Distributed Algorithms for Dynamic Topology Construction and Their Applications

by

Ching Law

M.Eng., Electrical Engineering and Computer Science
MIT, 1999
S.B., Computer Science and Engineering
MIT, 1998
S.B., Mathematics
MIT, 1998

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author .....
Department of Electrical Engineering and Computer Science
August 31, 2004

Certified by .....
Kai-Yeung Siu
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Distributed Algorithms for Dynamic Topology Construction and Their Applications

by

Ching Law

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We introduce new distributed algorithms that dynamically construct network topologies. These algorithms not only adapt to dynamic topologies where nodes join and leave, but also actively set up and remove links between the nodes, to achieve certain global graph properties.

First, we present a novel distributed algorithm for constructing overlay networks that are composed of d Hamilton cycles. The protocol is decentralized as no globally-known server is required. With high probability, the constructed topologies are expanders with $O(\log_d n)$ diameters and $2\sqrt{2d-1}+\epsilon$ second largest eigenvalues. Our protocol exploits the properties of random walks on expanders. A new node can join the network in $O(\log_d n)$ time with $O(d \log_d n)$ messages. A node can leave in $O(1)$ time with $O(d)$ messages.

Second, we investigate a layered construction of the random expander networks that can implement a distributed hash table. Layered expanders can achieve degree-optimal routing at $O(\log n / \log\log n)$ time, where each node has $O(\log n)$ neighbors. We also analyze a self-balancing scheme for the layered networks.

Third, we study the resource discovery problem, in which a network of machines discover one another by making network connections. We present two randomized algorithms to solve the resource discovery problem in $O(\log n)$ time.

Fourth, we apply the insight gained from the resource discovery algorithms on general networks to ad hoc wireless networks. A Bluetooth ad hoc network can be formed by interconnecting piconets into scatternets. We present and analyze a new randomized distributed protocol for Bluetooth scatternet formation. We prove that our protocol achieves $O(\log n)$ time complexity and $O(n)$ message complexity. In the scatternets formed by our protocol, the number of piconets is close to optimal, and any device is a member of at most two piconets.

Thesis Supervisor: Kai-Yeung Siu
Title: Associate Professor

Acknowledgments

First, I must thank my adviser Professor Kai-Yeung (Sunny) Siu for encouraging and supporting me throughout my graduate school career. Sunny has provided me enviable freedom in choosing my research directions.

A substantial portion of this thesis appeared in papers I coauthored with Sunny. Chapter 5 is based on papers I coauthored with Amar Mehta and Sunny.

I should also thank my thesis committee members Professor Piotr Indyk and Professor David Karger, for reading my thesis and offering invaluable suggestions to improve it.

I thank Professor Charles Leiserson. It was Charles' 6.046 take-home exam that gave me the first glimpse into computer science research. Charles accepted me as an undergraduate researcher, supervised my Master's thesis, and continued to guide me whenever I went to him for advice.

I would like to thank Professor Sanjay Sarma for the generous research assistantship support I received from the Auto-ID Center. I thank Professor Albert Meyer and Dr. Eric Lehman for taking me as his teaching assistant for 6.042. I am also grateful to my supervisor Dr. Ashwini Nanda at IBM Watson Research Center for two rewarding summer internships.

Kayi Lee has been my best companion in our studies in Computer Science at MIT. Discussing algorithms with him has always been enjoyable. He has given me much help with this thesis.

I have benefited from the knowledge and wisdom of other members in Sunny's research group. In particular, I would like to thank Anthony Kam and Paolo Narvaez.

I also want to thank my friends who have read my thesis or offered suggestions for its defense presentation—Henry Lam, Kayi Lee, Philip Lee, Alan Leung, Ting Li, and Joyce Ng.

I thank my sister Cindy for listening to my ramblings. Two critical ideas in this thesis originated from our discussions during our trips in China. Last but not least, I want to thank my parents for their support and patience through all these years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Distributed algorithms can construct and alter network topologies efficiently. When network size increases, centralized topology control becomes difficult. This thesis introduces new distributed algorithms that change the connectivity of a topology and distributed algorithms that construct highly scalable topologies. To assist the topology construction algorithms, we also present several distributed algorithms that discover and collect information on these changing topologies. Applications of our algorithms include resource discovery, distributed hash tables, dynamic on-line communities, and topology construction for wireless ad hoc networks.

We will give an overview of the thesis in the rest of this Chapter. Section 1.1 motivates the construction of highly scalable expander networks. We preview the key theoretical results in Section 1.2 and applications in Section 1.3. Section 1.4 describes the structure of the thesis.

1.1 Motivations

1.1.1 Special Interest Networks

In this section we describe the main motivations behind our work on dynamic distributed algorithms.

Most existing peer-to-peer systems focus on distributed sharing of resources. They usually facilitate sharing of storage (Napster, Gnutella, Freenet) or computation (SETI@Home). Many systems implement distributed hash tables.

Many believe that resource sharing is the major benefit of peer-to-peer networks. To facilitate resource sharing, a system usually needs to support *resource searching*. However, designing a scalable search system has always been very difficult.

Resource searching without any centralized directory usually suffers from poor performance when the number of participants grows. Distributed networks such as Gnutella

simply search by flooding the network and cannot provide any performance guarantee. A caching scheme such as the one employed by Freenet can replicate popular objects. However, less popular objects remain hard to be found.

A search system using a centralized directory requires a high-end machine to support a large network. Central directories are used by many established systems, such as ICQ, Napster, and SETI@Home. However, machines currently affordable by most individuals are unlikely to serve more than hundred thousand users. Although computers are getting cheaper and more powerful, the sizes of peer-to-peer networks and the amount of resource shared are also growing rapidly. Thus it is not clear whether a centralized server can serve a global peer-to-peer network in the near future.

We suggest that participants in peer-to-peer networks exhibit 'interest locality'. The resources a particular participant is interested in obtaining and sharing are correlated. For example, in a file-sharing network, the file that a participant will download in the future is likely to be related to the files that she downloaded in the past. Consider the following scenarios:

- In a music-sharing network, participants would have preferences in different genres. For example, some participants could be interested in classical music but not pop music, or vice versa. In a global network, participants would have different language preferences too.
- In a file-sharing network, participants would have very different interests as well. Some are interested in text documents only. Some are looking for multimedia files. Also, users are usually interested in applications on specific platforms.
- In a general peer-to-peer network, participants usually have different goals. Some are interested in file-sharing, some are looking for fellow on-line gamers, some are looking for special interest discussion groups.
- In a heterogeneous environment, participants have different computing resources. Computing devices can range from mobile phones to enterprise servers. Certain applications have delay constraints (action games), while others have bandwidth requirements (large file transfers).

In summary, it would be beneficial to peer with nodes of similar interests, instead of some arbitrary nodes.

We believe that a huge and all-purpose resource sharing network might not be necessary. Instead, we propose that different protocols can be used for "sharing" and "searching". We need large networks for searching so that we can potentially reach a large repository of resources. However, smaller networks are more efficient for most sharing activities. In other words, we mostly associate with smaller groups, we would like to be well-informed and have good 'connections' so that we can easily find and join other interesting groups. In order

to reach such small groups easily, we would like to propose a protocol to connect as many nodes as possible, so that any such group can be formed and discovered efficiently.

Small groups of peer-to-peer networks can be tied by common interests. However, users need the means to search and reach the interest groups. We will present distributed algorithms that create and maintain special interest groups. Any interest group can be searched and reached with performance proportional to its size. We call this the *Subset-Search* problem because we want to search for a subset in a network.

In this thesis, we propose a collection of algorithms to solve the problem we have described. First, we need to scalable network to serve as a large search platform (Section 1.2.1). We need data collection algorithms to run on such dynamic network (Section 1.2.4). Then we need a hierarchical system to optimize for searches (Section 1.2.2). We also need an effective algorithm for special interest networks to be established (Section 1.2.3). We did an special study for this problem on wireless ad hoc networks (Section 1.3.3).

1.1.2 Distinctions

Subset-Search is different from group communications. Many distributed systems face scalability problems, especially if they support features such as anonymity, data security, and resistance to various attacks. Many features require the nodes to have a consistent view of the entire network, thus making any protocol difficult to scale to a large number of nodes. However, many applications do not need to satisfy these strong guarantees. For example, some video streaming applications can tolerate packet losses. In general, we can usually gain scalability by reducing requirements. We will design a protocol just strong enough to support Subset-Search.

Subset-Search is different from distributed hash tables. It is because a user typically does not have hashes of the objects that are interested to him or her. As an analogy, a web search engine user usually searches by keywords and other attributes, but will not know the hash or key of the relevant web pages.

Subset-Search is different from search engines. Machines in peer-to-peer networks are much more dynamic than most web servers.

1.2 Theoretical Results

We highlight the main theoretical contributions of this thesis.

1.2.1 Random Expander Networks

We present a novel distributed protocol for constructing an overlay topology based on random regular graphs that are composed of $d \geq 4$ independent Hamilton cycles. The protocol

is completely decentralized as no globally-known server is required. The constructed topologies are expanders with $O(\log_d n)$ diameter with high probability.

Our construction is highly scalable because both the processing and the space requirements at each node grow logarithmically with the network size. A new node can join the topology at any existing node in $O(\log_d n)$ time with $O(d \log_d n)$ messages. A node can leave in $O(1)$ time with $O(d)$ messages.

The analysis of our construction is based on a novel technique of studying a sequence of probability spaces. We also applied many results from random walks and random graphs.

1.2.2 Layered Expander Networks

$\mathbb{H}$-graph is efficient for locating a member of a subset, when the size of the subset is not too small when compared with the size of the entire graph. However searching for a very small subset could be slow. For example, the expected time to find any particular node is $\Theta(n)$.

We show that layered random regular graphs can be constructed such that each node has $O(\log n)$ neighbors and any node can be located in $O(\log n)$ steps given its identifier. We also present a variant that achieves degree-optimal routing at $O(\log n / \log \log n)$ steps. We analyze the scheme where identifiers are assigned randomly and the scheme where identifiers are obtained by sampling in the existing graph. These recursively overlaid $\mathbb{H}$-graphs are called layered $\mathbb{H}$-graphs.

1.2.3 Resource Discovery and Connectivity

The problem of a network of computers discovering one another by making network connections is called the *resource discovery* problem. This problem was proposed by Harchol-Balter, Leighton, and Lewin [45]. The algorithmic task is to transform a weakly-connected graph into a complete graph by a distributed algorithm.

We present randomized algorithms to solve the resource discovery problem with $O(\log n)$ time complexity and $O(n^2)$ expected message complexity. There are several variations of the algorithm with different tradeoffs in time and message complexities.

The problem of transforming a weakly-connected graph into a strongly-connected graph is also very interesting, and in fact might not be easier than the resource discvoery problem. We investigate several approaches to address this problem.

1.2.4 Algorithms on Dynamic Network Topology

We also study several interesting distributed algorithms on dynamic network topology. These algorithms have applications in our constructions in of $\mathbb{H}$-graphs, but they are also of independent interests themselves. Most of the following algorithms depend on the expansion of the network graph to achieve good performance.

Broadcasts and Searches

We present and analyze broadcast and search algorithms on random regular graphs that are composed of Hamilton cycles.

An important application of these algorithms is a service for the discovery of communities sharing common interests. Let A be a subset of nodes of a degree-d $\mathbb{H}$-graph G such that $|A| / |G| = \psi$. For any such set A, we can find a member of A in $O(\log \psi^{-1} + \log \log n)$ time with $O(\psi^{-1} \log n)$ messages with high probability. Moreover, we can find a member of A in $O(\psi^{-1} + \log n)$ hops in expectation even if set A is selected by an adversary.

Network Size Estimation

In a distributed network, any node does not easily know the size of the network. However, our construction algorithms would run more efficiently if we have a good estimation of the network size. Therefore, we would like to have an efficient distributed algorithm to gather and disseminate the size estimates.

Estimating the size of a network graph distributedly is an interesting algorithmic problem. We propose a size-estimation algorithm based on random walks. With $O(\log n)$ walkers, we can obtain an $\Omega(n)$ estimate at all nodes in $O(n)$ time, with high probability.

1.3 Applications

We overview several applications of our distributed dynamic algorithms.

1.3.1 Distributed Lookup Service

A layered $\mathbb{H}$-graph can be used to implement a distributed hash table similar to those supported by Chord [92], CAN [83], Tapestry [99], and Pastry [85]. A lookup service stores key-value pairs in the network, such that the values can be looked up efficiently with the key. To implement a lookup service on the layered $\mathbb{H}$-graph, keys can be hashed into identifiers. A key is stored at the node whose identifier has the longest prefix match with the hashed identifier. Insertions, deletions, and updates of entries have $O(\log n)$ time complexity and $O(\log n)$ message complexity. With optimizations, the time complexity can be improved to $O(\log n / \log \log n)$.

1.3.2 On-line Communities

A scenario where layered $\mathbb{H}$-graph would be useful is a networked community where the network address of a node may change from time to time. A layered $\mathbb{H}$-graph construction can allow us to form an on-line community without any centralized directory. A node connected to the Internet by DHCP can have its IP address changed everyday. On a laptop

or PDA capable of local wireless access, the IP address can change even more frequently. A node changing its network address only needs to notify $O(\log n)$ other nodes in $O(1)$ time. Any node can reach another node in $O(\log n)$ hops given its identifier.

For example, a layered $\mathbb{H}$-graph can be used to implement an on-line instant messaging system without any centralized server. Consider a student Ben chatting on-line during classes. If the on-line messaging system is implemented on a layered $\mathbb{H}$-graph, then he only needs to notify $O(\log n)$ neighbor nodes when he moves to another classroom. When his friend Alice wants to obtain his new IP address after lunch break, she only needs to walk through $O(\log n)$ hops. Such feature is not easy implement on distributed systems because if Alice and Ben have overlapping lunch breaks, they would have difficulty finding each others' IP addresses in the afternoon.

1.3.3 Bluetooth Topology Construction

A Bluetooth ad hoc network can be formed by interconnecting piconets into scatternets. The constraints and properties of Bluetooth scatternets present special challenges in forming an ad hoc network efficiently. In Chapter 5, we present and analyze a new randomized protocol for Bluetooth scatternet formation.

We prove that our algorithm achieves $O(\log n)$ time complexity and $O(n)$ message complexity. The scatternets formed have the following properties: 1) any device is a member of at most two piconets, and 2) the number of piconets is close to be optimal. These properties can help prevent overloading of any single device and lead to low interference between piconets.

We validate these theoretical results by simulations. In addition, the simulations show that the scatternets formed have $O(\log n)$ diameter.

As an essential part of the scatternet formation protocol, we study the problem of device discovery: establishing multiple connections simultaneously with many Bluetooth devices. We investigate the collision rate and time requirement of the inquiry and page processes.

Deducing from the simulation results of scatternet formation and device discovery, we show that the total number of packets sent is $O(n)$ and that the maximum number of packets sent by any single device is $O(\log n)$.

1.4 Structure of Thesis

In Chapter 2, we present and analyze the construction of the random expander networks. In Chapter 3, we construct layered networks of random expanders to implement a fast distributed lookup service. Chapter 4 describes distributed algorithms for resource discovery. Chapter 5 presents an application of the topology construction methodology in wireless ad hoc networks. We conclude and discuss future work in Chapter 6.

A large portion of the results in this thesis first appeared in [65, 63, 62].

Chapter 2

Construction of Expander Networks

2.1 Introduction

The area of peer-to-peer networking has recently gained much attention in both the industry and the research community. Well-known peer-to-peer networks include Napster, Gnutella, Freenet, FastTrack, and eDonkey. However, most of these systems either require a centralized directory or cannot scale beyond a moderate number of nodes. The problem of finding an efficient distributed scalable solution has attracted a lot of research interests [83, 92, 85, 99]. This chapter introduces a distributed algorithm for constructing expander networks, which are suitable for peer-to-peer networking, without using any globally-known server.

Interesting properties and algorithms have been discovered for random regular graphs [34, 36, 37, 58]. In particular, it has been found that random regular graphs are expected to have big eigenvalue gaps [32] with high probability, and thus are good expanders.

