivity with base stations in a multi-hop wireless network. (The related problem of how best to aggregate data at intermediate nodes is not discussed in this paper.)

Information Gathering: Given a static connected wireless network of n nodes among which m packets are arbitrarily distributed, and a sink node s in the network, minimize the total time and work required for sending the m packets to the sink node.

We analyze a simple two-stage strategy that has the following time and work bounds:

Theorem 8.4.5 *Given a constant density spanner as in Chapter 7, the information gathering protocol needs $O(D(s) + \log n + m + \Delta_m \log^2 n)$ time steps w.h.p until all the packets are delivered to the sink node s. Here, Δ_m refers to the density of inactive nodes that have a packet to send.*

Theorem 8.4.6 *Given a constant density spanner as in Chapter 7, the gathering protocol needs $O(W'(s,m))$ work where $W'(s,m)$ is the optimal work required to send all the m packets to the sink node s.*

Our algorithms are self-stabilizing (i.e., can start in an arbitrary state) and can therefore adapt to changes in a wireless ad-hoc network. Our algorithms do not require any knowledge of the size of the network. For our algorithms to work correctly, it suffices that the nodes in the network have identifiers that are locally distinct. We only require that the nodes synchronize up to some reasonably small time difference, which can be easily accomplished using GPS signals or any form of beacons. Another important feature of our algorithms is that a constant amount of storage at any node suffices even in the case of gathering. The above properties make our algorithms applicable to sensor networks without any modifications.

Our results build on top of the distributed algorithm for organizing the wireless nodes into a constant density spanner presented in Chapter 7. The remainder of this chapter is organized as follows. In Section 8.4.1 we introduce the notations used in the rest of this chapter. The isolated broadcast algorithm and the proofs of the respective theorems can be found in Section 8.5. The multiple broadcast case is addressed in Section 8.6. The algorithm and proofs for the information gathering problem appear in Section 8.7.

8.4.1 Notation

In this section, we present the notation that is used in the rest of this chapter. Let V be the set of nodes in the network. For any transmission range r, let the graph $G_r = (V, E)$ denote the graph containing all edges $\{v, w\}$ with $c(v, w) \leq r$. Throughout, r_t denotes the transmission range and $d(u, v)$ denotes the shortest distance between u and v in G_{r_t}. Furthermore, given any node $s \in V$, $D(s)$ denotes the maximum distance of any node in G_{r_t} to the node $s \in V$, i.e., $D(s) = \max_{v \in V} d(s, v)$. We also denote by G' the constant density spanner obtained using the protocol presented in Chapter 7. Let U refer to the set of active nodes in G', and $\mathcal{G}$ refer to the set of gateway nodes. Recall from Chapter 7 that the constant d_1 refers to the number of active nodes that are within the interference range, r_i, of any node.

8.5 Isolated Broadcasting

Let node s be the source of the broadcast message. Since s has a maximum distance of $D(s)$ to any node in G_{r_t}, $D(s)$ is a lower bound on the time an optimal offline algorithm needs to broadcast a message from s to all nodes. Our goal is to come up with a broadcast scheme so that the time needed by the broadcast message to reach all nodes is as close to $D(s)$ as possible. We use the constant density spanner construction of Chapter 7 as the basis. If s is not an active node, i.e., $s \notin U$, then let ℓ be some active node that is within the transmission range of s. Then s first sends the message to ℓ. The broadcast scheme then proceeds in rounds that are synchronized among the nodes. In the broadcast scheme below, ℓ refers to the ID of an active node that owns the current slot. Every item below is a separate time step.

1. If ℓ received the broadcast message in the previous round and it is the first time it received the broadcast message, ℓ sends out the broadcast message.

2. If v is a gateway node and has already received the broadcast message, then v sends out an RTS (Request-To-Send) message with probability p.

3. If v is a gateway node and decided not to send out an RTS message or v is an active node, then v checks if it correctly received an RTS message. If so, and v has not received the broadcast message yet, v sends out a CTS (Clear-To-Send) message.

4. If v is a gateway node and sent out an RTS message, then v checks if it sensed a CTS message. If so, v sends out the broadcast message.

Notice that inactive nodes just need to listen to the wireless channel in order to receive the broadcast message eventually. This is because our spanner algorithm of Chapter 7 makes sure that message transmissions of active nodes in step 1 above never interfere at an inactive node. Thus only nodes in $U \cup \mathcal{G}$ may need to retransmit the message.

The following theorem directly implies Theorem 8.4.1.

Theorem 8.5.1 *Given the constant density spanner of G_{r_t} as described in Chapter 7, the broadcast algorithm with $p = 1/d_1$ needs $O(D(s) + \log n)$ rounds, with high probability, to deliver the broadcast message to all nodes.*

Proof. Recall that every message has to traverse a path of length at most $5D(s)$ to reach any node via the nodes in $U \cup \mathcal{G}$. Hence, consider any fixed node v, and let P be any shortest path from s to v via the dominating set. Let $\alpha \leq 5D(s)$ be the length of P. Suppose that it takes at least $\alpha + \delta$ time steps to send the broadcast message to v. Then there must have been at least δ time steps in which a node in P failed to forward the message to the next node in P, and the time steps can be associated with nodes of monotonically increasing order in P (by always looking at the node of maximum distance from s in P that failed to forward the packet at any given time step). The number

of possibilities to assign δ out of the $\alpha + \delta$ time steps to nodes in P in increasing order is $\binom{\alpha+\delta}{\delta}$, which is equal to the number of all binary sequences of length $\alpha + \delta$ with δ 1's. A 0 represents the event "move to the next node in P" and a 1 represents the event "failed transmission" at the current node in P. Since the probability of a node v to fail is equal to

$$\begin{aligned}
&\Pr[v \text{ did not decide to transmit the message}] + \Pr[v \text{ transmitted the message but failed}] \\
&\leq \quad (1-p) + p(1-(1-p)^{d_1-1}) = (1-p) + p(1-(1-1/d_1)^{d_1-1}) \leq 1 - p + p(1-1/e) \\
&= \quad 1 - p/e \ ,
\end{aligned}$$

it follows that the probability that the broadcast message needs more than $\alpha + \delta$ steps with $\delta = b \cdot \alpha$ is at most

$$\binom{\alpha+\delta}{\delta}(1-p/e)^{\delta} \leq \left(\frac{e(\alpha+\delta)}{\alpha}\right)^{\alpha} \cdot (1-p/e)^{\delta} \leq \left(e(1+b)e^{-pb/e}\right)^{\alpha}$$

which is polynomially small in n if $b = \Omega((1 + \log n/\alpha)/p)$. Thus, it takes $O(D(s) + \log n)$ time steps until the broadcast message reaches any particular node v, w.h.p., which completes the proof of the theorem. □

8.5.1 Work Efficiency

Next we consider the work efficiency of the broadcast algorithm. We assume that the cost for sending and sensing the RTS/CTS message is negligible so that we only need to bound the cost for retransmitting the broadcast message.