In this chapter, we form expander graphs by constructing a class of regular graphs which we call $\mathbb{H}$-graphs. An $\mathbb{H}$-graph is a $2d$-regular multigraph in which the set of edges is composed of d Hamilton cycles (Figure 2.1 is an example). Using random walk as a sampling algorithm, a node can join an $\mathbb{H}$-graph in $O(\log_d n)$ time with $O(d \log_d n)$ messages, and leave in $O(1)$ time with $O(d)$ messages.

Section 2.2 describes our network model and gives an overview of the design of $\mathbb{H}$-graphs. Section 2.3 introduces the protocol for constructing $\mathbb{H}$-graphs. We discuss several perfect sampling algorithms in Section 2.4 and a random-walk sampling algorithm in Section 2.5, with simulation results in Section 2.6. Two useful maintenance algorithms are discussed in Section 2.7. We discuss broadcasts in Section 2.8 and searches in Section 2.9. Then we study distributed network size estimation in Section 2.10. At last, fault tolerance issues

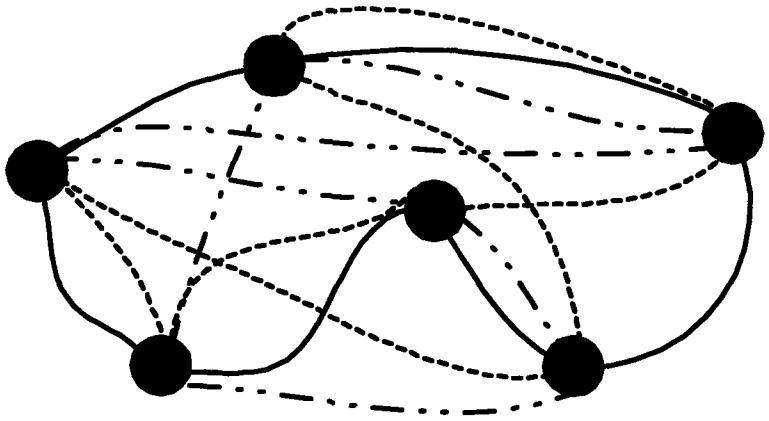

Figure 2.1: An $\mathbb{H}$-graph consisting of 3 Hamilton cycles.

are discussed in Section 2.11. We discuss related work in Section 2.12 and conclude with remarks on future work in Section 2.13.

2.2 Preliminaries

In this section, we will state our assumptions of the underlying network model and then describe the goals and constraints that lead to our design based on random regular graphs.

2.2.1 Network Model

We assume a network environment where any node u can send a message to node v as long as node u knows the address of v. If a node fails to receive a message for whatever reason, the sending node can repeat sending the message without causing the algorithm to fail. There can be node failures but not permanent link failures. Such model is assumed in [45] and most peer-to-peer research. An example of such an underlying network is the Internet when using some reliable messaging protocol such as TCP.

The space requirement will be expressed in the number of addresses. Theoretically, $O(\log N)$ bits are required to encode the address of a node, where N is the largest possible number of nodes. However, for all practical purposes, we can assume that the length of an address is effectively constant (e.g., 128 bits in IPv6).

We assume that there is a maximum delay between all pairs of nodes in the underlying network. In practice, the processing time per message is usually insignificant compared to

the communication time. Also, for a small message, the delivery time is mostly independent of the message size.

Each execution of a distributed algorithm will lead to a set of messages sent among the nodes in the graph. The message complexity of an algorithm is the size of this set. Some of these messages have causal relationships: there is some sequence $m_1, m_2, \ldots$ of messages where m_i cannot be sent before m_{i-1} has been received. We will express the time complexity of an algorithm as the length of the longest such sequence.

We will be concerned with the logical topologies overlaid on top of the underlying network. On a set of nodes with labels $[n] = \{1, 2, \ldots, n\}$, a topology can be effectively determined by sets of *neighbors* $N(u)$, which are nodes known to node u (not including u itself), for $u \in [n]$. In the rest of this paper, we represent such logical topology as a graph $G = (V, E)$, where $V = [n]$ and

$$(u, v) \in E \text{ iff } v \in N(u).$$

We consider distributed algorithms on the graph such that u can send a message to v only if $(u, v) \in E$. During the execution of our algorithm, an edge (u, v) can be added into E if u is informed of the address of v. Node u might also choose to delete (u, v) for some v to save space.

2.2.2 Goals and Requirements

Our first goal is to construct logical topologies that can support broadcasting and searching efficiently. Our second goal is to make our construction highly scalable. This implies the objectives:

A-1 Resources consumed at each individual node should be as low as possible.

A-2 Time complexities for joining and leaving of nodes should be as low as possible.

A key factor on the load of a node is the number of nodes that it has to communicate with. This parameter determines a node's minimum storage requirement and the maximum number of potential simultaneous network connections. The number of neighbors of a node is its degree in the graph. Therefore, objective A-1 dictates that the degrees should be as small as possible. In order to achieve objective A-2, we need to make sure that the algorithms for nodes joining and leaving are efficient.

2.2.3 A Random Graph Approach

To make worst-case scenarios unlikely, we have decided to construct the graph with a randomized protocol. We would like our topology to be 'symmetric' in the sense that every node shares an equal amount of responsibilities. For this reason and for providing better

fault tolerance, we did not choose a hierarchical approach. After considering various random graph models, we chose regular multi-graphs that are composed of independent Hamilton cycles. Regular graphs are chosen because we would like the degree to be bounded. Also, random walking on regular graphs, which is a key part of our protocol, has a uniform stationary distribution. Graphs composed of Hamilton cycles have the advantage that the join and leave operations only require local changes in the graph.

Now we shall define the set of $\mathbb{H}$-graphs. Let $\mathbb{H}_n$ denote the set of all Hamilton cycles on set $[n]$. We shall assume $n \geq 3$ for rest of this paper. Consider a multigraph $G = (V, E)$, such that $V = [n]$ and $E = (C_1, \ldots, C_d)$, where $C_1, C_2, \ldots, C_d \in \mathbb{H}_n$. Let $\mathbb{H}_{n,2d}$ be the set of all such $2d$-regular multigraphs. We call the elements in $\mathbb{H}_{n,2d}$ the $\mathbb{H}$-graphs. It can be derived that $|\mathbb{H}_n| = (n-1)!/2$ and $|\mathbb{H}_{n,2d}| = ((n-1)!/2)^d$. If $C_1, C_2, \ldots, C_d$ are independent uniform samples of $\mathbb{H}_n$, then $(V, (C_1, \ldots, C_d))$ is a uniform sample of $\mathbb{H}_{n,2d}$.

Following the notation in Bollobás [16], a probability space is a triple (Ω, Σ, P), where Ω is a finite set, Σ is the set of all subsets of Ω, and P is a measure on Σ such that $P(\Omega) = 1$ and $P(A) = \sum_{w \in A} P(\{w\})$ for any $A \in \Sigma$. In other words, P is determined by the values of $P(\{w\})$ for $w \in \Omega$. For simplicity, we will write $P(w)$ for $P(\{w\})$.

Let U_Ω be the uniform measure on set Ω so that $U_\Omega\{w\} = 1/|\Omega|$ for all $w \in \Omega$. For example, we have $U_{\mathbb{H}_{n,2d}}\{G\} = (2/(n-1)!)^d$ for all $G \in \mathbb{H}_{n,2d}$.

We shall consider two basic operations for a randomized topology construction protocol: JOIN and LEAVE. A JOIN(u) operation inserts node u into the graph. Any node in G should be able to accept a JOIN request at any time. Any node in G can also call LEAVE to remove itself from the graph G. Our algorithms of JOIN and LEAVE are described in Section 2.3. Given an initial probability space $\mathcal{S}_0$ and a sequence of JOIN and LEAVE requests, a randomized topology construction protocol will create a sequence of spaces $\mathcal{S}_1, \mathcal{S}_2, \ldots$.

Friedman [32, 33] showed that a graph chosen uniformly from $\mathbb{H}_{n,2d}$ is very unlikely to have a large second largest eigenvalue. In order to apply Friedman's theorem, we need a protocol that would produce a sequence of uniformly distributed spaces.

Given a probability space $\mathcal{S} = (\Omega', \Sigma', P')$, let $\Omega[\mathcal{S}] = \Omega'$ and $P[\mathcal{S}] = P'$. A probability space $\mathcal{S}$ is uniformly distributed if $P[\mathcal{S}] = U_{\Omega[\mathcal{S}]}$. We would like to have a protocol that creates a sequence of uniformly distributed probability spaces, given *any* sequence of JOIN and LEAVE requests. In addition, a new node should be free to call JOIN at any existing node in the graph.

Summarizing the objectives we have so far, we would like to have a protocol where

B-1 Low space complexity at any node.

B-2 Low time complexities for JOIN and LEAVE.

B-3 Low message complexities for JOIN and LEAVE.

B-4 The probability spaces produced are uniformly distributed.

Satisfying the first three properties is crucial because they are necessary for our protocol to be highly scalable. When property B-4 is not satisfied, we can try to construct sequences of probability spaces $\mathcal{S}_0, \mathcal{S}_1, \ldots$ having distributions close to be uniform. This can be achieved by an algorithm based on random walks presented in Section 2.5. We have yet to find a protocol satisfying all four properties simultaneously.

2.2.4 Global Variable Server

We will also consider variants where a globally-known server can simplify the algorithm or enhance the performance. We call such server a *global server,* whose address is known to all nodes in the graph. The global server should be reliable and highly available. We note that similar servers are assumed in most related work [77, 57] of randomized network constructions.

We will restrict our attention to those protocols that require the global servers to store $O(1)$ addresses. In other words, a global server implements $O(1)$ global variables that are visible to all nodes in the graph. The size of each variable is limited to $O(\log n)$ bits. We assume that atomic updates of the variables are ensured by the global server. A global server can be implemented as a cluster.

In summary, although the global server requires extra reliability, it does not need extraordinary storage, processing, or communication capacity.

2.3 Construction

In this section we introduce a framework for constructing a random regular network.

The graphs that we shall construct are $2d$-regular multigraphs in $\mathbb{H}_{n,2d}$, for $d \geq 4$. The neighbors of a node are labeled as

$$\mathsf{ngbr}_{-1}, \mathsf{ngbr}_1, \mathsf{ngbr}_{-2}, \ldots, \mathsf{ngbr}_{-d}, \mathsf{ngbr}_d.$$

For each i, ngbr_{-i} and ngbr_i denote a node's predecessor and successor on the ith Hamilton cycle (which will be referred to as the level-i cycle).

We start with 3 nodes, because there is only one possible $\mathbb{H}$-graph of size 3. In practice, we might want to use a different topology such as a complete graph when the graph is small.

The graph grows incrementally when new nodes call JOIN at existing nodes. Any node can leave the graph by calling LEAVE.

In the following algorithmic pseudocodes, the variable self identifies the node executing the procedure. All the actions are performed by self by default. The expression $u \Rightarrow$PROC() invokes a remote procedure call PROC at node u. The call is assumed to be non-blocking unless a return value is expected. Expression $u \Rightarrow$ var means that we request the value var

from node u. Expression $(u \Rightarrow \mathsf{var}) \leftarrow x$ means that we set the variable var of node u to value x. Thus, messages are exchanged between node self and node u.

Procedure LINK connects two nodes on the level-i cycle.

```
LINK(u, v, i)
1  (u ⇒ ngbr_i) ← v
2  (v ⇒ ngbr_{-i}) ← u
```

Procedure INSERT(u, i) makes node u the successor of self on the level-i cycle. We assume that INSERT is atomic. This can achieved by having a lock for each of the d Hamilton cycles at each node.

```
INSERT(u, i)
1  LINK(u, ngbr_i, i)
2  LINK(self, u, i)
```

A new node u joins by calling JOIN(u) at any existing node in the graph. Node u will be inserted into cycle i between node v_i and node $(v_i \Rightarrow \mathsf{ngbr}_i)$ for randomly chosen v_i's for $i = 1, \ldots, d$.

```
JOIN(u)
1  for i ← 1, ..., d in parallel
2  do v_i ← SAMPLE()
3  for i ← 1, ..., d in parallel
4  do v_i ⇒ INSERT(u, i)
```

```
SAMPLE()
1  return a node in the graph containing self, uniformly at random
```

Procedure SAMPLE() returns a node of the graph chosen uniformly at random. Implementations of SAMPLE are presented in Section 2.4.

```
LEAVE()
1  for i ← 1, ..., d in parallel
2  do LINK(ngbr_{-i}, ngbr_i, i)
```

We first show that JOIN can preserve uniform probability spaces.

Lemma 1. *Let $(\mathbb{H}_{n-1,2d}, \Sigma, P)$ be a uniformly distributed probability space. After an operation* JOIN, *the probability space $(\mathbb{H}_{n,2d}, \Sigma', P')$ is also uniformly distributed.*

Proof. Let $G = (V, E)$ be a uniformly random graph from $\mathbb{H}_{n-1,2d}$. Let $G' = (V', E')$ be the graph obtained after a JOIN operation. Let E_i and E'_i be the set of level-i edges in E and E' respectively, for $i = 1, \ldots, d$.

We need to show that each (V', E'_i) is an independent Hamilton cycle chosen from $\mathbb{H}_n$ uniformly. Procedure SAMPLE returns a node chosen from V uniformly at random. This means that it is equally likely that the new node is inserted between any pair of nodes in the cycle (V, E_i). Therefore, since (V, E_i) is a uniformly random Hamilton cycle in $\mathbb{H}_{n-1}$, (V', E'_i) will be a uniformly random Hamilton cycle in $\mathbb{H}_n$.

Since the d calls to SAMPLE are independent, the Hamilton cycles (V', E'_i) are independent. □

By Lemma 1, we can expect to obtain a uniform probability space if we remove the node that has just joined. However, if the space is uniform, the nodes should not be differentiable by the order of joining. Thus, Lemma 2 shows that the probability space remains uniformly distributed no matter which particular node leaves.

Lemma 2. *Let $(\mathbb{H}_{n,2d}, \Sigma, P)$ be a uniformly distributed probability space. After an operation* LEAVE *at any node in G, the probability space $(\mathbb{H}_{n-1,2d}, \Sigma', P')$ is also uniformly distributed.*

Proof. Let G be a uniform sample from $\mathbb{H}_{n,2d}$. We can consider each Hamilton cycle in G separately. Consider any particular level-i cycle of G. Let j be the node calling LEAVE. Given j, there are $\binom{n-1}{2}$ possible unordered sets of neighbors of j. We can partition $\mathbb{H}_n$ into $\binom{n-1}{2}$ sets according to these neighbors. We call these sets $H_{u,v}$ for each set of neighbors $\{u, v\}$.

Consider the map $\sigma : \mathbb{H}_n \mapsto \mathbb{H}'_{n-1}$, where $\mathbb{H}'_{n-1}$ is the set of all cycles on

$$\{1, \ldots, j-1, j+1, \ldots, n\},$$

such that $\sigma(C)$ is obtained by removing node j from the cycle C, and connecting the neighbors of j. It is clear that $\mathbb{H}'_{n-1}$ is essentially the same as $\mathbb{H}_{n-1}$, but with different labels for the nodes.

Let $\sigma^{-1}(C') = \{C \in \mathbb{H}_n \mid \sigma(C) = C'\}$. Consider any graph $C' \in \mathbb{H}'_{n-1}$. If $(u, v) \notin E(C')$, then $\sigma^{-1}(C) \cap H_{u,v} = \emptyset$. Therefore,

$$\sigma^{-1}(C) \subseteq \bigcup_{(u,v) \in E(C)} H_{u,v}$$

For each adjacent pair $\{u, v\}$ of neighbors in C, $\left|\sigma^{-1}(C) \cap H_{u,v}\right| = 1$, because C is created by removing j from a cycle in $H_{u,v}$, which happens exactly when u, v are the two neighbors of j. And then since C has $n-1$ distinct pairs of neighbors and that the $H_{u,v}$'s are disjoint, we have $\left|\sigma^{-1}(C)\right| = n - 1$.

Since each graph in $\mathbb{H}_n$ has probability $2/(n-1)!$, each graph in $\mathbb{H}_{n-1}$ has probability $(n-1)(2/(n-1)!) = 2/(n-2)!$.

Since the d cycles are independent of each other, $P'(G) = (2/(n-2)!)^d$ for any $G \in \mathbb{H}_{n-1,2d}$. Thus, $(\mathbb{H}_{n-1,2d}, \Sigma', P')$ is uniformly distributed. □

Theorem 3. *Let $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$ be a sequence of probability spaces such that*

- *$\mathcal{S}_0$ is uniformly distributed and $\Omega[\mathcal{S}_0] = \mathbb{H}_{k,2d}$ for some k;*
- *$\mathcal{S}_{i+1}$ is formed from $\mathcal{S}_i$ by* JOIN *or* LEAVE*;*
- *$|\Omega[\mathcal{S}_i]| \geq 3$ for all $i \geq 0$.*

Then $\mathcal{S}_i$ is uniformly distributed for all $i \geq 0$.

Proof. We start with a $\mathbb{H}$-graph of size 3. We note that $|\mathbb{H}_{3,2d}| = 1$, thus the initial probability space $\mathcal{S}_0$ has uniform distribution. The theorem then follows from Lemma 1 and Lemma 2 by induction. □

A graph G of size n has a corresponding n by n matrix A, in which the entry A_{ij} is the number of edges from node i to node j. Let $\lambda(G)$ be the second largest eigenvalue of graph G's matrix. Friedman [32] showed that random regular graphs have close to optimal $\lambda(G)$ with high probability. Although he mainly considered the graphs that are composed of d random permutations, he also showed that his results hold for graphs composed of d random Hamilton cycles. Theorem 4 restates a recent improvement by Friedman [33].

Theorem 4 (Friedman). *Let G be a graph chosen from $\mathbb{H}_{n,2d}$ uniformly at random. For any $\epsilon > 0$,*

$$\lambda(G) \leq 2\sqrt{2d-1} + \epsilon$$

with probability $1 - O(n^{-\tau})$, where $\tau = \lceil \sqrt{2d-1} \rceil - 1$.

It has been known that $\lambda(G) \geq 2\sqrt{2d-1} + O(1/\log_d n)$ for any $2d$-regular graph. Therefore, no family of $2d$-regular graphs can have smaller asymptotic $\lambda(G)$ bounds than Theorem 4.

As a consequence of Theorem 4, $\mathbb{H}$-graphs are expanders with $O(\log_d n)$ diameter with high probability because of the relation between eigenvalues and expanders [7, 52].

Let $\rho(G) \equiv \lambda(G)/\Delta(G)$ for regular graph G of degree $\Delta(G)$. We note that $\rho(G)$ is the second largest eigenvalue of the Markov chain with transitions based on G.

2.4 Perfect Sampling

In this section, we will discuss several implementations for procedure SAMPLE of Section 2.3.

2.4.1 Global Server

In a simple centralized solution using a publicly-known sampling server, each joining node can obtain d uniformly random nodes from the sampling server. Each JOIN or LEAVE operation only takes $O(1)$ time and $O(1)$ messages. The space required at the sampling server is $O(n)$.

The sampling server sampleserver keeps the addresses of all nodes of the network.

```
SAMPLE-CENTRAL()
1  return sampleserver ⇒ RANDOM-NODE()
```

```
RANDOM-NODES()
1  return UNIFORM(G)
```

This process take $O(1)$ time and $2d$ messages. We can also easily combine those $2d$ messages into a single query. Whenever a node is added or removed from the network, $O(1)$ maintenance messages are required. The space (number of addresses) required at the sampling servers are $O(n)$.

This is a simple and efficient solution. However, it subjects to most of the disadvantages of a centralized solution. For example, we need to have a complete trust on the sampling server returning unbiased samples of the network. We also need to have a reliable public source for the server's address.

Although this solution is subject to most disadvantages of centralized systems, it is still better than a fully centralized solution because only an address list is stored at the sampling server. It is not required to store any description of the nodes, as searching is still distributed. Thus the load on the sampling server only depends on the rate of topology changes but not the search queries and other algorithms running on the network.

2.4.2 Broadcast

Instead of using a central server, a joining node can broadcast a request to all other nodes. Each node is asked to reply with a chosen probability $p = O(1/n)$. Thus the joining node can expect to receive $O(1)$ replies and then pick one as the sample. A broadcast on an $\mathbb{H}$-graph sends at most $(2d-1)n$ messages and terminates in $O(\log_d n)$ steps with high probability (Section 2.8).

The procedure SAMPLE-BROADCAST calls the generic service BROADCAST (Section 2.8) so that SAMPLE-PROC is executed once on every node of the graph. Parameter p is the probability of replying.

```
SAMPLE-BROADCAST(p)
 1  repeat
 2          S_1 ← S_2 ← ··· ← S_d ← ∅
 3          P ← NEW (SAMPLE-PROC, {s ← self, p ← p})
 4          BROADCAST(P)
 5          SLEEP(Θ(log n))
 6    until min_{1≤i≤d} |S_i| > 0
 7  S ← ∅
 8  for i ← 1 to d
 9  do S ← S ∪ UNIFORM(S_i)
10  return S
```

Procedure SAMPLE-BROADCAST needs an estimation of $\log n$. Estimation algorithms are discussed in Section 2.10.

In procedure SAMPLE-PROC, the set I keeps the set of levels (Hamilton cycles) that a node decides to respond to. Set I is a subset of $\{1,\ldots,d\}$ such that for each $i \in \{1,\ldots,d\}$, $i \in I$ with probability p.

```
SAMPLE-PROC()
1  locals s, p
2  I ← ∅
3  for i ← 1 to d
4  do I ← I ∪ i with probability p
5  if I ≠ ∅
6    then s ⇒ SAMPLE-REPLY(self, I)
7  return true
```

```
SAMPLE-REPLY(u, I)
1  for i ∈ I
2  do S_i ← S_i ∪ u
```

The replies are then aggregated at the variables S_i at the source node s. The probability that any set S_i is empty is $(1-p)^n$. The probability that all sets S_i are non-empty is $(1-(1-p)^n)^d$. Thus, in expectation, $(1-(1-p)^n)^{-d}$ rounds of broadcast are needed.

The probability that a node calls the broadcast source s with SAMPLE-REPLY is $1-(1-p)^d$. Thus the source s expects to receive $n(1-(1-p)^d)$ reply messages during each round of broadcast. The overall expected number of reply messages is $n\frac{1-(1-p)^d}{(1-(1-p)^n)^d}$. Let $p = \beta/n$. For large n,

$$n\frac{1-(1-p)^d}{(1-(1-p)^n)^d} \approx \frac{d\beta}{(1-e^{-\beta})^d}.$$

d	β	$\frac{d\beta}{(1-e^{-\beta})^d}$
4	2.337	14.036
8	3.315	35.658
16	4.229	85.570
32	5.101	198.476

Table 2.1: The approximate optimal values of β (so that $p = \beta/n$) and the expected number of replied messages generated by SAMPLE-BROADCAST.

Therefore, it is possible to select the parameter p so that the expected total number of reply messages is a function of d. The function $\frac{d\beta}{(1-e^{-\beta})^d}$ is minimized when $e^{\beta} - 1 = \beta d$. Table 2.1 gives the optimal values of β and $\frac{d\beta}{(1-e^{-\beta})^d}$ for several choices of d.

2.4.3 Converge-Cast

In the worst case, the source node has to handle $O(n)$ replies resulting from a broadcast. We note that a broadcast can construct a $O(\log n)$-depth spanning tree in $O(\log n)$ time. Instead of asking each node to reply independently, a protocol can 'combine' the sampling results at the internal nodes of the spanning tree, so that each node only needs to handle $O(d)$ messages in the worst case.

```
SAMPLE-SPANNING(PROC, nonce h)
 1  if h was seen recently
 2     then return NULL
 3  for each v ∈ N(self) in parallel
 4  do S ← S ∪ {v ⇒ SAMPLE-SPANNING(PROC, h)}
 5  z ← self
 6  t ← 1
 7  for each (z', t') in S
 8  do t ← t + t'
 9     with probability t'/t
10                    then z ← z'
11  return (z, t)
```

Theorem 5. *Procedure* SAMPLE-SPANNING *returns each node in the graph G with probability $1/|G|$.*

Proof. First, we observe that SAMPLE-SPANNING establishes a spanning tree rooted at the initial node. Thus, we only need to show that in any subtree T rooted at node v, each node in T is returned with probability $1/|T|$.

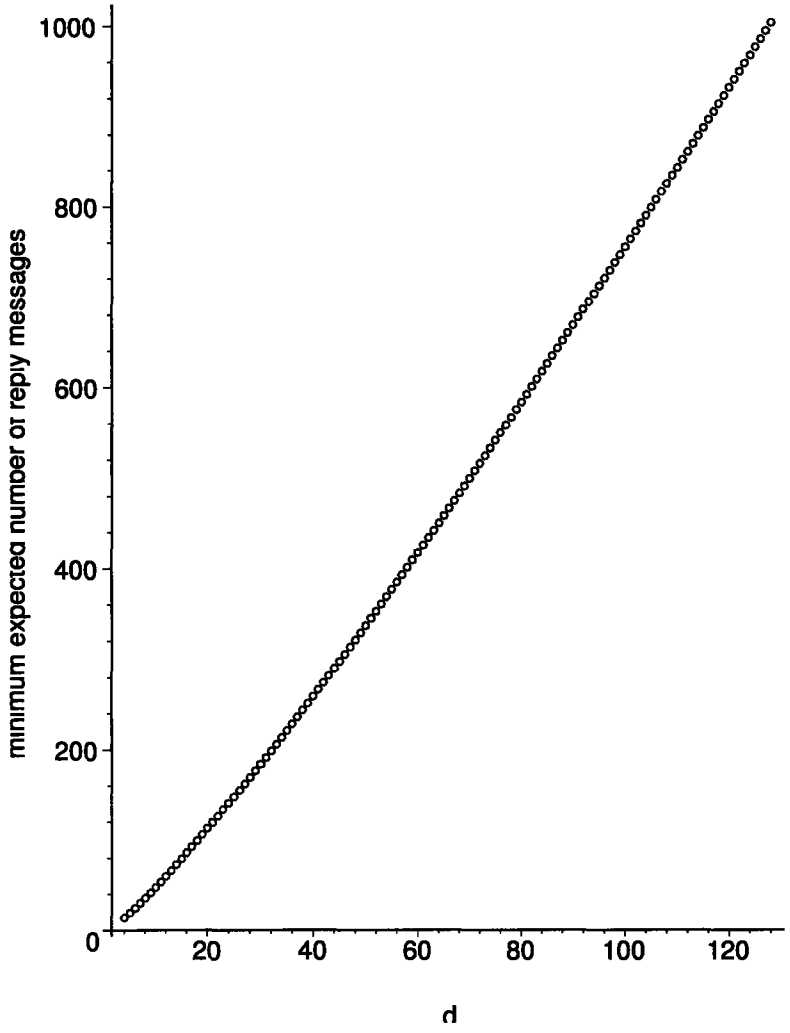

Figure 2.2: The minimum expected number of reply messages received in SAMPLE-BROADCAST for $d = 4, 5, \ldots, 128$.

For the base case, a leaf has to return itself with probability 1. Inductively, let $T_1, \ldots, T_k$ be the subtrees of v so that the `for` loop in SAMPLESPANNING goes through these trees in order of $T_1, \ldots, T_k$. Let z' be the node returned by subtree T_i. In the ith iteration of the `for` loop, there is a probability of $\frac{|T_i|}{1+\sum_{1\le l\le i}|T_l|}$ that z' replaces the existing value of variable z. And the probability that z' is not replaced in loop $j > i$ is $\frac{1+\sum_{1\le l\le j-1}|T_l|}{1+\sum_{1\le l\le j}|T_l|}$. Therefore, the probability that z' is returned by $v \Rightarrow$ SAMPLESPANNING is

$$\begin{aligned}
&\frac{|T_i|}{1+\sum_{1\le l\le i}|T_l|}\frac{1+\sum_{1\le l\le i}|T_l|}{1+\sum_{1\le l\le i+1}|T_l|}\cdots\frac{1+\sum_{1\le l\le k-1}|T_l|}{1+\sum_{1\le l\le k}|T_l|}\\
&=\frac{|T_i|}{1+\sum_{1\le l\le k}|T_l|}\\
&=\frac{|T_i|}{|T|}.
\end{aligned}$$

Since each node in T_i is selected with probability $1/|T_i|$ by our inductive assumption, each node in T_i will be returned by v with probability $1/|T|$. It is straightforward to verify that v itself is returned with probability $1/|T|$. □

We note that SAMPLE-SPANNING is a broadcast together with information aggregation along the broadcast tree from the leaves to the root. Therefore, it has $O(\log_d n)$ time complexity and $O(dn)$ message complexity.

2.4.4 Coupling From The Past

Perfect sampling algorithms for Markov chains have been discovered in recent years. Lovász and Winkler [68] gave the first exact sampling algorithm for unknown Markov chains. Propp and Wilson [81] introduced faster algorithms based on their 'coupling from the past' (CFTP) procedure. They gave an algorithm COVER-CFTP which generates a state distributed with stationary π in expected time at most 15 times the cover time. The expected cover time for $\mathbb{H}$-graphs is around $n \log n$ [6] with high probability. Thus the running time of COVER-CFTP on an $\mathbb{H}$-graph is $O(n \log n)$. It is not clear if a faster CFTP algorithm can be found by discovering a monotone coupling [80].

2.5 Approximate Sampling

The sampling algorithms in Section 2.4 are either centralized or require $\Omega(n)$ messages, and thus are not satisfactory for our goal of efficient and distributed construction. The existence of a distributed perfect sampling algorithms for regular graphs with $o(n)$ message complexity is an open problem.

Sampling Algorithm	Time	Messages	Space	Sampling
Global Server	$O(1)$	$O(1)$	$O(n)$	perfect
Broadcast	$O(\log_d n)$	$O(dn)$	$O(d)$	perfect
CFTP	$O(n \log n)$	$O(n \log n)$	$O(d)$	perfect
Random Walk	$O(\log_d n)$	$O(\log_d n)$	$O(d)$	approximate

Table 2.2: The complexities of the sampling algorithms. "Space" is the worst-case storage requirement at any node.

In this section, we describe an approach for approximate sampling using random walks. A graph can be considered as a Markov chain. Sampling by random walks is usually called Markov Chain Monte Carlo [48]. Since $\mathbb{H}$-graphs are regular, the limiting distribution of random walks on $\mathbb{H}$-graphs is the uniform distribution. Moreover, since an $\mathbb{H}$-graph is an expander with high probability, a random walk of $O(\log n)$ steps on an $\mathbb{H}$-graph can sample the nodes of the graph with a distribution close to be uniform. We shall show that our protocol with approximate sampling increases the probability of producing 'bad graphs' (those with large $\lambda(G)$) by only a constant factor. Therefore, if the graph is sufficiently large, the high probability result of Theorem 4 can still be applied. With approximate sampling, we need to start with a uniformly distributed probability space of sufficiently large graphs. When the graph is small, we can use a perfect sampling algorithm described in Section 2.4.

Procedure SAMPLE-RW(t) performs a random walk on the graph and returns the node after t steps.

```
SAMPLE-RW(t)
1  if t = 0
2     then return self
3     else  v ← a random element in N(self)
4           return v ⇒ SAMPLE-RW(t − 1)
```

Procedure SAMPLE-RW is presented here as a tail recursion through remote procedure calls. We can as well pass along the network address of the initial node. Procedure SAMPLE-RW can serve as an implementation of SAMPLE in Section 2.3. The variant of JOIN using SAMPLE-RW will be denoted as JOIN-RW. Table 2.2 summarizes the complexities of the sampling algorithms we have considered.

2.5.1 Nodes Joining

We now analyze the sequence of probability spaces of $\mathbb{H}$-graphs generated by the JOIN-RW operations. Lemmas 6 and 7 are consequences of Theorem 5.1 in [67].