Suppose that the area covered by disks of radius r_t around all nodes in the system is A. Then a lower bound for the number of message transmissions of an optimal offline algorithm is

$|A|/(4\pi {r_t}^2)$. Next we prove an asymptotically matching upper bound for our broadcast algorithm. We show the following theorem.

Theorem 8.5.2 *Given the constant density spanner of G_{r_t} as described in Chapter 7, the broadcast algorithm needs $O(|A|/{r_t}^2)$ work to send a broadcast message from s to all nodes.*

Proof. Due to the constant density of the network of active nodes, we need $\Omega(|A|/{r_t}^2)$ transmissions of the broadcast message in order to reach all active nodes. Each active node has a fixed time slot, and thus one transmission from each active node is enough for all inactive nodes to receive the message. Since the spanner construction from Chapter 7 results in a spanner of constant density, $O(|A|/{r_t}^2)$ messages will suffice to deliver the broadcast message to all inactive nodes.

It remains to show that the work necessary for the active nodes to receive the message (from gateway nodes) is also $O(|A|/{r_t}^2)$. Note that an active or gateway node only sends a CTS message if a broadcast message can be sent to it without interfering with other transmissions. Since every active or gateway node sends a CTS at most once, the number of message transmissions necessary for all active or gateway nodes to receive the message is equal to the total number of active and gateway nodes, which is $O(|A|/{r_t}^2)$.

Thus, the broadcast algorithm will send $O(|A|/{r_t}^2)$ messages to send the broadcast message to all the nodes in the network. □

8.5.2 Self-stabilization

The broadcast algorithm can also provide self-stabilization. We add a fifth step to the broadcast algorithm in which if a node v just woke up or moved, then v sends an AWAKE signal. Furthermore, we replace the first step with the following step:

- If ℓ is active and received the broadcast message in the previous round and it is the first time

it received the broadcast message or if ℓ sensed an AWAKE signal in the previous round or if ℓ just became active in this round, ℓ sends out the broadcast message.

8.6 Broadcasting Multiple Messages

Next we look at the case that the source s wants to send out multiple broadcast messages instead of just one. Then s attaches continuous sequence numbers to the messages, starting with 1.

The broadcast scheme proceeds in rounds that are synchronized among the nodes. Each active or gateway node v keeps track of two numbers, i_v and j_v. Number i_v denotes the minimum message number v has not received so far and number j_v denotes the minimum message number (v knows about since its last successful transmission attempt) a node of distance at most r_t from v has not received so far. In the broadcast scheme below, ℓ refers to the ID of an active node that owns the current slot. Initially, for each gateway and active node v, i_v=j_v=1. In each round, every node $v \neq s$ does the following. Each item below represents a separate time step.

1. If ℓ received the broadcast message with sequence number $i' = i_\ell$ in the previous round, then it sets $i_\ell = i_\ell + 1$ and sends out the broadcast message with sequence number i'.

   If v is a gateway node and received a broadcast message with sequence number $i' = i_v$, then it sets $i_v = i_v + 1$.

2. If v is an active or gateway node, then it sends out an $\langle \text{RTR}, i_v \rangle$ message (RTR means "Ready-To-Receive") with probability p. If v decides not to send out an RTR message, it checks whether it is able to receive an $\langle \text{RTR}, i' \rangle$ message. If so, it sets $j_v = \min\{j_v, i'\}$.

3. If v is a gateway node and $i_v > j_v$, then it sends out an $\langle \text{RTS}, j_v \rangle$ message with probability p.

   If v is a gateway node and decided not to send out an RTS message or v is an active node,

then v checks if it correctly received an $\langle \text{RTS}, j' \rangle$ message with $j' = i_v$. If so, v sends out a CTS message.

4. If v is a gateway node and sent out an $\langle \text{RTS}, j_v \rangle$ message, then v checks if it sensed a CTS message. If so, v sends out the broadcast message with sequence number j_v. Afterwards, v sets $j_v = \min\{j_v + 1, i_v - 1\}$. If v is a gateway node and did not send a message but received a broadcast message with sequence number $i' = i_v$, then it sets $i_v = i_v + 1$.

The source node s uses the same protocol as above with the only difference that it only executes the first step. As in the case of isolated broadcast algorithm, the inactive nodes just need to listen to the wireless channel in order to receive the broadcast messages eventually. Only active nodes and gateway nodes may retransmit the messages.

8.6.1 Time Efficiency

The following theorem demonstrates that this protocol has a high time efficiency. Recall that d_1 refers to the number of active nodes that are within the interference range, r_i, of any node.

Theorem 8.6.1 *Given the constant density spanner of G_{r_t} as described in Chapter 7, the multiple broadcast algorithm with $p = 1/d_1$ needs $O(D(s) + m + \log n)$ rounds, with high probability, to deliver m broadcast messages to all nodes.*

Proof. The proof involves substantial extensions to the proof of Theorem 8.4.1. We use a delay sequence argument to prove the theorem. For simplicity, we assume each round of the algorithm takes one time unit. Let v be the last node that received all broadcast messages in $U \cup \mathcal{G}$, let $P = (u_0 = s, u_1, \ldots, u_d = v)$ be any shortest path of active and gateway nodes from s to v, and let $\alpha \leq 5D(s)$ be the length of P. Suppose that it takes t_m time steps for all broadcast messages

to reach v. Given a node w, let $m(w)$ denote the number of messages node w has already received. We go backwards in time from the moment the last message reached v and move monotonically backwards along P from $v_m = v$ so that at any time we stay at a node u_j with

- $m(u_j) = m$,
- $m(w) = m - 1$ for some node w within range r_t of u_j, and
- $m(w) \geq m - 1$ for all nodes w within range r_t of u_j.

Such a node is called m-active. This is done until we reach the first time step t_{m-1} at which there is a node $u_{j'}$ with $j' \leq j$ that is not m-active. We then set $u_{j'} = v_{m-1}$. An m-active node must always exist at any time from $t_{m-1} + 1$ to t_m because within these steps, every node from s to the current u_j must have only neighbors w with $m(w) \geq m - 1$, s has all m messages, and at step t_m, u_{d-1} has initially at least one neighbor w with $m(w) = m - 1$, namely v.

From step t_{m-1}, we go backwards in time and move monotonically backwards along P from v_{m-1} so that at any time we stay at a node u_j that is currently $m - 1$-active. This is done until we reach the first time step t_{m-2} at which there is a node $u_{j'}$ with $j' \leq j$ that is only $m - 2$-active. We then set this $u_{j'} = v_{m-2}$. Again, an $m - 1$-active node must always exist at any time from $t_{m-2} + 1$ to t_{m-1} because within these steps, every node from s to the current u_j must have only neighbors w with $m(w) \geq m - 2$, s has all m messages, and at step t_{m-1}, the node before v_{m-1} has initially at least one neighbor w with $m(w) = m - 2$, namely v_{m-1}.