Lemma 6. *If G is a k-regular graph of size n, then for any $i \in G$,*

$$\left|\Pr\{\text{SAMPLE-RW}(t) \text{ returns } i\} - \frac{1}{n}\right| \leq \left(\frac{\lambda(G)}{k}\right)^t.$$

Proof. By Theorem 5.1 of [67], we have

$$|P_t(j) - \pi(j)| \leq \sqrt{\frac{\pi(j)}{\pi(i)}}\rho^t,$$

where $\rho = \frac{\lambda(G)}{k}$. Since $\pi(j) = \pi(i) = 1/n$ for all i, j, $\sqrt{\frac{\pi(j)}{\pi(i)}} = 1$. □

Lemma 7. *Let $G \in \mathbb{H}_{n,2d}$ be a graph such that $\lambda(G) \leq 2\sqrt{2d-1} + \epsilon$ for $d \geq 2$. Let*

$$t = \left\lceil \frac{\log \frac{dn^r}{c}}{\log \frac{2d}{2\sqrt{2d-1}+\epsilon}} \right\rceil = O\left(\log_d n\right)$$

where r and c are positive constants. Then for any $v \in G$, we have

$$\left|\Pr\{\text{SAMPLE-RW}(t) \text{ returns } v\} - \frac{1}{n}\right| \leq \frac{c}{dn^r}.$$

Proof. Let $\rho = \frac{2\sqrt{2d-1}+\epsilon}{2d}$ and $t^* = \log_{1/\rho} dn^r/c$. By Lemma 6, if $t \geq t^*$, then $\rho^t \leq c/dn^r$. □

Theorem 8 shows that although our protocol using SAMPLE-RW does not sample perfectly, the probability that we obtain a graph of large second eigenvalue remains very small.

Theorem 8. *Let $d \geq 3$. Let $\mathcal{S}_n, \mathcal{S}_{n+1}, \mathcal{S}_{n+2}, \ldots$ be a sequence of probability spaces where $\mathcal{S}_k = (\mathbb{H}_{k,2d}, \Sigma_k, P_k)$. Let $\mathcal{S}_n$ be a uniformly distributed probability space and let $\mathcal{S}_{k+1}$ be formed from $\mathcal{S}_k$ by operation* JOIN-RW *using* SAMPLE-RW$\left(\left\lceil \frac{\log \frac{dk^r}{c}}{\log \frac{2d}{2\sqrt{2d-1}+\epsilon}} \right\rceil\right)$ *with $c > 0$ and $r > 2$. Then for all $k \geq n$,*

$$P_k\left\{ \left\{ G \in \mathbb{H}_{k,2d} \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon \right\} \right\} \geq 1 - O(n^{-\tau}),$$

where $\tau = \lceil \sqrt{2d-1} \rceil - 1$.

Proof. Let $\Upsilon_n = \left\{ G \in \mathbb{H}_{n,2d} \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon \right\}$ be the set of 'good' graphs in $\mathbb{H}_{n,2d}$. Let $J(G)$ be the set of graphs obtained by operation JOIN (it does not matter whether we consider JOIN or JOIN-RW) on G. For any $k \geq n$, let

$$\Upsilon_{k+1} \equiv \left\{ G \in J(\Upsilon_k) \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon \right\}. \tag{2.1}$$

The probability space $\mathcal{S}_k$ can be considered as a product space $(\mathcal{C}_k)^d$ where $\Omega[\mathcal{C}_k] = \mathbb{H}_k$. For any $C \in \mathbb{H}_k$ and $l \in \{1, \ldots, d\}$, let

$$H_k[l, C] \equiv \{\, G \in \mathbb{H}_{k,2d} \mid C \text{ is the level-}l \text{ Hamilton cycle of } G \,\}$$

be the set of graphs whose level-l cycle is C.

For any probability space (Ω, Σ, P) and any sets $A, B \subseteq \Omega$, let

$$P\{A \mid B\} \equiv P\{A \cap B\} / P\{B\}.$$

We shall prove that for $m \geq n$,

$$P_m\{\Upsilon_m\} \geq 1 - \sum_{j=n}^{m} e^{c \sum_{i=n}^{j-1} i^{1-r}} O(j^{-\tau}), \tag{2.2}$$

and

$$\sup_{\substack{C \in \mathbb{H}_m \\ 1 \leq l \leq d}} P_m\{\, H_m[l, C] \mid \Upsilon_m \,\} \leq \frac{e^{\frac{c}{d} \sum_{i=n}^{m-1} i^{1-r}}}{(m-1)!/2}. \tag{2.3}$$

Because of our assumption that $\mathcal{S}_n$ is a uniformly distributed probability space and Theorem 4, $P_n\{\Upsilon_n\} \geq 1 - O(n^{-\tau})$. The eigenvalue of a graph depends on the structure of the graph but not the labels. In other words, a graph has the same eigenvalue no matter how we label the nodes. Thus, we have

$$P_n\{\, H_n[l, C] \mid \Upsilon_n \,\} = P_n\{H_n[l, C]\} = \frac{1}{(n-1)!/2}.$$

Therefore, Inequalities (2.2) and (2.3) are satisfied for the base case $m = n$.

Assuming that Inequalities (2.2) and (2.3) are satisfied for $m = k$, we shall show that they are satisfied for $m = k + 1$.

Let $\mathbf{t}(k, d) = \left\lceil \frac{\log \frac{dk^r}{c}}{\log \frac{2d}{2\sqrt{2d-1}+\epsilon}} \right\rceil$ be the number of steps walked by SAMPLE-RW.

The set $J(\Upsilon_k)$ are those graphs that can be produced by an operation JOIN on the graphs in Υ_k. Let $C' \in \mathbb{H}_k$ be the cycle such that node $k+1$ is removed from a cycle $C \in \mathbb{H}_{k+1}$. A graph in $H_{k+1}[l, C] \cap J(\Upsilon_k)$ must be created by inserting the node $k+1$ at level l between two nodes of a graph in $H_k[l, C'] \cap \Upsilon_k$. We have

$$\begin{aligned}
&\sup_{C \in \mathbb{H}_{k+1},\, 1 \leq l \leq d} P_{k+1}\{\, H_{k+1}[l, C] \mid J(\Upsilon_k) \,\} \\
&\leq \sup_{C' \in \mathbb{H}_k,\, 1 \leq l \leq d} P_k\{\, H_k[l, C'] \mid \Upsilon_k \,\} \times \\
&\quad \sup_{v \in [k], G \in \Upsilon_k} \Pr\{\text{SAMPLE-RW}(\mathbf{t}(k, d)) \text{ on } G \text{ returns } v\}.
\end{aligned}$$

Let

$$\sup_{v\in[k],G\in\Upsilon_k} \Pr\{\text{SAMPLE-RW}(\mathbf{t}(k,d)) \text{ on } G \text{ returns } v\} = 1/k + \epsilon_k.$$

Informally, ϵ_k is the amount of deviation resulted from the approximate sampling on some $G \in \Upsilon_k$ by a random walk. It would be zero if a perfect sampling algorithm is used. By Lemma 7, $\epsilon_k \leq \frac{c}{dk^r}$. By the inductive assumption of Inequality (2.3) of $m = k$, we have

$$\begin{aligned}
&\sup_{C\in\mathbb{H}_{k+1},\,1\leq l\leq d} P_{k+1}\{H_{k+1}[l,C] \mid \Upsilon_{k+1}\} \\
&\sup_{C\in\mathbb{H}_{k+1},\,1\leq l\leq d} P_{k+1}\{H_{k+1}[l,C] \mid J(\Upsilon_k)\} \\
&\leq \frac{e^{\frac{c}{d}\sum_{i=n}^{k-1} i^{1-r}}}{(k-1)!/2}\,\frac{1+\frac{c}{dk^{r-1}}}{k} \\
&\leq \frac{e^{\frac{c}{d}\sum_{i=n}^{(k+1)-1} i^{1-r}}}{((k+1)-1)!/2}.
\end{aligned}$$

Since all pairs (C, l) are symmetric, we have

$$\sup_{C\in\mathbb{H}_{k+1},\,1\leq l\leq d} P_{k+1}\{H_{k+1}[l,C] \mid \Upsilon_{k+1}\} = \sup_{C\in\mathbb{H}_{k+1},\,1\leq l\leq d} P_{k+1}\{H_{k+1}[l,C] \mid J(\Upsilon_k)\}.$$

Thus Inequality (2.3) is satisfied for $m = k + 1$.

Since a new node is inserted into the d Hamilton cycles independently during each JOIN operation, we have

$$\begin{aligned}
&\sup_{G\in J(\Upsilon_k)} P_{k+1}\{G \mid J(\Upsilon_k)\} \\
&\leq \prod_{1\leq l\leq d} \sup_{C\in\mathbb{H}_{k+1}} P_{k+1}\{H_{k+1}[l,C] \mid J(\Upsilon_k)\} \\
&\leq \left(\frac{e^{\frac{c}{d}\sum_{i=n}^{(k+1)-1} i^{1-r}}}{((k+1)-1)!/2}\right)^d.
\end{aligned}$$

Dividing both sides by $U_{\mathbb{H}_{k+1,2d}}\{G\}$, we have

$$\sup_{G\in J(\Upsilon_k)} \frac{P_{k+1}\{G \mid J(\Upsilon_k)\}}{U_{\mathbb{H}_{k+1,2d}}\{G\}} \leq e^{c\sum_{i=n}^{(k+1)-1} i^{1-r}}. \tag{2.4}$$

Starting from Equation (2.1), we have

$$\begin{aligned}
&P_{k+1}\{\Upsilon_{k+1}\} \\
&= P_{k+1}\left\{\left\{ G \in J(\Upsilon_k) \mid \lambda(G) \leq 2\sqrt{2d-1}+\epsilon \right\}\right\} \\
&= \sum_{G\in J(\Upsilon_k),\, \lambda(G)\leq 2\sqrt{2d-1}+\epsilon} P_{k+1}\{G\} \\
&= \sum_{G\in J(\Upsilon_k)} P_{k+1}\{G\} - \sum_{G\in J(\Upsilon_k),\, \lambda(G)>2\sqrt{2d-1}+\epsilon} P_{k+1}\{G\}. \qquad (2.5)
\end{aligned}$$

The first term can be derived from our inductive assumption that Inequality (2.2) is satisfied for $m = k$:

$$\begin{aligned}
\sum_{G\in J(\Upsilon_k)} P_{k+1}\{G\} &= P_{k+1}\{J(\Upsilon_k)\} \\
&= P_k\{\Upsilon_k\} \\
&= 1 - \sum_{j=n}^{k} e^{c\sum_{i=n}^{j-1} i^{1-r}} O(j^{-\tau}). \qquad (2.6)
\end{aligned}$$

Given Theorem 4 and Inequality (2.4), the second term can be bounded as follows:

$$\begin{aligned}
&\sum_{G\in J(\Upsilon_k),\, \lambda(G)>2\sqrt{2d-1}+\epsilon} P_{k+1}\{G\} \\
&\leq \sum_{G\in J(\Upsilon_k),\, \lambda(G)>2\sqrt{2d-1}+\epsilon} P_{k+1}\{G \mid J(\Upsilon_k)\} \\
&\leq \sum_{G\in J(\Upsilon_k),\lambda(G)>2\sqrt{2d-1}+\epsilon} e^{c\sum_{i=n}^{(k+1)-1} i^{1-r}} U_{\mathbb{H}_{k+1,2d}}\{G\} \\
&< e^{c\sum_{i=n}^{k} i^{1-r}} U_{\mathbb{H}_{k+1,2d}}\left\{\left\{ G \in \mathbb{H}_{k+1,2d} \mid \lambda(G) > 2\sqrt{2d-1}+\epsilon \right\}\right\} \\
&< e^{c\sum_{i=n}^{k} i^{1-r}} O((k+1)^{-\tau}). \qquad (2.7)
\end{aligned}$$

Substituting Equations (2.6) and (2.7) back into Equation (2.5), we obtain

$$P_{k+1}\{\Upsilon_{k+1}\} \geq 1 - \sum_{j=n}^{k+1} \left(e^{c\sum_{i=n}^{j-1} i^{1-r}}\right) O(j^{-\tau}).$$

Since $r > 2$, $e^{c\sum_{i=n}^{k} i^{1-r}}$ is bounded by a constant. Similarly, $\sum_{j=n}^{k} j^{-\tau}$ is bounded by a constant since $\tau > 1$. In fact, $\sum_{j=n}^{k} j^{-\tau}$ becomes negligible quickly with moderate τ. For

example, when $\tau = 4$ and $n = 50$, we have $\sum_{j=50}^{\infty} j^{-4} < 3 \times 10^{-6}$. Therefore, for any $k \geq n$,

$$\begin{aligned} & P_k\left\{\left\{G \in \mathbb{H}_{k,2d} \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon\right\}\right\} \\ \geq\ & P_k\{\Upsilon_k\} \\ \geq\ & 1 - \sum_{j=n}^{k} e^{c\sum_{i=n}^{j-1} i^{1-r}} O(j^{-\tau}) \\ =\ & 1 - O(n^{-\tau}). \end{aligned}$$

□

We note that Theorem 8 is useful only when the initial graph size n is sufficiently large for $O(n^{-\tau})$ to be close to zero. Section 3.2 considers the transitions between complete graphs and $\mathbb{H}$-graphs.

2.5.2 Nodes Leaving

The operation LEAVE can have a much larger effect on the probability of obtaining bad graphs. We can obtain a similar result if the number of operations is bounded by a power of the size n of the graphs in the initial uniformly distributed probability space. The proof of Theorem 9 is similar to that of Theorem 8.

Theorem 9. *Consider a sequence $\mathcal{S}_0, \ldots, \mathcal{S}_N$ of probability spaces and an integer n such that*

- *$\langle \sigma_1, \sigma_2, \ldots, \sigma_N \rangle$ is a sequence of operations, such that each operation σ_i is either a* LEAVE *or a* JOIN-RW *using* SAMPLE-RW $\left(\left\lceil \frac{\log \frac{d(i+n)^r}{c}}{\log \frac{2d}{2\sqrt{2d-1}+\epsilon}} \right\rceil\right)$ *where $c > 0$ and $r > 2$;*
- *$\mathcal{S}_0$ is a uniformly distributed probability space and $\Omega[\mathcal{S}_0] = \mathbb{H}_{n,2d}$;*
- *$\mathcal{S}_i$ is obtained from $\mathcal{S}_{i-1}$ by operation σ_i for $i = 1, \ldots, N$;*
- *$|\Omega[\mathcal{S}_i]| \geq n$ for all $i \in \{0, \ldots, N\}$.*

Let $\tau = \lceil \sqrt{2d-1} \rceil - 1$. If $N = O(n^{tau})$ such that $q < r - 1$, then

$$P_i\left\{\left\{G \in \Omega[\mathcal{S}_i] \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon\right\}\right\} \geq 1 - O(n^{q-\tau})$$

for $i = 1, \ldots, N$.

Proof. For $i = 0, \ldots, N$, let $n_i = |\Omega[\mathcal{S}_i]|$. Let the set $\Omega[\mathcal{S}_0]$ of the nodes in the initial probability space be $\{-1, -2, \ldots, -n\}$. If σ_i is a JOIN operation, then the new node is labeled i. In this way, the requests can be represented as integers, such that

$$\Omega[\mathcal{S}_{i+1}] \equiv \begin{cases} \Omega[\mathcal{S}_i] \cup \{\sigma_{i+1}\} & \text{if } \sigma_{i+1} = i+1, \\ \Omega[\mathcal{S}_i] \setminus \{\sigma_{i+1}\} & \text{if } \sigma_{i+1} \in \Omega[\mathcal{S}_i]. \end{cases}$$

The request σ_i must either adds a node labeled i or removes an existing node in $\Omega[\mathcal{S}_{i-1}]$. For notational simplicity, we still use $\mathbb{H}_{n_i}$ for the set of Hamilton cycles on $\Omega[\mathcal{S}_i]$, although the labels of the nodes change after each request.

Let

$$\Upsilon_0 \equiv \left\{ G \in \Omega[\mathcal{S}_0] \mid \lambda(G) \leq 2\sqrt{2d-1} + \epsilon \right\}.$$

Depending on the operation σ_{i+1}, we have

$$\Upsilon_{i+1} \equiv \begin{cases} \left\{ G \in J(\Upsilon_i) \mid \lambda(G) \leq \sqrt{2d-1} + \epsilon \right\} & \text{if } \sigma_{i+1} \text{ is a JOIN}, \\ \left\{ G \in L_{\sigma_{i+1}}(\Upsilon_i) \mid \lambda(G) \leq \sqrt{2d-1} + \epsilon \right\} & \text{if } \sigma_{i+1} \text{ is a LEAVE}, \end{cases}$$

where $L_v(A)$ is the set of graphs obtained by removing node v in the graphs of A.

We can derive a sequence z from σ by

$$z_i \equiv \begin{cases} 1 & \text{if } \sigma_i \text{ is a JOIN operation}, \\ 0 & \text{if } \sigma_i \text{ is a LEAVE operation}. \end{cases}$$

Let

$$g(j) = \sum_{i=0}^{j-1} z_i n_i^{1-r}.$$

We shall prove the following inequalities for $m = 0, 1, \ldots, N$:

$$P_m\{\Upsilon_m\} \geq 1 - \sum_{j=0}^{m} z_j e^{cg(j)} O(n_j^{-\tau}),$$

and

$$\sup_{\substack{C \in \mathbb{H}_{n_m} \\ 1 \leq l \leq d}} P_m\{ H_{n_m}[l, C] \mid \Upsilon_m \} \leq \frac{e^{\frac{c}{d} g(m)}}{(n_m - 1)!/2}.$$

The base case and the inductive step for a JOIN operation is the same as the proof in Theorem 8. In the following, we will consider the LEAVE operations.

Consider a LEAVE operation σ_{k+1}. Fix any $l \in \{1, \ldots, d\}$. A cycle $C \in \mathbb{H}_{n_{k+1}}$ can be obtained by removing node σ_{k+1} in exactly $n_{k+1} = n_k - 1$ different cycles. Therefore,

$$\begin{aligned}
&\sup_{C \in \mathbb{H}_{n_{k+1}}, 1 \le l \le d} P_{k+1} \left\{ H_{n_{k+1}}[l, C] \mid \Upsilon_{k+1} \right\} \\
&\sup_{C \in \mathbb{H}_{n_{k+1}}, 1 \le l \le d} P_{k+1} \left\{ H_{n_{k+1}}[l, C] \mid L_{\sigma_{k+1}}(\Upsilon_k) \right\} \\
\le &\sup_{C \in \mathbb{H}_{n_k}, 1 \le l \le d} (n_k - 1) P_k \left\{ H_{n_k}[l, C] \mid \Upsilon_k \right\} \\
\le &(n_k - 1) \frac{e^{\frac{c}{d} g(k)}}{(n_k - 1)!/2} \\
= &\frac{e^{\frac{c}{d} g(k+1)}}{(n_{k+1} - 1)!/2}.
\end{aligned}$$

We can show that $g(j)$ is bounded by a constant:

$$\begin{aligned}
g(j) &= \sum_{i=0}^{j-1} z_i n_i^{1-r} \\
&\le \sum_{i=0}^{N-1} n^{1-r} \\
&= N n^{1-r} \\
&= o(1)
\end{aligned}$$

because $N = o(n^{r-1})$.

At last, we can conclude that for $k = 1, \ldots, N$,

$$\begin{aligned}
P_k \{\Upsilon_k\} &\ge 1 - \sum_{j=0}^{k} z_j e^{cg(j)} O(n_j^{-\tau}) \\
&= 1 - \sum_{j=0}^{k} z_j O(n_j^{-\tau}) \\
&= 1 - N O(n^{-\tau}) \\
&= 1 - O(n^{q-\tau}).
\end{aligned}$$

□

In the long run, the probability spaces may deviate further and further away from the uniformly distributed spaces. (We expect that, in practice, the 'degradation' will be very

slow, especially if the leaving nodes are not chosen by an adversary.) In the following, we outline two strategies for slowing down the deviation. When the request sequence is unbounded and arbitrary, we can also use a regeneration algorithm (Section 2.7.2) to refresh the probability space. In Section 2.11 we study the effects of leaving nodes on the second eigenvalue of the graph.

Youngest Node

We observe that if the node to leave is always the node just joined (the youngest node), then Theorem 8 would remain applicable. But we cannot avoid other nodes leave before the youngest node z. Nevertheless, when some node v wants to leave, it can just swap its set of neighbors $N(v)$ with the neighbors of z. Thus, for the graph topology, it would appear as if the youngest node z has left. We call the ordered list of the neighbors $\langle \mathsf{ngbr}_{-1}, \mathsf{ngbr}_1, \mathsf{ngbr}_{-2}, \ldots, \mathsf{ngbr}_{-d}, \mathsf{ngbr}_d \rangle$ the *shell* of a node. Swapping shells takes $O(1)$ time and $O(d)$ messages.

Now our problem reduces to locating the youngest node z by any leaving node. Let g be a server known to all nodes in the graph. Each node $u \in G$ contains a field prev that points to the node that joined just before u. Server g's variable youngest should always point to the current youngest node.

- When the youngest node z leaves, $g \Rightarrow$ youngest is updated to $z \Rightarrow$ prev.
- When a new node u joins the network, $(u \Rightarrow \mathsf{prev}) \leftarrow (g \Rightarrow \mathsf{youngest})$ and then $g \Rightarrow$ youngest is updated to u.

Only $O(1)$ messages are required for each JOIN-RW or LEAVE operation. We note that only $O(1)$ space is required at the global server.

Instead of using a global server, the address youngest can also be broadcast over the graph after every JOIN-RW or LEAVE operation. A broadcast takes $O(\log_d n)$ time and $O(dn)$ messages.

However, this simple scheme only works for nodes leaving but not failing. It is possible to detect failing nodes by the neighbors but the implementation can get quite complicated.

Shell Recycling

The previous approach either requires a global server or a broadcast during every JOIN-RW or LEAVE. We now investigate a distributed solution without broadcasting.

When a node v departs, it can ask a neighbor u to store v's shell. Then u will simulate node v and have (at most) $4d$ effective neighbors. We call u a *shell-host*. The idea is to save the shell and wait for a new node to adopt it. When there are sufficient shell-hosts, a new node can easily find a shell-host and adopt a shell. This scheme can be extended so that there are at most b extra shells for each shell-host.

When the set of shell-hosts occupies a proportion ψ of the graph, a new node will only need $O(\psi^{-1})$ expected time to find a shell-host. In the worst case that the set of shell-hosts is determined by an adversary, it will still only take $O(\psi^{-1} + \log n)$ expected time to find a shell-host. These search algorithms are studied in Section 2.9.

We note that this will only have a limited effect on the performance of the algorithms running on the graph. If we allow b extra shells for each shell-host, then a graph of size n will be simulating a graph of size $(b+1)n$ in the worst case. Each node has to store $(2d)b$ additional addresses. Since the time complexity of the algorithms presented in this chapter is $O(\log n)$, the extra time cost is only $O(\log b)$.

Although this approach is distributed and efficient, it has the limitation that shell-hosts can become saturated in the extreme case. The number of shells per shell-host dictates the maximum factor that a graph can shrink. For example, if we set the load factor b to be 2, then the entire graph can host at most $2n$ additional shells. This means that a graph shrunk by two-third will run out of shell-hosts.

At last, we note that shell-hosts recycling has the extra benefit of speeding up the process of nodes joining, especially if an expensive perfect sampling algorithm is used.

2.6 Simulation Results

We have performed some simulations to gain more insights into the constructions of the $\mathbb{H}$-graphs. We would like to estimate the constants hidden in the asymptotic results. We would like to obtain some guidelines for setting the various parameters of our algorithms. For example, we would like to know what should be the sufficient size of d for an $\mathbb{H}$-graph of certain size.

According to Theorem 4, if an $\mathbb{H}$-graph G is chosen from a set $\mathbb{H}_{n,2d}$ uniformly at random, then for any $\epsilon > 0$,

$$\lambda(G) \leq 2\sqrt{2d-1} + \epsilon \tag{2.8}$$

with probability $1 - O(n^{-\tau})$, where $\tau = \lceil \sqrt{2d-1} \rceil - 1$. However, it is not clear that given ϵ, what size of n is sufficient so that $O(n^{-\tau})$ is insignificant.

First, we have a program `perfectHGraphs` (in Python)that sample $\mathbb{H}$-graphs from the uniform probability spaces. Given n and d, `perfectHGraphs` generates d independent Hamilton cycles on a set of n nodes. Program `perfectHGraphs` outputs matrices in CSC format.

Program `eigenvalues` (in C++) takes in the matrices generated by `perfectHGraphs` and finds their second largest eigenvalues using the library ARPACK++, which is a C++ wrapper for ARPACK in Fortran. ARPACK contains the fastest eigenvalue solver among the numerical packages we have tested. Program `perfectHGraphs` is written in C++.

Program `eigenAnalyze` (in Python) takes in the eigenvalues found by `eigenvalues` and calculate the number of instances satisfying Inequality (2.8) for a given ϵ.

Our program first creates a 3-node $\mathbb{H}$-graph and then have new nodes joining sequentially. When the size of the graph is $50, 100, \ldots, 1000$, we find the second largest eigenvalue of the graph by calling ARPACK++[40], an eigenvalue package.

We simulate our construction algorithm using sampling algorithm SAMPLE-RW. As suggested by Lemma 7, number of steps taken is set to $\left\lceil 2\log_{d/2}\frac{n^r}{c}\right\rceil + 4$. We fixed $c = 1$ and $r = 3$ (no observable effects on the results when r is set to 1 or 5). Since the purpose of the simulations is to evaluate the second largest eigenvalues of the graphs produced by our protocol, we used the graph size n, instead of an estimate (discussed in Section 2.10), for determining the number of random walk steps.

Let us call a graph 'bad' if it does not satisfy Inequality (2.8). In our simulations, we count the number of bad graphs observed in 100,000 independent trials. In Figure 2.3, with $d = 4$, we can see that the number of bad graphs drops quicker against increasing graph size when the ϵ in Inequality (2.8) is larger. For example, when $\epsilon = 0.1$, we observed 218 bad graphs when $n = 250$, 26 bad graphs when $n = 500$, and 0 bad graph when $n = 1000$.

Next, we investigated the effects of parameter d (half of the node degree). The solid lines in Figure 2.4 represent the number of bad graphs for $d = 4, 8, 16$, with $\epsilon = d/100$. The ϵ is set to be proportional to d because the eigenvalues should be normalized by dividing $2d$, the largest eigenvalue, for a fair comparison. The number of bad graphs decrease with increasing d. The dashed lines represent the graphs using perfect sampling. We did not observe a significant difference between the results using perfect sampling and the results using approximated sampling.

According to our simulation results, $d = 4$ is quite sufficient to obtain a high probability of satisfying Inequality (2.8) when $\epsilon \geq 0.1$. A larger d gives us smaller eigenvalues for small graphs, lower time complexity for joining and diameter (both $O(\log_d n)$), but higher space and message complexities.

The high probability results in Section 2.5 only apply when we start with a uniform probability space of sufficient graph size. However, the simulations show that even if we start from the 3-node $\mathbb{H}$-graph, there is a small probability of obtaining bad graphs in the long run, as most of the bad graphs become 'good' when the graph size increases.

2.7 Auxillary Algorithms

2.7.1 Small Graphs

We note that the high probability results are not very useful when the graph is small. In the analysis in Section 2.5, we need an initial uniform distributed probability space of moderate size. In this subsection we discuss small graphs and their transitions to $\mathbb{H}$-graphs.

Besides the above concern, we also note that our protocol in Section 2.3 is not efficient when the graph is small. If $n \leq 2d + 1$, we can maintain a complete graph with fewer edges

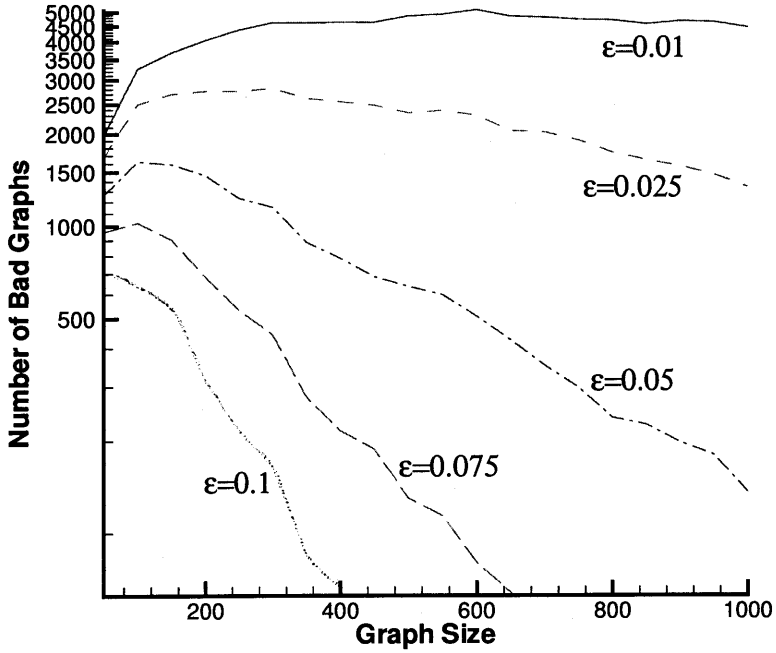

Figure 2.3: Number of graphs not satisfying Inequality (2.8) in 100,000 trials, for $d = 4$ and $\epsilon = 0.01, 0.025, 0.05, 0.075, 0.1$.

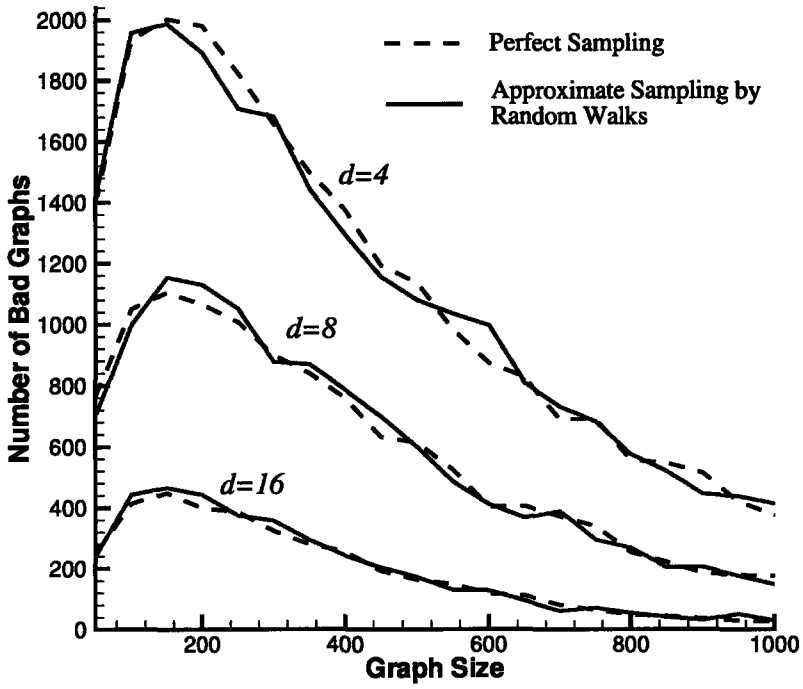

Figure 2.4: Number of graphs not satisfying Inequality (2.8) in 100,000 trials, for $d = 4, 8, 16$ and $\epsilon = d/100$.

than an $\mathbb{H}$-graph. Even when n is slightly larger than $2d$, it might still be more efficient to maintain a complete graph. With a complete graph, each JOIN or LEAVE takes $O(1)$ time and $O(n)$ messages.

The conversion from a complete graph into an $\mathbb{H}$-graph is not difficult because each node has the complete knowledge of the graph. In Section 3.2 we discuss algorithms that convert $\mathbb{H}$-graphs to and from complete graphs.