Continuing with this argument gives us a sequence of nodes $v_1, \ldots, v_m$ of monotonic order from s to v and a sequence of time steps $1 \leq t_1 < t_2 < \ldots < t_{m-1} < t_m$ with the property that between $t_i + 1$ and t_{i+1} there is a monotonic sequence of $i + 1$-active nodes from v_i to v_{i+1}. For any i, at most $d_1 \cdot l_i$ of the time steps from $t_i + 1$ to t_{i+1} can represent successful transmissions (i.e.

message $i+1$ was successfully sent from u_j to some active or gateway node w within range r_t of u_j), where l_i is the distance along P from v_i to v_{i+1}. This is because every active or gateway node between v_i and v_{i+1} has at most d_1 other active or gateway nodes.

Hence, altogether, there can be at most $d_1\alpha$ successful transmissions. All others must have failed. All transmissions of active nodes are successful, since they have fixed time slots. For gateway nodes, the probability of a transmission to fail is at most

$$\begin{aligned}
&\Pr[u_j \text{ did not decide to send a message}]+\\
&\Pr[u_j \text{ decided to send a message with an incorrect sequence number}]+\\
&\Pr[u_j \text{ decided to send a message but failed}]\\
&\quad \le (1-p)+p((1-p)+p(1-(1-p)^{d_1-1}))+(p(1-(1-p)^{d_1-1}))\\
&\quad \le 1-(p/9)
\end{aligned}$$

Thus, the probability that the broadcast message needs more than $S+\delta$ steps, $S = d_1\alpha + m$ with $\delta = b \cdot S$ is at most

$$\binom{S+\delta}{\delta}\binom{S}{\alpha}(1-p/9)^{\delta} \le \left(\frac{e(S+\delta)}{S}\right)^{S}\left(\frac{eS}{S-\alpha}\right)^{S-\alpha}(1-p/9)^{b\cdot S} \le (3e(1+b)e^{-pb/9})^{S}$$

This is polynomially small in n if $b = \Omega((1+\log n/S)/p)$. Thus, in $O(D(s)+m+\log n)$ time steps all broadcast messages reach all active and gateway nodes, with high probability. Since each active node has a fixed time slot, m transmissions from each active node is enough for inactive nodes to receive all messages. Thus, in $O(D(s)+m+\log n)$ time steps, all nodes will receive all m messages, w.h.p.. □

8.6.2 Work Efficiency

The following theorem demonstrates that this protocol has a high time efficiency.

Theorem 8.6.2 *Given a constant density spanner of G_{r_t} as described in Chapter 7, the multiple broadcast algorithm needs $O(m|A|/{r_t}^2)$ work to send all m broadcast messages to all nodes in the system.*

Proof. The proof is very similar to the proof of Theorem 8.5.2. A lower bound on the number of message transmissions of an optimal offline algorithm is $\Omega(m|A|/{r_t}^2)$. Similarly, $O(m|A|/{r_t}^2)$ messages will be sufficient to deliver m broadcast messages to all inactive nodes and $O(m|A|/{r_t}^2)$ messages will be sufficient to deliver m broadcast messages to all active nodes. Thus, the broadcast algorithm will need $O(m|A|/{r_t}^2)$ messages w.h.p., to send m broadcast messages to all nodes in the system. □

8.6.3 Self-stabilization

The multiple broadcast algorithm can also provide self-stabilization. We can add a fifth step to the broadcast algorithm in which if a node v just woke up or moved, then v sends an AWAKE signal. Each active node v also keeps track of number k_v, which is initially equal to 1. Number k_v denotes the minimum message number a newly awake or arrived node of distance at most r_t from v has not received so far. We replace the first step with the following step:

- If ℓ received the broadcast message with sequence number $i' = i_\ell$ in the previous round, then it sets $i_\ell = i_\ell + 1$ and sends out the broadcast message with sequence number i'.

  If ℓ sensed an AWAKE signal in the previous round, it sets $k_\ell = 1$. If $k_\ell < i_\ell$, ℓ sends out the broadcast message with sequence number k_ℓ and sets $k_\ell = k_\ell + 1$.

8.7 Information Gathering

In this section we consider the problem of information gathering in ad hoc wireless networks. Information gathering is an important communication primitive in networks where all the packets have the same destination called the *sink*. Thus, one can view the process of information gathering as the reverse process of broadcasting from the sink node. We consider the situation where a total of m packets distributed in an arbitrary way among the nodes in the wireless network are to be delivered to a sink node s in the network. We seek bounds on the time and work required for all the m packets to reach the sink. Firstly, we comment that $\Omega(m)$ is a lower bound in the case of wireless ad hoc networks since the sink node can accept at most one message during every time step. Similarly, $D(s)$ is also a lower bound on the number of time steps required. Thus, $\Omega(m + D(s))$ is a lower bound on any solution for the information gathering problem. In the following, we show how to perform information gathering efficiently.

Our solution has two stages. In stage 1, we first build a tree rooted at s in G_{r_t}, called the *gathering tree*, $T(s)$, and establish some of the properties of $T(s)$. In stage 2, we show how to perform information gathering on $T(s)$ and prove time bounds for delivering the m messages to s. Each stage has 4 time slots reserved. We however note that for stage 1 to work correctly, the active nodes need not wait for their owned timed slot to transmit ROUTE messages. This is because of the fact that active nodes that have a ROUTE message to send are always at a distance $r_t \oplus r_i$ apart and hence their transmissions do not interfere. By having time slots it is however easy to integrate the following protocols along with the others.

8.7.1 Stage 1: Building Gathering Tree $T(s)$

We first show how to build the gathering tree. The sink node s, if it is not in $U \cup \mathcal{G}$, selects an active node ℓ such that $d(\ell, s) = 1$ and sends a route packet to ℓ with sequence number of 0 of the form $\langle \text{ROUTE}, s, 0 \rangle$. The rest of the nodes do the following. Initially, $d'(s, v) = \infty, \pi(v) = \text{NULL}$ for all $v \in U \cup \mathcal{G}$ where $d'(s, v)$ is an upper bound on the distance of s to v and $\pi(v)$ denotes the predecessor of v in a path from v to s and $d'(s, s) = 0$ and $\pi(s) = s$.

1. If $u \in U \cup \mathcal{G}$ receives a message $\langle \text{ROUTE}, v, d'(s, v) \rangle$ from v with a sequence number of $d'(s, v)$ and if $d'(s, v) + 1 < d'(s, u)$, then u sets $\pi(u) = v$ and $d'(s, u) = d'(s, v) + 1$. Node u also sets flag$(u) = 1$ in this case, indicating that u has to send a route message since u updated its predecessor. If u is inactive and $d'(s, v) + 1 < d'(s, u)$ and $v \in U$, then u updates $\pi(u) = v$ and $d'(s, u) = d'(s, v) + 1$.