We note that a complete graph can be formed from a weakly-connected graph by a resource discovery algorithm. These algorithms are discussed in Chapter 4.

2.7.2 Regeneration

Because of various reasons, the probability space produced by our protocol may deviate too far away from the uniformly distributed space. It could be caused by the extraordinary shrinkage discussed in Section 2.5.2 or by node failures. Although we are so far unable to find a distributed algorithm to 'repair' a probability space, we can regenerate the graph by creating a new set of Hamilton cycles. In the following, we will present a regeneration algorithm.

Our approach is to start from a small graph again and insert nodes until the new graph has included all existing nodes. Apparently, it would take up to n steps to construct a new $\mathbb{H}_{n,2d}$ graph. However, it is possible to speed up this process by having nodes joining in parallel. For example, consider a set of new nodes joining G at the same time so that some node $v \in G$ is picked simultaneously by k new nodes on the level-i Hamilton cycle. In this case, v can insert the k nodes between v and $v \Rightarrow \mathsf{ngbr}_i$ in a random order. However, this parallel process should only proceed at a controlled rate. For example, we should not simultaneously insert n nodes into a 3-node graph, because then each of the 3 existing nodes has to manage around $n/3$ new nodes at the same time.

During each round, each node in the new graph can invite its neighbors in the old graph to join. For example, let v_0, v_1, v_2 be the initial members of the new graph. In the first round, they will invite all their neighbors to join the new graph. Let these neighbors be $v_3, \ldots, v_{k_1}$. In the second round, after nodes $v_3, \ldots, v_{k_1}$ have joined, they can in turn invite their neighbors to join. In each round, the number of new nodes is at most $2d-1$ times the current size of the new graph. Therefore, each existing node in the new graph expects to see $O(d^2)$ LINK requests. Effectively, this process expands like a broadcast, which takes $O(\log_d n)$ rounds to cover the entire graph.

We can use a perfect sampling algorithm when the graph is small and switch to SAMPLE-RW after some size n_0. The overall time and message complexities are $O(\log_d^2 n)$ and $O(d(n_0^2 + n\log_d))$.

It is technically impossible to detect a deviation of the probability space. Distributed estimation of the second largest eigenvalue is likely to be very expensive. We can invoke a regeneration based on a fixed schedule or an estimate of the number of LEAVE operations.

There is a tradeoff between the frequency of regeneration and the number of steps taken in sampling process by random walks. When the number of steps by SAMPLE-RW is smaller, the probability space may deviate faster, but this can be compensated by a more frequent invocation of the regeneration algorithm.

2.8 Broadcasts

In this section we discuss broadcasts on the $\mathbb{H}$-graphs. We present a tree-based algorithm with good time complexity and a cycle-based algorithm with good message complexity.

2.8.1 Broadcast with a Spanning Tree

We first describe a generic algorithm for constructing a tree on the graph on demand. The TREEWALK service carries a generic procedure PROC to be executed at each node reached by the algorithm. PROC should return false when the walk should be terminated at the current node, and return true otherwise. In practice, this procedure can be implemented in a portable language to be executed in the sandboxes of the nodes. The algorithm needs to carry along a nonce, which should be a number that will not be repeated for a long time. This is used to ensure that each node executes each instance of TREEWALK at most once. As an example, procedure NEW-NONCE returns a random 64-bit number.

In the following algorithm pseudocodes, the variable self identifies the node executing the procedure. All actions are performed by self by default. The expression $u \Rightarrow$PROC() invokes a remote procedure call PROC at node u. The call is assumed to be non-blocking unless the call returns a value. An expression $u \Rightarrow$ var means that we access the variable var of node u. And, $(u \Rightarrow \mathsf{var}) \leftarrow x$ means that we set the variable var of node u to value x. Thus, messages are exchanged between node self and node u.

```
TREEWALK(PROC, depth t, breadth q, parent u, nonce h)
1  if h is seen recently
2     then return
3  if PROC() and t > 0
4     then S ← a random subset of N(self) \ u of size at most q
5          for each v ∈ S in parallel
6          do v ⇒ TREEWALK(PROC, t − 1, q, self, h)
```

```
NEW-NONCE()
1  return UNIFORM({0, ..., 2^64 − 1})
```

```
UNIFORM(S)
1  return an element in S uniformly at random
```

```
BROADCAST(PROC)
1  TREEWALK(PROC, ∞, ∞, NULL, NEW-NONCE())
```

Effectively, BROADCAST floods any connected graph by constructing a spanning tree, because both the depth and breadth are not constrained.

There is nothing innovative in procedures TREEWALK and BROADCAST by themselves. However, by exploiting the expander property of the $\mathbb{H}$-graphs generated by our protocol, we will be able to show that BROADCAST is fast and efficient on these graphs. The expander property of a graph can be bounded from its second largest eigenvalue.

First, we show that BROADCAST can reach the entire graph in $O(\log n)$ steps.

Lemma 10. *Consider a k-regular graph G of size n such that $\rho = \lambda(G)/k$. The number of nodes reached by* BROADCAST *after t steps is at least*

$$\frac{n}{1+4\frac{n-1}{(\rho^{-1}+\sqrt{\rho^{-2}-1})^{2t}}} \approx \frac{1}{4}\left(\rho^{-1}+\sqrt{\rho^{-2}-1}\right)^{2t}. \tag{2.9}$$

Proof. For any $X \subseteq G$, let $N(X)$ be the set of neighbors of X:

$$N(X) = \bigcup_{u \in X} \{v \mid (v,u) \in E(G)\}.$$

Let $N^t(X)$ be the set of nodes reachable from X after t steps. For any $t \geq 0$, $N^{t+1}(X) = N(N^t(X))$.

Let $\lambda = \lambda(G)$.

By Theorem 2 of [53], we have

$$|N^t(X)| \geq \frac{P_t^2(k/\lambda)\,|X|}{1+(P_t^2(k/\lambda)-1)\,|X|/n},$$

where P_t is the Chebychev polynomial of degree t, which is the unique polynomial satisfying the equation

$$P_t(\cos x) = \cos(tx),$$

for any complex number x.

When $X = \{v\}$, we have

$$\begin{aligned} |N^t(v)| &\geq \frac{P_t^2(k/\lambda)}{1+(P_t^2(k/\lambda)-1)/n} \\ &\geq \frac{n}{1+\frac{n-1}{P_t^2(k/\lambda)}}. \end{aligned} \tag{2.10}$$

Let $x = \cosh y = k/\lambda$. Then $\sinh y = \sqrt{x^2 - 1}$.

$$e^{ty} = (\cosh y + \sinh y)^t = (x + \sqrt{x^2 - 1})^t,$$

for $x > 1$. Then

$$\begin{aligned} P_t(k/\lambda) &= P_t(\cosh y) \\ &= \cosh ty \\ &> e^{ty}/2 = \frac{(x + \sqrt{x^2 - 1})^t}{2}. \end{aligned}$$

Substituting $P_t(k/\lambda)$ back into Equation (2.10), we have

$$\left|N^t(v)\right| \geq \frac{n}{1 + 4\frac{n-1}{(x+\sqrt{x^2-1})^{2t}}}.$$

□

Theorem 11. *Let G be a $2d$-regular $\mathbb{H}$-graph generated by our protocol. Then with high probability, the number of nodes reached by* BROADCAST *after t steps is at least*

$$\frac{n}{1 + 4\frac{n-1}{(2(d-1))^t}}. \tag{2.11}$$

Proof. From Theorem 8, we can pick ϵ such that

$$\rho(G) \leq \frac{2\sqrt{2d-2}}{2d-1}$$

with high probability. Then, we have

$$\begin{aligned} \rho^{-1} + \sqrt{\rho^{-2} - 1} &\geq \frac{2d-1}{2\sqrt{2d-2}} + \frac{\sqrt{4d+1-4d-8d+8}}{2\sqrt{2d-2}} \\ &= \frac{2d-1+2d-3}{2\sqrt{2d-2}} \\ &= \sqrt{2(d-1)}. \end{aligned}$$

This implies that the number of nodes reached by BROADCAST after t steps is at least

$$\frac{n}{1 + 4\frac{n-1}{(2(d-1))^t}}.$$

□

Theorem 12. *With high probability,* BROADCAST *on a $2d$-regular $\mathbb{H}$-graph generated by our protocol terminates in $\left\lfloor 2\frac{\log(n-1)+1}{\log(d-1)+1} \right\rfloor + 1 = O(\log_d n)$ steps.*

Proof. From Lemma 10, the number of nodes covered after t steps is at least

$$\frac{n}{1+4\frac{n-1}{(2(d-1))^t}}.$$

Therefore, we are done after t steps if

$$\frac{n}{1+4\frac{n-1}{(2(d-1))^t}} > n-1,$$

which is equivalent to

$$t > \frac{2\log(n-1)+1}{\log(d-1)+1}.$$

□

Theorem 13. *The number of messages sent by* BROADCAST *on a $2d$-regular graph is at most $(2d-1)n+1$.*

Proof. There are dn undirected edges. The maximum possible number of messages sent is $2dn$ because two messages can be sent on each edge. However, there will be some one-way edges where messages are sent in only one direction.

During the round that a node is discovered, all edges with incoming messages will not have outgoing messages. Thus, each node, except the initial node, is responsible for at least one one-way edge. Therefore, the total number of messages is at most $2dn-(n-1) = (2d-1)n+1$.

□

2.8.2 Broadcast along a Cycle

In some situations, we do not need to broadcast under a tight time constraint but would like to minimize the total number of messages sent.

We first present a generic algorithm for walking on a Hamilton cycle of an $\mathbb{H}$-graph. We note that in most random graph models, finding a Hamilton cycle is nontrivial.

```
CYCLEWALK(PROC, depth t, direction i, nonce h)
1  if h is seen recently
2     then return
3  if PROC() and t > 0
4     then ngbr_i ⇒ CYCLEWALK(PROC, t − 1, i, h)
```

```
BROADCAST-CYCLEWALK(PROC)
1  i ← UNIFORM({1, ..., d})
2  h ← NEW-NONCE()
3  CYCLEWALK(PROC, ∞, i, h)
4  CYCLEWALK(PROC, ∞, −i, h)
```

The reason that we have a separate procedure CYCLEWALK and a depth parameter t is that we will reuse CYCLEWALK with finite depth in Section 2.9 and Chapter 3.

Theorem 14. *Procedure* BROADCAST-CYCLEWALK *terminates in* $\lceil n/2 \rceil$ *steps and sends at most* $n+1$ *messages.*

Proof. After $\lceil n/2 \rceil$ steps, the broadcasts along the two directions will meet. Number of messages sent is at most $2\lceil n/2 \rceil \leq n+1$.

□

2.9 Searches

In this section we analyze the search algorithms on the $\mathbb{H}$-graphs. These algorithms can be used to implement a discovery service as proposed in Section 5.1. Although such algorithms can run on almost any peer-to-peer networks, we can obtain good theoretical performance bounds on $\mathbb{H}$-graphs.

Starting at any node, we would like to search for some node satisfying certain properties that can be expressed as a Boolean function $\phi : G \mapsto \{\mathsf{true}, \mathsf{false}\}$. We call ϕ a *criterion function.* Starting from an arbitrary node, a search algorithm has to look for a node v such that $\phi(v) = \mathsf{true}$. For example, we can have ϕ' so that $\phi'(v)$ is true if and only if v is a Freenet server with at least 128 kbps bandwidth and 10 GB storage. We assume that the criterion function can be described in $O(1)$ space and be evaluated in $O(1)$ time. Sometimes the evaluation of ϕ might require privacy—Fagin, Naor, and Winkler [25] discussed how we can compare information without leaking it.

For any criterion function ϕ, let

$$\phi[G] = \{u \in G \mid \phi(u) = \mathsf{true}\}$$

be the *target set.* We express the performance of a search algorithm in terms of a parameter $\psi = \psi_\phi(G) = |\phi[G]| \,/\, |G|$, which is the proportion of the nodes in G satisfying ϕ. We call ψ the *density* of ϕ on G. In most scenarios, we expect that ψ can be lower-bounded by a constant. In other words, when an $\mathbb{H}$-graph grows, the number of nodes satisfying criterion ϕ is expected to increase proportionally.

Let $A = \phi[G]$ be the target set. Our search algorithms to be presented will try to return any node in A. We assume that another specific protocol can take over once a node in A

is found. We note that the algorithms in this section can be easily modified to discover as many nodes in A as possible given certain time and message constraints.

If we need a complicated function ϕ, it can be implemented in a portable language. If simple rule-based matching is sufficient, ϕ can be implemented in an XML query language to access meta-data stored as Resource Description Framework (RDF) documents at the nodes.

Effectively, our algorithms perform on-demand searching, instead of indexing at a centralized server. With sufficient hardware support, indexing could produce better response time. However, the index cannot be refreshed very frequently and thus usually contains obsolete entries. Moreover, a criterion function is much more flexible than indexes.

We will consider two models for the probability distribution of the target set A in G. First, we can assume that set A is distributed independently from the randomized construction of the $\mathbb{H}$-graph G and the initial node u. This means that the selection of A is independent of the random events that lead to the particular topology of G and the choice of initial node u. We call such A an 'independent target set'. Alternatively, we can assume a worst case scenario where set A is picked by an adversary after G is constructed and node initial node u is selected. We call such set A an 'adversarial target set'. We will present algorithms SEARCH-CYCLEWALK and SEARCH-TREEWALK for the independent target set and algorithm SEARCH-RANDOMWALK for the adversarial target set.

Before we describe the search algorithms, we need to introduce a function NEW for initializing a procedure object. The expression

$$\text{NEW}(\text{PROC}, \{x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k\})$$

creates a new instance of PROC such that local variables $x_1, \ldots, x_k$ are initialized to the values $v_1, \ldots, v_k$. These values are stored within the procedure object PROC.

SEARCH-PROC encapsulates the criterion function ϕ and remembers the node s that initiated the search. It will be used in the our search algorithms.

```
SEARCH-PROC()
1  locals  φ, s
2  if φ(self) = true
3     then s ⇒ SEARCH-REPLY(self)
4          return false
5     else return true
```

```
SEARCH-REPLY(v)
1  R ← R ∪ v
```

Procedure SEARCH-PROC returns true when the search should be continued. When a node satisfying the criterion function is found, it notifies the initial node, and then returns false to signal the end of the search.

2.9.1 Search by Cycle Walk

As a warm-up, we start with a simple search procedure SEARCH-CYCLEWALK, which picks a Hamilton cycle randomly and walks along that cycle. We assume an independent target set for the complexity analyses.

```
SEARCH-CYCLEWALK(φ, t)
1  R ← ∅
2  P ← NEW (SEARCH-PROC, {φ ← φ, s ← self})
3  CYCLEWALK(P, t, UNIFORM({−1, 1, −2, 2, ..., −d, d}), NEW-NONCE())
4  wait for t steps or until R is nonempty
5  return R
```

Lemma 15. *Consider two sets $S, T \subseteq G$, where S is randomly distributed over G, then S and T are disjoint with probability at most $e^{-|S||T|/n}$.*

Proof. Let $S = \{s_1, s_2, \ldots, s_k\}$. The probability that $T \cap S = \emptyset$ is

$$\begin{aligned}
&\Pr\{s_1 \notin T\} \Pr\{s_2 \notin T \mid s_1 \notin T\} \cdots \Pr\{s_k \notin T \mid \{s_1, \ldots, s_{k-1}\} \cap T = \emptyset\} \\
&= (1 - |T|/n)(1 - |T|/(n-1)) \cdots (1 - |T|/(n-(k-1))) \\
&< (1 - |T|/n)^{|S|} \\
&< e^{-|S||T|/n}.
\end{aligned}$$

□

Theorem 16. *Given an $\mathbb{H}$-graph G constructed by our protocol and a criterion function ϕ, the probability that* SEARCH-CYCLEWALK(ϕ, t) *fails to find a node in $\phi[G]$ is at most $e^{-t\psi_\phi(G)}$.*

Proof. After t steps, the algorithm should have reached a set of t nodes. The probability that $\phi[G]$ does not intersect with this set is at most $e^{-t\psi_\phi(G)}$ by Lemma 15. □

We can use SEARCH-CYCLEWALK(ϕ, ∞) if we are confident that the density of ϕ is not too small.