2. If ℓ is active and $d'(s, \ell) \neq \infty$ and flag$(\ell) = 1$, then ℓ sends a ROUTE message of the form $\langle \text{ROUTE}, \ell, d'(s, \ell) \rangle$ and sets flag$(v) = 0$ signifying that the update has been notified.

   If $u \in \mathcal{G}$ and $d'(s, u) \neq \infty$ and flag$(u) = 1$ then u sends a RTS message with probability p to be determined later.

3. If $u \in U \cup \mathcal{G}$ and u received a RTS message then u sends a CTS message.

4. If $v \in \mathcal{G}$ and v sent a RTS message and receives a CTS message then v sends a route message $\langle \text{ROUTE}, v, d'(s, v) \rangle$ and sets flag$(v) = 0$ signifying that the update has been notified.

The above construction has the following properties. Let $T' = (V', E')$ be a graph with $V' = U \cup \{s\} \cup \mathcal{G}$ and $E' = \{(v, \pi(v)) | v \in V'\}$. Then, T' is a shortest path tree rooted at s for the graph $G'' = (V'', E'')$ with $V'' = V'$ and $E'' = E \cap ((U \cup \{s\}) \times \mathcal{G})$. We set $T(s) = (V_T, E_T)$ with $V_T = V$ and $E_T = \{(v, \pi(v)) | v \in V_T\}$ where $\pi(v)$ is set as in step (1) of the above protocol.

Since the graph G' defined in Section 8.4.1 is a 5-spanner of the original network G_{r_t}, it also holds that $\max_{v \in V}\ d_{T(s)}(s,v) \leq 5\ \max_{v \in V}\ d(s,v)$. Thus the maximum distance of any node from s in $T(s)$ is only a constant factor away from the maximum distance of any node from s in G.

Finally, we show the following lemma for the time steps required to construct $T(s)$ using the above protocol.

Lemma 8.7.1 *Given a connected dominating set of active and gateway nodes with constant density d_1 in G_{r_t}, the protocol to construct the gathering tree $T(s)$ given above takes at most $O(\log n + D(s))$ time steps w.h.p. when the probability p is set to $1/d_1$.*

Proof. The proof follows from the following claim.

Claim 8.7.2 *After $O(d + \log n)$ time steps it holds that w.h.p., all nodes $v \in U \cup \mathcal{G}$ that are at a distance d from s in G'' have their predecessor pointer $\pi(v)$ such that the path from v to s has length of d.*

Proof. The proof of the claim follows by induction on d starting from $d = 0$. Here only s is the node that has a distance 0 from s and the claim holds in this case due to the initialization. For the induction step, assume that all nodes with distance d have their predecessor correctly assigned. For any node u at distance $d + 1$ from s in G'' let it take more than $(d + 1) + \delta$ time steps to have $\pi(u)$ set correctly. Then in a path P of length $d + 1$ from u to s, we can associate δ nodes that are monotonic in distance from s that failed to forward a route packet. Using calculations similar to that of Theorem 8.5.1, then a value of $\delta = O(\log n)$ suffices to get the claim hold w.h.p. □

Once we have the above claim, then the lemma follows as the maximum distance of any active or gateway node from s is $D(s)$. Inactive nodes do not have to send any route messages but can use step 1 of the protocol to set their predecessor pointer. □

8.7.2 Stage 2: Gathering on $T(s)$

In this section we show how to use the gathering tree constructed in stage 1 to perform information gathering. In $T(s)$, each node has an unique path to the sink node s by following the predecessor pointers π. Nodes use this path system to eventually deliver packets to s. However, in our model, complications arise due to the fact that a lot of inactive nodes choose the same active node as their predecessor. All such nodes transmitting simultaneously will result in interference. Hence, we proceed as follows.

Of the 4 slots available for this stage, the active node uses the first time slot to deliver packets and the second and third time slots are used to coordinate the actions of the inactive nodes.

Nodes $\ell \in U \cup \mathcal{G}$ have a queue called Q_ℓ which can hold a constant number of packets. This queue works as a first-in-first-out list and supports operations *enqueue* and *dequeue* which add a packet and return a packet respectively to the queue Q_ℓ.

In the following, we only consider inactive nodes that have a packet. Thus when we refer to inactive nodes, it is those inactive nodes that have a packet to send. Inactive nodes have a state among $\{\texttt{awake}, \texttt{asleep}\}$. Initially all inactive nodes are in the `asleep` state.

1. If ℓ is active and has an non-empty queue, then ℓ sends the packet *dequeue*(Q_ℓ) during the time slot owned by ℓ. This packet has a destination $\pi(\ell)$ and nodes other than $\pi(\ell)$ discard the packet and $\pi(\ell)$ stores the packet by calling *enqueue* on $Q_{\pi(\ell)}$. In the second time slot, the active nodes listen to the channel.

   If g is a gateway node and has a non-empty queue then g sends an RTS message containing the id of $\pi(g)$ with probability p, where p is to be determined later.

2. If $u \in U \cup \mathcal{G}$ and Q_u is not full and u receives an RTS message containing the id of u then u sends a CTS message.

If u is inactive and has a packet to send and u is `awake` then u sends an I-RTS *(for Inactive-RTS)* message to $\pi(u)$ with a probability $1/2$.

3. If g is a gateway node and sent an RTS message in the previous time step and receives a CTS message from $\pi(g)$ then g sends the packet *dequeue*(Q_g) to $\pi(g)$. This packet has a destination $\pi(g)$ and other nodes that receive the packet ignore it.

   If ℓ is active and ℓ receives an I-RTS message from an inactive node u then ℓ sends an I-CTS message. If ℓ senses a busy channel but does not receive any I-RTS message, then ℓ sends a collision message of the form $\langle \ell, \text{COLLIDE} \rangle$. Otherwise if ℓ senses a free channel then ℓ sends a free message of the form $\langle \ell, \text{FREE} \rangle$. These messages are sent during the third reserved time slot.

4. If u is inactive and `asleep` and receives a free message then u becomes `awake`. If u is inactive and decided not to send an I-RTS message in the previous time step and u is `awake` and receives a collision message then u decides to go to `asleep` state with probability $1/2$. If u is inactive and sent an I-RTS in the earlier step and gets an I-CTS then u sends the packet to $\pi(u)$.

We show in the following that the gathering protocol described above is efficient in terms of the time and work, and also it suffices for nodes in $U \cup \mathcal{G}$ to have a queue that can store a constant number of packets in transit. We use the parameter Δ_m defined to be the density of nodes that have a packet to send. That is, Δ_m refers to the maximum number of nodes that are within a disk of radius r_t around any node that has a packet to send to s.

Lemma 8.7.3 *For any inactive node v, it takes $O(\Delta_m \log^2 n)$ time steps w.h.p, until v sends the packet to an active node that is within the transmission range of v.*

Proof. The proof is similar to that of the proof of Theorem 7.8.3. Given a stable gathering tree, the active nodes have reserved rounds that are distinct within the $r_i \oplus r_i$ range, we can treat the actions of active nodes independent of each other. In the following, we can therefore concentrate on a specific active node, say ℓ.

Consider any inactive node-active node pair, (v, ℓ). Node v has at most $O(\Delta_m)$ inactive nodes in its interference range sending an I-RTS message to some leader node. If more than one node is in awake state, with respect to ℓ, decides to send an I-RTS message, then ℓ will send a COLLIDE message. Since the COLLIDE message will be received by the inactive nodes within r_t range of ℓ awake nodes that decided not to send a CLIENT message to ℓ in the previous slot will go to asleep state.

Consider time to be partitioned into groups of consecutive rounds such that each group ends with a round where the active node ℓ sends either a COLLIDE message or an I-CTS message. A group ending with a COLLIDE message signifies a failed group and a group ending with an I-CTS message is a successful group. Notice that at the end of every group, whether successful or not, all the inactive nodes within the r_t range of ℓ go to awake state (by step 3 of the protocol).

It is not difficult to show that the expected number of rounds in each group, successful or failed, is $O(\log n)$ and any group is successful with constant probability. Due to symmetry reasons any inactive node is equally likely to be send an I-RTS message in a successful group. Thus, during any successful group, for a given pair (v, ℓ) ,

$$\Pr[\ v \text{ sends an I-RTS message successfully to } \ell] \geq 1/c\Delta_m$$

for some constant $c > 1$.

There are Δ_m inactive nodes that compete to send a packet, and due to symmetry reasons,

it then follows that there is a constant probability that in $O(\Delta_m \log n)$ time steps any particular inactive node can send the packet to the corresponding active node. Using Chernoff bounds the stated claim follows. □

For the queue size, node $u \in U \cup \mathcal{G}$ can associate slots in the queue with distinct nodes in $U \cup \mathcal{G}$ that are its neighbors in $T(s)$. Since there are only $O(d_1)$ such neighbors and nodes have a constant probability of forwarding a packet during each time step, it suffices to have a queue of size $O(d_1^2)$ so that w.h.p. all packets in transit can be accommodated.

We can show the following time bound for any packet to reach s.

Lemma 8.7.4 *Given a connected dominating set of active nodes with constant density at most d_1 in G_{r_t} and a gathering tree $T(s)$ with sink node s, the information gathering algorithm presented above with $p = 1/d_1$ needs at most $O(m + \Delta_m \log^2 n + D(s) + \log n)$ time steps w.h.p so that all the m packets reach the sink s.*

Proof. We use a technique that is similar to the proof of Theorem 8.6.1. Also, once a packet that is at an inactive node reaches an active node, then it stays in the queue of nodes in $U \cup \mathcal{G}$ only before it reaches s. Thus we can consider the time required for any packet as the sum of the time required to reach a nearest active node and the time required to reach s through nodes in $U \cup \mathcal{G}$. The former is bounded by Lemma 8.7.3 for those packets that start from an inactive node. Thus it is left to bound the latter.

For the latter, we proceed as follows. Using the gathering tree, every packet that has to reach s has to travel a distance of at most $5D(s)$. Let the last packet P to reach s arrive at time T_0 and let $u_0 = s$. The packet P then must have reached some node in $u_1 \in U \cup \mathcal{G}$ at time T_1 and is still at node u_1 during all the time steps between T_1 and $T_0 - 1$. Continuing the history of P further, it must have reached some node $u_2 \in U \cup \mathcal{G}$ at time T_2 and during the time steps between T_2 and

$T_1 - 1$ is at node u_2. We can similarly follow the history of P till it reached a node in $U \cup \mathcal{G}$ for the first time. (It could be that the packet is initially at some node in $U \cup \mathcal{G}$ or it has reached a node in $U \cup \mathcal{G}$ from an inactive node.) This means that all the attempts to forward the packet from u_i to u_{i-1} in the above sequence failed between time steps T_i and $T_{i-1} - 1$.

Since the active nodes use reserved time slots, the transmissions of active nodes cannot fail. For a gateway node g, the probability that a transmission fails is at most:

$$\begin{aligned}
&\Pr[g \text{ did not decide to transmit the message}] + \Pr[g \text{ transmitted the message but failed}] \\
&\leq \quad (1-p) + p(1-(1-p)^{d_1-1}) = (1-p) + p(1-(1-1/d_1)^{d_1-1}) \leq 1 - p + p(1-1/e) \\
&= \quad 1 - p/e \ ,
\end{aligned}$$

for the chosen value of $p = 1/d_1$. Now assume that it took more than $S + \delta$ time steps for the last packet to reach s for some value of $\delta = bS$ with $S = 5d_1 D(s) + m$. Out of these, at most S attempts must have been successful and all the δ attempts have failed. The probability of this happening is at most:

$$\binom{S+\delta}{\delta} (1-(p/e))^{\delta} \leq \left(\frac{e(S+\delta)}{S}\right)^{S} \cdot (1-(p/e))^{\delta} \leq \left(e(1+b)e^{-pb/e}\right)^{S}$$

The above probability is polynomially small if $b = \Omega((1 + \log n/S)1/p)$. Hence, for the packets to reach s, it takes at most $O(D(s) + m + \log n)$ time steps once the packets have reached some node in $U \cup \mathcal{G}$.

Now add both the time bounds, we get the final bound as stated. □

Finally, ignoring the work performed while sending the RTS/CTS message and the I-RTS/I-

CTS message, since we use paths that are only a constant factor away from the shortest paths to s, we get the following result with respect to work.

Theorem 8.7.5 *Once a stable gathering tree has been constructed, the gathering protocol described above needs $O(W'(s,m))$ work to send all the m messages to the sink node s where $W'(s,m)$ denotes the optimal work required to send the m packets to the sink node s.*

Proof. Ignoring the work done to send I-RTS message, it holds that any inactive node u performs optimal work to forward a packet to $\pi(u)$.

For nodes $U \cup \mathcal{G}$ we proceed as follows. We ignore the work done to send/receive RTS/CTS message. Each active node has a fixed time slot, and thus one transmission from each active node is enough to forward the message from the active node ℓ to $\pi(\ell)$ in the gathering tree. It remains to show that the work necessary for the gateway nodes to forward the message (from gateway nodes) in the gathering tree. Note that an active or gateway node only sends a CTS message if a message can be sent to it without interfering with other transmissions. Since every active or gateway node sends a CTS at most once, the number of message transmissions necessary for all active or gateway nodes to receive the message is optimal.

Thus, there are no unsuccessful packet transmissions in the network when using the gathering protocol. Since the paths to the sink node s in the gathering tree are only 5 times longer than the shortest paths, any packet is forwarded along a path of length that is a constant factor away from the shortest paths to s.