Theorem 17. *Given an $\mathbb{H}$-graph G constructed by our protocol and a criterion function ϕ, the expected time complexity and message complexity of* SEARCH-CYCLEWALK(ϕ, ∞) *is $(\psi_\phi(G))^{-1}$.*

Proof. At each step the probability that the algorithm reaches a node in $\phi[G]$ is at least $\psi_\phi(G)$, thus the expected number of steps required is at most $(\psi_\phi(G))^{-1}$. □

2.9.2 Search by Tree Walk

Next, we will consider a search algorithm that is faster but also sends more messages.

```
SEARCH-TREEWALK(φ, t, q)
1  R ← ∅
2  P ← NEW (SEARCH-PROC, {φ ← φ, s ← self})
3  TREEWALK(P, t, q, NULL, NEW-NONCE())
4  wait for t steps or until R is nonempty
5  return R
```

Theorem 18. *Let G be a $2d$-regular $\mathbb{H}$-graph generated by our protocol and let ϕ be a criterion function. Let $n = |G|$ and $\psi = \psi_\phi(G)$. With high probability,*

$$\log_{2(d-1)} \frac{4 \ln n}{\psi} + O\left(\frac{\log n}{\psi n}\right) = O\left(\log \log n + \log \psi^{-1}\right)$$

steps are sufficient to find a node in $\phi[G]$.

Proof. Let X_t be the set of nodes reached by SEARCH-TREEWALK(ϕ, t, ∞) after t steps. By Lemma 15, if

$$|X_t| \geq \ln n / \psi,$$

the probability that X_t and A do not intersect is at most $1/n$. By Theorem 11, this condition is equivalent to

$$\frac{n}{1 + 4\frac{n-1}{(2(d-1))^t}} \geq \frac{\ln n}{\psi},$$

which can be rewritten as

$$\begin{aligned} t &\geq \log_{2(d-1)} \frac{4(1 - \frac{1}{n})}{\frac{\psi}{\ln n} - \frac{1}{n}} \\ &= \log_{2(d-1)} \frac{4 \ln n}{\psi} + O\left(\frac{\log n}{\psi n}\right). \end{aligned}$$

□

Lemma 19. *Let G be a $2d$-regular $\mathbb{H}$-graph generated by our protocol and let ϕ be a criterion function. The total number of messages sent by* SEARCH-TREEWALK(ϕ, t, ∞) *is at most*

$$\left(\frac{4 \ln n}{\psi}\right)^{\log_{2d-2} 2d-1} \left(1 + \frac{1}{2d2} + O\left(\frac{\log n}{d\psi n}\right)\right) = O\left(\psi^{-1} \log n\right)$$

when $t = \log_{2(d-1)} \frac{4 \ln n}{\psi} + O\left(\frac{\ln n}{\psi n}\right)$.

Proof. Let $\psi = \psi_\phi(G)$. In the worst case, SEARCH-TREEWALK(ϕ, t, ∞) will spawn a tree of messages with depth t and branching factor $2d-1$, except at the root, which can has $2d$ children. Therefore, the total number of messages sent is at most

$$\sum_{i=0}^{t}(2d-1)^i = \frac{(2d-1)^{t+1}-1}{2d-2}$$
$$< \left(\frac{4\ln n}{\psi}\right)^{\log_{2d-2} 2d-1}\left(1+\frac{1}{2d-2}+O\left(\frac{\log n}{d\psi n}\right)\right).$$

□

2.9.3 Search by Random Walk

We now proceed to show that random walks can help us achieve good search performance even when the target set is selected by an adversary.

```
SEARCH-RANDOMWALK(φ, t)
1  if φ(self) = true
2     then return self
3     else if t > 0
4             then return UNIFORM(N(self)) ⇒
5                     SEARCH-RANDOMWALK(φ, t − 1)
```

Theorem 20. *Consider a k-regular graph G and a criterion function ϕ chosen by an on-line adversary. Let $\psi = \psi_\phi(G)$.*

The expected number of steps required by SEARCH-RANDOMWALK(ϕ, ∞) *on G is at most*

$$\frac{1}{1-\lambda(G)/k}\left(\frac{1}{\psi}+\frac{1}{2}\ln\left((n-1)\frac{1-\psi}{\psi}\right)\right) = O(\psi^{-1}+\log n).$$

Proof. A random walk on G of size n can be considered as an n-state discrete-time Markov chain such that each node in G corresponds to a state in the chain. Let X_t be the state of a random walk, where X_0 is the initial state. The transition probability $\Pr\{X_t = j \mid X_{t-1} = i\}$ is the number of edges from node i to node j divided by k (G is a regular multi-graph). Let $A = \phi[G]$. Following the terminology of [5, 6], let T_A be the hitting time on set A:

$$T_A = \min\{t \geq 0 \mid X_t \in A\}.$$

Let π be the stationary distribution of the Markov chain. In other words, let $\pi(Y)$ be the probability that $X_t \in Y$ when $t \to \infty$. Since G is regular, π is the uniform distribution. Let $\mathrm{E}_i[T_A]$ be the expected value of T_A given initial state $X_0 = i$. Let $\mathrm{E}_\pi[T_A]$ be the expected value of T_A given a uniform initial distribution, i.e., $\Pr\{X_0 = i\} = 1/n$ for all $i \in G$.

Let $\tau = \frac{1}{1-\lambda(G)/k}$. According to Corollary 8 of [5],

$$\mathrm{E}_i[T_A] \leq \mathrm{E}_\pi[T_A] + \tau + \frac{1}{2}\tau \ln \frac{1-\pi(i)}{1/\pi(i)} \frac{\mathrm{E}_\pi[T_A]}{\tau}. \tag{2.12}$$

for any $i \in G$.

By Lemma 2 of [5], we have

$$\mathrm{E}_\pi[T_A] \leq \frac{\tau\pi(\overline{A})}{\pi(A)} = \frac{\tau(1-\psi)}{\psi}. \tag{2.13}$$

Substituting Equation (2.13) into Equation (2.12), we have

$$\mathrm{E}_i[T_A] \leq \tau\left(1/\psi + \frac{1}{2}\ln\left((n-1)\frac{1-\psi}{\psi}\right)\right).$$

□

2.10 Network Size Estimation

In a distributed environment, any node in a network may not easily know the total size of the network. However, many of our algorithms, such as sampling by broadcasts (Section 2.4.2) and sampling by random walks (Section 2.5) would run more efficiently if a good estimation of the network size is available. Therefore, we would like to have an efficient distributed algorithm to gather and disseminate estimates of the network size.

Cohen [20] introduced a framework for size estimation and discussed techniques for parallelization. However, we are not aware of any work on a distributed version of the framework.

Estimating the size of a network graph distributedly is an interesting algorithm problem. Our goal is for every node to have an estimate that is close to the graph size. Since many algorithms such as JOIN-RW have time complexity $O(\log n)$, if the estimate of the graph size is n^μ, $\mu \geq 1$, then the complexity is increased by a factor of μ. In the following, we outline several approaches.

Server A simple solution is to use a global server to store the graph size. Whenever a node joins or leaves the graph, the server is notified. The storage requirement at the server is only $O(\log n)$. The number of messages sent to and from the server is only $O(1)$ per JOIN or LEAVE.

Broadcast Broadcasts can count the number of nodes in the graph. To avoid messaging bottleneck at the root, we can accumulate the results from the leaves back to the root along the broadcast tree. The message complexity is $O(dn)$ and time complexity is $O(\log_d n)$. However, some protocol is required to decide which node should be the broadcast initiator. Possible schemes include passing tokens and random initiators.

Random Walks Feige [27] gave a randomized LOGSPACE algorithm COMPONENT for testing graph connectivity. We observe that COMPONENT can also be used to estimate the size of an arbitrary connected graph in $O(n^3)$ time and a regular graph in $O(n^2)$ time.

For expander graphs, Gillman [38]'s Chernoff bound for random walks can be employed to obtain a smaller time complexity. Adapting Gillman's modified Aldous's procedure [4], we can show that, in $O(\beta^{-2} n \log n)$ steps, a node can estimate the size of the graph within error βn with high probability.

Gillman derived bounds for estimating $\pi(A)$ with random walks, where A a subset of the graph, and π is the stationary distribution.

In expander graphs, both algorithms of Feige and Gillman are based on random walks, so they can be easily applied in a distributed setting. In the following, we will give a sketch on applying the Gillman's bounds for estimating the size of a graph.

Theorem 21. *Using the modified Aldous's procedure, the cost to estimate the size of a regular graph G to within βn with probability $1 - n^{-\Theta(1)}$ is at most $O\left(\frac{n \log n}{\beta^2(1-\rho(G))}\right)$.*

Proof. From Corollary 3.3 of [38], procedure APM can estimate $\pi(A)$ to within $\alpha\pi(A)$ with probability at least $1 - \delta$ in

$$O\left(\frac{1}{1-\rho(G)} \log \frac{1}{\delta} \left(\log \frac{1}{\pi(s)} + \frac{1}{\alpha^2 \pi(A)}\right)\right)$$

random walk steps.

We let $A = \{s\}$ where s is an arbitrary node, thus $n = 1/\pi(A) = 1/\pi(s)$. Let t be an estimate of $\pi(s)$. To estimate n to within error βn, we need to have

$$(1 - \beta)n < 1/t < (1 + \beta)n$$

It's sufficient if we have

$$(1 - \beta/2)/n < t < (1 + \beta/2)/n$$

Setting $\alpha = \beta/2$ and $\delta = n^{-\Theta(1)}$, the time complexity required by the modified Aldous's

procedure is

$$
\begin{aligned}
&O\left(\frac{1}{1-\rho(G)}\log\frac{1}{\delta}\left(\log\frac{1}{\pi(s)}+\frac{1}{\alpha^2\pi(A)}\right)\right)\\
&=O\left(\frac{1}{1-\rho(G)}\log n\left(\log n+\frac{n}{\beta^2}\right)\right)\\
&=O\left(\frac{n\log n}{\beta^2(1-\rho(G))}\right).
\end{aligned}
$$

□

2.10.1 Size Estimation by Mobile Agents

In the rest of this section, we study a new distributed protocol for network size estimation. We consider mobile agents (*walkers*) that move from node to node on the network. Walkers are also called 'pebbles' or 'agents' in some literature. A walker counts the number of nodes that it has visited by marking the nodes. Whenever it visits an unmarked node, its counter increments. The walker also notifies the nodes with its current estimate. The markings, estimates, and the walkers themselves should all have expiry times. There can be multiple walkers so that the nodes can take the maximum of all the non-expired estimates.

A natural strategy for the walker is to walk randomly. However, for our $\mathbb{H}$-graphs, a walker can choose to walk on the Hamilton cycles. Walking on Hamilton cycle is simpler but is weaker against adversarial attacks. We will analyze both options.

2.10.2 Cycle Walks

We first analyze the effectiveness of a cycle walker. In the following, we assume that walkers are created at arbitrary nodes, which can be chosen by an adversary.

In order to quantify the performance of such algorithm on a dynamic network, certain assumptions must be made on the rates node arrivals and departures. For the rest of this section, time complexity is normalized such that it takes a unit time for a single walk step (moving from one node to another). We assume a maximum arrival rate α such that during any time interval $\tau \geq \tau_a$, the number of node arrivals is at most $\alpha\tau$. We also assume maximum departure rate δ such that during any time interval $\tau \geq \tau_l$, the number of node departures is at most $\delta\tau n$, where n is the maximum size of the network during this time interval.

Theorem 22. *Let the maximum arrival rate be $\alpha < 1$. A cycle-walker is created on a size-n Hamiltonian graph G at time 0. At time $(1+x)n$ where $x > \alpha/(1-\alpha)$, every node $v \in G$ has received an estimate* $\mathsf{est}(v)$ *such that*

$$\min \mathsf{est}(v) > \frac{x-\alpha(1+x)}{1+\alpha(1+x)}n',$$

where n' is the size of the graph at time $(1+x)n$.

Proof. First, after time yn, the walker's counter is yn (the number of nodes the walker has visited). At time yn, the size of the network is at most $(1+y\alpha)n$. We let the walker complete a round trip on the Hamilton cycle after this point. At the same time, more nodes can arrive at the network. Since the total time taken by the walker is $(1+x)n$, the time required to complete a cycle after time yn is $(1+x-y)n$. In the worst case, the size of the graph could become as big as $(1+\alpha(1+x))n$. Therefore, we can pick y such that

$$(1+x-y)n = (1+\alpha(1+x))n$$

and still have sufficient time to complete the round trip on the Hamilton cycle. Thus we have

$$y = x - \alpha(1+x)$$

Since we need $y > 0$, we must have $\alpha < 1$ and $x > \alpha/(1-\alpha)$. Thus, $\min_{v \in G} \text{est}(v) \geq yn = (x - \alpha(1+x))n$. Since $n' \leq (1+(1+x)\alpha)n$, we have

$$\min \text{est}(v) \geq \frac{x-\alpha(1+x)}{1+(1+x)\alpha} n'.$$

□

The major disadvantage of the cycle walk algorithm is a fault tolerance concern. Since there is only one option for the walker at each step, if the next node on the cycle is slow or unavailable, then the walker will be stalled. On the other hand, a random walker to be considered in the next section can simply pick another responsive neighbor.

2.10.3 Random Walks

In this section we analyze random walkers on static networks. We show that after sufficient time, a walker can obtain an estimate which is a fraction of the graph size.

We will analyze the random walker in two phases. In phase I, the random walkers counts a constant fraction of the total network. In phase II, the random walkers distribute the estimates to all the nodes in the network. We note that the division of the two phases are used for the convenience of the analysis only. The walkers and the nodes do not need to be aware of the phases.

Let H be the binary entropy function:

$$H(x) \equiv x \log_2 x^{-1} + (1-x)\log_2 (1-x)^{-1}.$$

First, we show that the random-walker can reach a certain fraction of the graph in phase I.

Lemma 23. *Consider a random-walker on graph G of n nodes. Let κ be a real number between 0 and 1. Let $a = \rho(G) + \kappa - \rho(G)\kappa$ and $\phi \geq \frac{H(\kappa)+1}{\ln a^{-1}}$. After $\phi n + 1$ steps, the walker has reached at least κn nodes with probability at least $1 - \sqrt{n}2^{-n}$.*

Proof. Consider any $\kappa \in (0,1)$. Kahale [51] has shown that the probability that a random walk of length t stays inside a set A of size κn is at most

$$\sqrt{\pi(A) \sum_{v \in A} q(v)^2/\pi(v) a^{t-1}}$$

where q is the distribution of the initial position of the walk. Since $\pi(A) = \kappa$, $\pi(v) = 1/n$, and $q(v) = 1$ only if v is the starting node, we have

$$\Pr\{\text{walk stays in } A\} \leq \sqrt{\kappa n} a^{t-1}$$

Since there are at most $\binom{n}{\kappa n}$ choices for the set A,

$$\begin{aligned} &\Pr\{\text{walk stays in a set of size } \kappa n\} \\ &\leq \binom{n}{\kappa n} \sqrt{\kappa n} a^{t-1} \\ &\leq 2^{nH(\kappa)} \sqrt{\kappa n} a^{t-1} \\ &= \sqrt{n} 2^{nH(\kappa) + (t-1)\log_2 a}. \end{aligned} \tag{2.14}$$

Thus, if we choose $t = \phi n + 1$ where

$$\phi = \frac{H(\kappa) + 1}{\log_2 a^{-1}},$$

then

$$\Pr\{\text{walk stays in a set of size } \kappa n\} \leq \sqrt{n} 2^{-n}.$$

□

We need a technical lemma on the distribution of the position of random walker after phase I.