Hence, it holds that the gathering protocol is work-optimal, up to constant factors, to send all the packets to the sink node s. □

8.7.3 Self-stabilization

The algorithms presented for both the stages can be made self-stabilizing as follows. For stage 1, we require that nodes $v \in U$ keep sending the current value of $d'(s, v)$ as ROUTE messages during the time slot owned by v. When a node w just wakes up, it first sets $d'(s, w) = -\infty$ and $\pi(w) =$ NULL and listens to the channel for a period of cd_1 time steps for a constant $c \geq 3$. During this period, since other active nodes send their current distance estimate, node w can update its distance estimate and parent pointer accordingly. If additionally $w \in U \cup \mathcal{G}$ then w can participate in sending ROUTE messages after the initial listening period. This would also imply that also stage 2 can recover from any initial configuration if every node that has a packet and continues to retain the packet in case the predecessor pointer is undefined, (NULL). Ignoring those packets in transit that were lost by nodes that have moved or been powered off, all the other packets can indeed reach the sink node once the system recovers till stage 1.

8.8 Chapter Summary and Acknowledgements

Using our model presented in Chapter 7, we designed algorithms for broadcasting and gathering in wireless ad hoc networks. All our algorithms are efficient and self-stabilizing which is an important property for distributed systems. The algorithms are simple enough so that they not only look appealing in theory but may also work well in practice. Further, our algorithms only require a constant amount of storage at any node which make the algorithms highly useful in practical situations.

A preliminary version of the results in this chapter appeared in [76]. This work is done jointly with Melih Onus and Andrea Richa from the Department of Computer Science at the Arizona State University.

Chapter 9

Conclusions

In this thesis we argued that formal study of overlay networks is important to understand their various aspects and properties. We have looked specifically at two different classes of overlay networks namely, peer-to-peer networks and wireless ad hoc networks. Our focus was in two directions: how to arrive at good topologies and how to design efficient routing strategies.

The results presented in this thesis improve the state-of-the-art in many cases. For example, our construction from Chapter 5 is the first known construction to handle heterogeneity in a fair generality while still being efficient. The time and work required for join/leave operations also match those of most known randomized constructions. The results of Chapter 6 present a unified scheme to arrive at P2P topologies that can guarantee good properties.

In Part III we started by providing a new model for wireless communication which takes into account many of the limitations of existing models that are used in the theoretical community. Using this model, we presented self-stabilizing protocols for arriving at a sparse backbone. The backbone is then used to support higher order communication primitives.

While in this thesis we have studied aspects of topology and routing in overlay networks,

the taxonomy of algorithms for overlay networks is much broader. In the context of peer-to-peer networks, areas such as security issues, trust and reputation among peers, and accounting models are gaining a lot of attention in recent years.

Building overlay networks that are robust against malicious behavior is still an open problem. The first such construction emerged recently [132] but the results hold only for maintaining a cycle network. Similarly, peers require mechanisms so that they can trust the decisions of other peers even when operating under the influence of malicious peers with disruptive tendencies. This is essential so that peers can successfully delegate tasks or cooperate with other peers. File sharing systems need some accounting mechanisms so as to minimize free-riding where a set of users behave in a non-cooperative way and try to only benefit from others' resources while not contributing to the overall (social) benefit of the system. The decentralized nature of operation of overlay networks certainly makes these problems much more difficult.

Thus, the area of overlay networks still has important problems that arise in many practical settings. It it thus important to study analytical guarantees apart form suggesting heuristics. For this, it is required that such problems be formulated clearly, and justifiable assumptions be made to analyze solutions.

Finally, while there exist several solution methodologies that researchers have found useful, as the field matures itself it is worth-while to focus on developing general tools and techniques that could prove useful in multiple settings. We anticipate that in the coming years, such generalization would yield fruitful results. These take the flavor of witness tree mechanisms [107, 130], network flow based parameters such as the routing number [130] and the flow number [74], which have found wide applicability apart from their suggested application.

Bibliography

[1] The Archipelago project. Available at www.cnds.jhu.edu/rsearch/networks/archipelago/.

[2] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[3] M. Adler, E. Halperin, R. Karp, and V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer systems. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, 2003.

[4] M. Adler and C. Scheideler. Efficient communication strategies for ad hoc wireless networks. *Theory of Computing Systems*, 33:337–391, 2000.

[5] K. Alzoubi, P.-J. Wan, and O. Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *Proc. of the IEEE Hawaii International Conference on System Sciences (HICSS)*, 2002.

[6] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[7] B. Awerbuch, D. Holmer, and H. Rubens. The pulse protocol: Energy efficient infrastructure access, 2004.

[8] B. Awerbuch and C. Scheideler. The Hyperring: A low-congestion deterministic data structure for distributed environments. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, 2004.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. Technical Report UMIACS-TR 2002-53 and CS-TR 4373, University of Maryland, 2002.

[10] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):102–113, 2003.

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

[12] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Can we elect if we cannot compare? In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 324–332, 2003.

[13] M. Bender, M. Farach-Colton, S. He, B. Kuszmaul, and C. Leiserson. Adversarial contention resolution for simple channels. In *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–332, 2005.

[14] A. Bhargava, K. Kothapalli, C. Riley, M. Thober, and C. Scheideler. Pagoda: An overlay network for data management, routing and multicasting. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 170–179, 2004.

[15] P. Billingsley. *Probability and Measure*. Wiley Inter–Science, 3 edition, 1995.

[16] P. Bose, J. Gudmundsson, and M. Smid. Constructing plane spanners of bounded degree and low weight. In *Proc. of the European Symposium on Algorithms (ESA)*, pages 234–246, 2002.

[17] P. Bose and P. Morin. Online routing in triangulations. In *Proc. of the International Symposium on Algorithms and Computation (ISSAC)*, pages 113–122, 1999.

[18] P. Bose, P. Morin, A. Brodnik, S. Carlsson, E. Demaine, R. Fleischer, J. Munro, and A. Lopez-Ortiz. Online routing in convex subdivisions. In *Proc. of the International Symposium on Algorithms and Computation (ISSAC)*, pages 47–59, 2000.

[19] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *ACM/Kluwer Wireless Networks*, 7(6):609–616, 2001.

[20] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20:1489–1499, 2002.

[21] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of ACM SIGMETRICS 2003*, 2003.

[22] M. Castro, M.B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlay network. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, 2003.

[23] X. Cheng, D. Huang, W. Li, W. Wu, and D. Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks: An International Journal*, 42, 2003.

[24] I. Chlamtac and O. Weinstein. The wave expansion approach to broadcasting in multihop radio networks. *IEEE Transactions on Communications*, 39(3):426–433, 1991.

[25] S. Choi. IEEE 802.11e MAC-level FEC performance evaluation and enhancement. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, 2002.

[26] M. Choy and A. K. Singh. Efficient fault tolerant algorithms for resource allocation in distributed systems. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 169–177, 1992.

[27] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.

[28] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 206–219, 1986.

[29] Internet Systems Consortium. http://www.isc.org.

[30] A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 492–501, 2003.

[31] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report Technical Report 2001-31, Stanford University, 2001.

[32] E. W. Dijkstra. Self stabilization in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[33] Distributed.net. Available at http://www.distributed.net/.

[34] J. R. Douceur. The sybil attack. In *Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[35] E. J. Duarte-Melo and M. Liu. Data-gathering wireless sensor networks: organization and capacity. *Computer Networks*, 43(4):519–537, 2003.

[36] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 717–724, 2003.

[37] D. Dubhashi and A. Panconesi. Near-optimal distributed edge coloring. In *Proc. of the European Symposium on Algorithms (ESA)*, pages 448–459, 1995.

[38] M. Enachescu, A. Goel, R. Govindan, and R. Motwani. Scale free aggregation in sensor networks. In *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 71–84, 2004.

[39] D. Jefferson et al. Distributed simulation and the time warp operating system. In *Proc. of the 11th ACM Symposium on Operating Systems Principles (SOSP)*, 1987.

[40] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45, 1998.

[41] U. Feige and J. Kilian. Zero-knowledge and chromatic number. In *Proc. of the Annual Conference on Computational Complexity*, 1996.

[42] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[43] I. Finocchi, A. Panconesi, and R. Silvestri. Experimental analysis of simple, distributed vertex coloring algorithms. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, pages 606–615, 2002.

[44] P. Flocchini, B. Mans, and N. Santoro. On the impact of sense of direction on message complexity. *Information Processing Letters*, 63(1):23–31, 1997.

[45] Folding@home. Available at http://folding.stanford.edu/.

[46] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[47] J. Gao, L.J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proc. of the the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 45–55, 2001.

[48] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[49] C. GauthierDickey, D. Zappala, and V. Lo. A fully distributed architecture for massively multiplayer online games. In *ACM SIGCOMM Workshop on Network and System Support for Games*, 2004.

[50] Gnutella. Available at http://www.gnutella.com/.

[51] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry breaking in sparse graphs. *SIAM J. Discrete Math.*, 1:434–446, 1989.

[52] D. A. Grable. A large deviation inequality for functions of independent, multi-way choices. *Comb. Probab. Comput.*, 7(1), 1998.

[53] D. A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in $O(\log \log n)$ rounds. *Random Structures and Algorithms*, 10(3):385–405, 1997.

[54] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, pages 314–329, 2003.

[55] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, 2000.

[56] P. Gupta and P.R. Kumar. Internets in the sky: The capacity of three dimensional wireless networks. *Communications in Information and Systems*, 1(1):33–50, 2001.

[57] P. Gupta and P.R. Kumar. Towards an information theory of large networks: An achievable rate region. In *IEEE International Symposium on Information Theory (ISIT)*, 2001.

[58] Q. Han, S. Mehrotra, and N. Venkatasubramanian. Energy efficient data collection in distributed sensor environments. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, pages 590–597, 2004.

[59] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. of the USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[60] H. Huang, A. W. Richa, and M. Segal. Approximation algorithms for the mobile piercing set

problem with applications to clustering in ad-hoc networks. *ACM Baltzer Journal on Mobile Networks and Applications (MONET)*, pages 141–149, 2004.

[61] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of the 6th ACM/IEEE Mobicom Conference*, pages 56–67, 2000.

[62] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 197–212, 2000.

[63] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2001.

[64] O. Johansson. Simpled distributed $\Delta + 1$ coloring of graphs. *Information Processing Letters*, 70:229–232, 1999.

[65] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[66] T. Jurdzinski and G. Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proc. of the International Symposium on Algorithms and Computation (ISSAC)*, pages 535–549, 2002.

[67] M. F. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[68] M. Karchmer and J. Naor. A fast parallel algorithm to color a graph with Δ colors. *J. Algorithms*, 9:83–91, 1988.

[69] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proc. of the ACM symposium on Theory of computing (STOC)*, pages 654–663, 1997.

[70] R. Karp and A. Widgerson. A fast parallel algorithm for the maximal independent set problem. *Journal of the Association for Computing Machinery*, 32(4):762–773, 1985.

[71] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1975.

[72] Kazaa. Available at http://www.kazaa.com/, 2003.

[73] J. Keil and W. Gutwin. The Delaunay triangulation closely approximates the complete euclidean graph. In *1st Workshop on Algorithms Data Structure (LNCS382)*, 1989.

[74] P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, pages 184–193, 2002.

[75] K. Kothapalli, M. Onus, A. Richa, and C. Scheideler. Constant density spanners for wireless ad-hoc networks. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 116–125, 2005.

[76] K. Kothapalli, M. Onus, A. Richa, and C. Scheideler. Efficient broadcasting and gathering

in wireless ad hoc networks. In *Proc. of the IEEE International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 346–351, 2005.

[77] K. Kothapalli, M. Onus, C. Scheideler, and C. Schindelhauer. Distributed coloring in $\tilde{O}(\sqrt{\log n})$–bits. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[78] K. Kothapalli and C. Scheideler. Information gathering in adversarial systems: Lines and cycles. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 333–342, 2003.

[79] K. Kothapalli and C. Scheideler. Supervised peer-to-peer systems. In *Proc. of the IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, pages 188–193, 2005.

[80] D. Kowalski and A. Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 63–72, 2002.

[81] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 73–82, 2003.

[82] D. Kowalski and A. Pelc. Faster deterministic broadcasting in ad hoc radio networks. In *Proc. of the Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 109–120, 2003.

[83] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Radio network clustering from scratch. In *Proc. of the European Symposium on Algorithms (ESA)*, 2004.

[84] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 25–32, 2003.

[85] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2003.

[86] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobil ad-hoc routing. In *Proc. of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, 2002.

[87] F. Kuhn and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proc. of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 69–78, 2003.

[88] V. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. End-to-end packet-scheduling in wireless ad-hoc networks. In *Proc. of ACM Symposium on Discrete Algorithms (SODA)*, pages 1021–1030, 2004.

[89] J. Kuruvila, A. Nayak, and I. Stojmenovic. Hop count optimal position based packet routing algorithms for ad hoc wireless networks with a realistic physical layer. In *1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2004.

[90] E. Kushilevitz and Y. Mansour. An $\Omega(D \log(n/D))$ lower bound for broadcast in radio networks. *SIAM Journal on Computing*, 27(3):702–712, 1998.

[91] K. Lakshminarayanan, A. Rao, I. Stoica, , and S. Shenker. Flexible and robust large scale multicast using I3. Technical Report UCB Technical Report No. UCB/CSD-02-1187, University of California, Berkeley, 2002.

[92] D.H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Computers*, 24(12):1145–1155, 1975.

[93] X.-Y. Li, G. Calinescu, and P.-J. Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *Proc. of the Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM)*, pages 1268–1277, 2002.

[94] X.-Y. Li, P.-J. Wan, and Y. Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 564–567, 2001.

[95] X.-Y. Li, P.-J. Wan, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless netowrks. In *IEEE Hawaii International Conference on System Sciences (HICSS)*, 2002.

[96] Z. Li and P. Mohapatra. Qron: Qos-aware routing in overlay networks. *IEEE Journal On Selected Areas In Communications*, 22(1), 2004.

[97] B. Liang and Z. J. Haas. Virtual backbone generation and maintenance in ad hoc network mobility management. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, pages 1293–1302, 26–30 2000.

[98] N. Linial. Locality in distributed graph algorithms. *SIAM Journal of Computing*, 21:193–201, 1992.

[99] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems; routing distances and fault resilience. In *Proceedings of the ACM SIGCOMM*, 2003.

[100] L. Lovasz. On the ratio of the optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[101] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72– 93, 2005.

[102] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. of the 17th ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1985.

[103] M. Luby. Removing randomness in parallel without processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[104] H. Luo and G. Pottie. Routing explicit side information for data compression in wireless sensor networks. In *Proc. of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2005.

[105] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[106] G. De Marco and A. Pelc. Fast distributed graph coloring with $O(\Delta)$ colors. In *Proc. of ACM Symposium on Discrete Algorithms (SODA)*, pages 630–635, 2001.

[107] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two choices: A survery of techniques and results. pages 255–312. Kluwer Academic Publishers, 2001.

[108] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[109] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[110] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 50–59, 2003.

[111] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Lser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *Int. World Wide Web Conference*, 2003.

[112] S-Y. Ni, Y-C. Tseng, Y-S. Chen, and J-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *ACM Mobicom*, pages 151–162, 1999.

[113] Committee on the Internet In The Evolving Information Infrastructure. *The Internet's Coming of Age*. National Academy Press, 2001.

[114] A. Oram, editor. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly Media, Inc, 2001.

[115] IEEE P802.11a/D7.0-1999. Wireless LAN medium access control (MAC) and physical layer (PHY) specification: High speed physical layer in the 6 ghz band. IEEE, New York, 1999.

[116] A. Panconesi and A. Srinivasan. Fast randomized algorithms for distributed edge coloring. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 251–262, 1992.

[117] A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *J. Algorithms*, 20(2):356–374, 1996.

[118] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 492–499, 2001.

[119] S. Parthasarathy and R. Gandhi. Distributed algorithms for coloring and domination in wireless ad hoc networks. In *Proc. of the International Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2004.

[120] L. Peterson and B. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, 3 edition, 2003.

[121] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

[122] L. Quin and T. Kunz. On-demand routing in MANETs: The impact of a realistic physical layer model. In *Proc. of the 2nd International Conference ADHOC-NOW*, pages 37–48, 2003.

[123] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *Proceedings of ACM SIGCOMM*, pages 331–342, 2004.

[124] T.S. Rappaport. *Wireless Communications: Principle and Practice*. Prentice Hall, 1996.

[125] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, pages 161–172, 2001.

[126] S. Ratnasamy, M. Handley, R.M. Karp, and S. Shenker. Application-level multicast using

content-addressable networks. In *Proceedings of the International Workshop on Networked Group Communication (NGC)*, 2001.

[127] C. Riley and C. Scheideler. A distributed hash table for computational grids. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[128] C. Riley and C. Scheideler. Guaranteed broadcasting using SPON: Supervised P2P overlay network. In *International Zürich Seminar on Communications*, 2004.

[129] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[130] C. Scheideler. *Universal Routing Strategies for Interconnection Networks*, volume 1390 of *Lecture Notes in Computer Science*. Springer, 1998.

[131] C. Scheideler. *Probabilistic Methods for Coordination Problems*. HNI-Verlagsschriftenreihe 78, University of Paderborn, 2000.

[132] C. Scheideler. How to spread adversarial nodes? rotate! In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, 2005.

[133] C. Schindelhauer and G. Schomaker. Weighted distributed hash tables. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2005.

[134] C. Scott, P. Wolfe, and M. Erwin. *Virtual Private Networks.* O'Reilly Media, Inc, 1999.

[135] SETI@home. Available at http://setiathome.ssl.berkeley.edu/.

[136] G. Singh. Efficient leader election using sense of direction. *Distributed Computing*, 10(3):159–165, 1997.

[137] P. Slavik. A tight analysis of the greedy algorithm for set cover. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 435–441, 1996.

[138] Wen-Zhan Song, Yu Wang, Xiang-Yang Li, and O. Frieder. Localized algorithms for energy efficient topology in wireless ad hoc networks. In *Proc. of the ACM International Symposium on Mobile ad hoc networking and computing (MOBIHOC)*, pages 98–108, 2004.

[139] M. Srivatsa, B. Gedik, and L. Liu. Scaling unstructured peer-to-peer networks with multi-tier capacity-aware overlay topologies. See http://www.cc.gatech.edu/∼mudhakar/.

[140] R. Steinmetz and K. Wehrle, editors. *Peer-to-peer systems and applications*. Lecture Notes in Computer Science, Vol. 3485. Springer, 2005.

[141] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. of the International Peer-to-Peer Systems (IPTPS)*, pages 191–202, 2002.

[142] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[143] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, , and B. Yeager. Project JXTA 2.0 Super-Peer virtual network sun microsystems, 2003.

[144] P. J. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, 2002.

[145] C. Wang and B. Li. Peer-to-peer overlay networks: A survey. Technical report, Department of Computer Science, Hong Kong University of Science and Technology, 2003.

[146] Y. Wang and X.-Y. Li. Geometric spanners for wireless ad hoc networks. In *Proc. of the IEEE International Conference Distributed Computing Systems (ICDCS)*, 2002.

[147] Y. Wang and X.-Y. Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 59–68, 2003.

[148] R. Wattenhofer, J. Li, P. Bahl, and Y.-M. Wang. Distributed topology control for wireless multihop ad-hoc networks. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, pages 1388–1397, 2001.

[149] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, 2002.

[150] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14, 1999.

[151] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of the IEEE International Conference on Data Engineering (ICDE)*, 2003.

[152] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[153] Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, 2004.

[154] R. Zhang and Y. C. Hu. Borg: A hybrid protocol for scalable application level multicast in peer-to-peer networks. In *Proc. of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM)*, 2003.

[155] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[156] Y. Zhu, H. Wang, and Y. Hu. A super-peer based lookup in highly structured peer-to-peer networks. In *Proc. of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, 2003.

[157] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz. Bayeux: An architecture for scalable and fault tolerant wide-area data dissemination. In *Proc. of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2001.

Vita

Kishore Kothapalli was born in Rajahmundry, India in 1976. He completed his schooling in Kakinada and then obatined his Bachelor degree Computer Science from the Regional Engineering College, Warangal in 1996. From 1996 to 1998 he was at the Indian Institute of Technology, Kanpur studying for a Master's degree in Computer Science. He worked in the software industry from 1998-2001 in Hyderabad, India and New York, USA. In 2001, he moved to the Johns Hopkins University to pursue his Ph. D. in Computer Science.