Lemma 24. *Let $\mathbf{r}$ be the distribution of a random walk on a graph G after t steps, then*

$$\sum_{v \in G} \frac{r(v)^2}{\pi(v)} \leq \rho(G)^t (n-1) + 1.$$

Proof. By Fill [28],

$$\sum_{v \in G} \frac{(p_t(v) - \pi(v))^2}{\pi(v)} \leq \rho(G)^t \sum_{v \in G} \frac{(p_0(v) - \pi(v))^2}{\pi(v)},$$

where $\mathbf{p_0}$ is the initial distribution of the walk. Since the random walk starts at an arbitrary position, we have

$$\begin{aligned}
&\sum_{v \in G} \frac{(p_0(v) - \pi(v))^2}{\pi(v)} \\
&= (n-1)\frac{1}{n} + n\left(1 - \frac{1}{n}\right)^2 \\
&= n - 1.
\end{aligned}$$

Let $\mathbf{r} = \mathbf{p}_t$, then

$$\begin{aligned}
&\sum_{v \in G} \frac{(r(v) - \pi(v))^2}{\pi(v)} \\
&= \sum_{v \in G} \frac{r(v)^2}{\pi(v)} + \sum_{v \in G} \pi(v) - 2\sum_{v \in G} r(v) \\
&= \sum_{v \in G} \frac{r(v)^2}{\pi(v)} - 1.
\end{aligned}$$

Therefore,

$$\sum_{v \in G} \frac{r(v)^2}{\pi(v)} \leq \rho(G)^t (n-1) + 1.$$

□

Next, we give a bound on the probability that a random walker fails to reach a certain node in phase II.

Lemma 25. *Let t and l be the lengths of phases I and II respectively. Consider a regular graph G of size n. Any node in G is not reached by a walker during phase II with probability at most*

$$\sqrt{\rho(G)^t(n-1)+1}\,(1 - (1 - \rho(G))/n)^{l-1}.$$

Proof. Let $\rho = \rho(G)$. Let v be any node in G. Let $A = G \setminus v$, and thus $\pi(A) = 1 - 1/n$.

Consider a random walker that walks for t steps in phase I and l steps in phase II.

$$\begin{aligned}&\Pr\{\text{node } v \text{ not reached by the walker in phase II}\}\\&= \Pr\{\text{walker does not start at } v \text{ in phase II}\}\\&\quad \cdot \Pr\{\text{walker stays in } A \text{ in phase II}\}.\end{aligned}$$

Let $\mathbf{r}$ be the distribution of the walker at the end of phase I. Let $\mathbf{q}$ be the distribution of the walker in A given that the walker is in A at the end of phase I.

For any $u \in A$,

$$\begin{aligned}r(u) &= q(u)\Pr\{\text{walk ends in } A \text{ in phase I}\}\\&= q(u)(1-r(v)).\end{aligned}$$

Thus,

$$\sum_{u\in A} q(u)^2 = \sum_{u\in A} \frac{r(u)^2}{(1-r(v))^2}.$$

Let $a = \rho + \pi(A) - \rho\pi(A) = 1 - (1-\rho)/n$. By Kahale [51],

$$\begin{aligned}&\Pr\{\text{walk does not reach node } v \text{ during phase II}\}\\&= \Pr\{\text{walk does not start at } v\}\Pr\{\text{walker remains in } A\}\\&\le (1-r(v))\sqrt{\pi(A)\sum_{u\in A} q(u)^2/\pi(u)a^{l-1}}\\&= \sqrt{\pi(A)\sum_{u\in A} r(u)^2/\pi(u)a^{l-1}}\\&\le \sqrt{\rho^t(n-1)+1}\,(1-(1-\rho)/n)^{l-1},\end{aligned}$$

where the last inequality follows from Lemma 24. □

The following theorem shows that after $O(n)$ steps, each node in the graph will obtain a size estimate of $\Theta(n)$ if there are $\Theta(\log n)$ random walkers.

Theorem 26. *Let G be a graph of size n. Let there be $\mu \ln n$ walkers. Let $\kappa \in (0,1)$. Let $\rho = \rho(G)$. Let $a = \rho + \kappa - \rho\kappa$. The probability that all nodes in G having received an estimate of at least κn in time $\left(\frac{H(\kappa)+1}{\log_2 a^{-1}} + \frac{2}{\mu(1-\rho)}\right) n + 2$ is at least $1 - \frac{1}{n}(1+O(\epsilon^n))$ for some $\epsilon < 1$.*

Proof. Let $\phi = \frac{H(\kappa)+1}{\log_2 a^{-1}}$. Let $\psi = 2/(\mu(1-\rho))$. Let $t = \phi n + 1$ and $l = \psi n + 1$ be the lengths of phases I and II.

By Lemma 25, we have

$$\begin{aligned}
&\Pr\{\text{a walk fails to reach } v \text{ in phase II}\} \\
&\leq \sqrt{\rho^t(n-1)+1}\,(1-(1-\rho)/n)^{l-1} \\
&\leq e^{\rho^t(n-1)/2}\,(1-(1-\rho)/n)^{l-1}. \\
&\leq e^{\rho^t(n-1)/2-\psi(1-\rho)}.
\end{aligned}$$

Then

$$\begin{aligned}
&\Pr\{\text{any node not receiving an estimate of at least } \kappa n\} \\
&\leq n \prod_{v \in G} \Pr\{\text{all walkers with an estimate at least } \kappa n \text{ do not reach } v \text{ in phase II}\} \\
&\leq n(\Pr\{\text{a walker fails to obtain an estimate of at least } \kappa n \text{ in phase I}\} \\
&\quad + \Pr\{\text{a walker fails to reach a node in phase II}\})^{\mu \ln n} \\
&= n\left(\sqrt{n}2^{-n} + e^{\rho^t(n-1)/2-\psi(1-\rho)}\right)^{\mu \ln n}.
\end{aligned}$$

Since $t \geq cn$ for some constant c, we have

$$\rho^t(n-1)/2 = O(\epsilon^n)$$

and

$$\sqrt{n}2^{-n} = O(\epsilon^n)$$

for some $\epsilon < 1$.

Thus,

$$\begin{aligned}
&n\left(\sqrt{n}2^{-n} + e^{\rho^t(n-1)/2-\psi(1-\rho)}\right)^{\mu \ln n} \\
&= n\left(O(\epsilon^n) + e^{O(\epsilon^n)-\psi(1-\rho)}\right)^{\mu \ln n} \\
&\leq n e^{O(\epsilon_1^n)-\mu\psi(1-\rho)\ln n}\left(1 + \frac{O(\epsilon^n)}{e^{O(\epsilon^n)-\psi(1-\rho)}}\right)^{\mu \ln n} \\
&= n n^{-\mu\psi(1-\rho)}\,(1+O(\epsilon_2^n)) \\
&= \frac{1}{n}\,(1+O(\epsilon_2^n)),
\end{aligned}$$

because $\mu\psi(1-\rho) = 2$.

□

Example 1. Consider an H-graph G of size $n = 100000$ with $d = 8$. If we have $\lceil 50 \ln n \rceil = 576$ random walkers, after 15000 time steps, all nodes have received an estimate of 1000 with probability at least 0.999929.

Proof. Let $\kappa = 0.01$. First, we have $\rho(G) \leq 0.48413$ with high probability. Thus $a = \rho + \kappa - \rho\kappa \leq 0.4893$. We can easily derive the following parameters:

$$\begin{aligned} H(\kappa) &< 0.080793136 \\ nH(\kappa) &< 8079.3136 \\ \mu &= 50 \\ t &= 7900 \\ l &= 7100 \\ (t-1)\log_2 a &\leq -8147 \end{aligned}$$

The total time required is $t + l = 15000$.
According to Equation (2.14) in the proof of Lemma 23,

$$\begin{aligned} \Pr\{\text{walk stays in a set of size } \kappa n\} &\leq \sqrt{\kappa n} 2^{nH(\kappa)+(t-1)\log_2 a} \\ &< \sqrt{10002}^{8080-8147} \\ &< 2.15 \times 10^{-19}, \end{aligned}$$

and by Lemma 25,

$$\begin{aligned} &\Pr\{\text{a walk fails to reach } v \text{ in phase II}\} \\ &\leq \sqrt{\rho^t(n-1)+1}\,(1-(1-\rho)/n)^{l-1} \\ &< 0.964045549. \end{aligned}$$

Therefore,

$$\begin{aligned} &\Pr\{\text{any node not receiving an estimate of at least } \kappa n\} \\ &= n\left(2.15 \times 10^{-19} + 0.964045549\right)^{\mu \ln n} \\ &< 100000\left(2.15 \times 10^{-19} + 0.964045549\right)^{50 \ln 100000} \\ &< 0.0000701158. \end{aligned}$$

□

In Theorem 26, there are three selectable parameters:

- the minimum estimate guarantee: κn,

- total time required: $t + l = (\phi + \psi)n + 2$,
- number of walkers: $\mu \ln n$.

Thus, this is a three-way trade-off among the quality of the estimate, the time complexity, and the message complexity.

Although this scheme has higher overhead than other methods discussed above, it is more robust and also easier to implement. In Example 1, our calculation predicts that 576 random walkers are required to achieve an estimate of 1000 in 15000 time steps. For fault tolerance, we can simply dispatch 1000 walkers (each node can choose to spawn one with probably 0.01) and prepare for the lost of many of them.

2.11 On-line Fault Tolerance

We have seen that the $\mathbb{H}$-graphs can maintain small second largest eigenvalues against an off-line adversary selecting the sequence of nodes joining and leaving. It is easy to see that even an on-line adversary cannot do much harm by the selection of joining nodes. However, it has much larger power when if it can choose the set of nodes to leave given the current topology. We now investigate the effect of having such adversary choosing a leaving set.

Let $E_l(G)$ be the set of adjacent nodes on the level-l Hamilton cycle of an $\mathbb{H}$-graph G.

The sum $\sum_{1 \le l \le d} \sum_{\{u,v\} \in E_l(G)} (f(u) - f(v))^2$, where f is a function that assigns each node $v \in G$ a real value $f(v)$, is usually called the Dirichlet sum. It is known that the gap between $2d$ and $\lambda(G)$ can be expressed as a Raleigh quotient [19]:

$$2d - \lambda(G) = \inf_{\sum_{u \in G} f(u) = 0} \frac{\sum_{1 \le l \le d} \sum_{\{u,v\} \in E_l(G)} (f(u) - f(v))^2}{\sum_{u \in G} f(u)^2}. \tag{2.15}$$

Given a nonempty subset $A \subset G \in \mathbb{H}_{n,2d}$, let

$$t_G(A) \equiv \sup_{u \in G \setminus A} \sum_{1 \le l \le d} |\{ v \in A \mid \{u, v\} \in E_l(G) \}|.$$

The value $t_G(A)$ is the maximum number of nodes in A that is next to any node in $G \setminus A$. Since $|G| \ge 3$, we have $1 \le t_G(A) \le 2d$. Theorem 27 shows that $t_G(A)$ can be used to bound the change of $\lambda(G)$ when a set A leaves an $\mathbb{H}$-graph G.

Theorem 27. *Given an $\mathbb{H}$-graph G and a subset $A \subset G$, if G_A is obtained by removing A from G by operation* LEAVE, *then*

$$\lambda(G_A) \le \lambda(G) + t_G(A).$$

Proof. Consider any function f on G_A such that $\sum_{u \in G_A} f(u) = 0$. We can extend f to G so that $f(u) = 0$ for any $u \in A$. Therefore, we have $\sum_{u \in G} f(u) = 0$ and $\sum_{u \in G} f(u)^2 = \sum_{u \in G_A} f(u)^2$.

Let $G \in \mathbb{H}_{n,2d}$. For each $l = 1, \ldots, d$, consider the pairs

$$(a_{l,1}, b_{l,1}), (a_{l,2}, b_{l,2}), \ldots, (a_{l,h_l}, b_{l,h_l}),$$

such that for each pair $i = 1, \ldots, h_l$, there is a sequence of nodes $u_1, \ldots, u_k$ on the level-l cycle where

- $u_1 = a_{l,i}$,
- $u_k = b_{l,i}$,
- $u_j \in A$ for $j = 2, \ldots, k-1$,
- $u_j.\mathsf{ngbr}_l = u_{j+1}$ for $j = 1, \ldots, k-1$.

After the nodes between $a_{l,i}$ and $b_{l,i}$ have left from G, $a_{l,i}$ and $b_{l,i}$ become neighbors in G_A. Thus,

$$\begin{aligned}
&\left(\sum_{j=1}^{k-1}(f(u_{j+1}) - f(u_j))^2\right) - (f(u_k) - f(u_1))^2 \\
&= f(u_k)^2 + f(u_1)^2 - (f(u_k) - f(u_1))^2 \\
&= 2f(u_k)f(u_1) \\
&= 2f(a_{l,i})f(b_{l,i}).
\end{aligned}$$

because $f(u_j) = 0$ for $j = 2, \ldots, k-1$.

Summing over the nodes on the level-l cycle, we have

$$\sum_{\{u,v\} \in E_l(G)} (f(u) - f(v))^2 - \sum_{\{u,v\} \in E_l(G_A)} (f(u) - f(v))^2 = \sum_{1 \le i \le h_l} 2f(a_{l,i})f(b_{l,i}).$$

Summing over all cycles $l = 1, \ldots, d$,

$$\sum_{\substack{\{u,v\} \in E_l(G) \\ 1 \le l \le d}} (f(u) - f(v))^2 - \sum_{\substack{\{u,v\} \in E_l(G_A) \\ 1 \le l \le d}} (f(u) - f(v))^2 = \sum_{1 \le l \le d} \sum_{1 \le i \le h_l} 2f(a_{l,i})f(b_{l,i}). \quad (2.16)$$

Each node in $G \setminus A$ appears in at most $t_G(A)$ terms in the summation

$$\sum_{1 \le l \le d} \sum_{1 \le i \le h_l} 2f(a_{l,i})f(b_{l,i}).$$

Therefore, $\sum_{1 \le i \le h_l} 2f(a_{l,i})f(b_{l,i})$ can be separated into at most $t_G(A)$ summations such that each node in $G \setminus A$ appears at most once in each summation. For any such summation, we have

$$\sum_i 2f(u_i)f(v_i) \le \sum_i f(u_i)^2 + f(v_i)^2 \le \sum_{w \in G \setminus A} f(w)^2.$$

Combining these $t_G(A)$ summations, we have

$$\sum_{1 \le i \le d} \sum_{1 \le i \le h_l} 2f(a_{l,i})f(b_{l,i}) \le t_G(A) \sum_{v \in G_A} f(v)^2.$$

Substituting back into Equation (2.16) gives

$$\sum_{\substack{\{u,v\} \in E_l(G_A) \\ 1 \le l \le d}} (f(u) - f(v))^2 \ge \sum_{\substack{\{u,v\} \in E_l(G) \\ 1 \le l \le d}} (f(u) - f(v))^2 - t_G(A) \sum_{v \in G_A} f(v)^2.$$

Now, let f be the function attaining the infimum for $2d - \lambda(G_A)$ in Equation (2.15),

$$\begin{aligned}
&2d - \lambda(G_A) \\
&= \frac{\displaystyle\sum_{\substack{\{u,v\} \in E_l(G_A) \\ 1 \le l \le d}} (f(u) - f(v))^2}{\sum_{v \in G_A} f(v)^2} \\
&= \frac{1}{\sum_{v \in G} f(v)^2} \sum_{\substack{\{u,v\} \in E_l(G_A) \\ 1 \le l \le d}} (f(u) - f(v))^2 \\
&\ge \frac{1}{\sum_{v \in G} f(v)^2} \left(\sum_{\substack{\{u,v\} \in E_l(G) \\ 1 \le l \le d}} (f(u) - f(v))^2 - t_G(A) \sum_{v \in G_A} f(v)^2 \right) \\
&\ge (2d - \lambda(G)) - t_G(A).
\end{aligned}$$

Therefore, we have

$$\lambda(G_A) \le \lambda(G) + t_G(A).$$

□

From Theorem 27, we can see that if $t_G(A)$ is small, then removing set A from G has small effect on the second largest eigenvalue. Indeed, if A is not large and is chosen randomly, we can expect that $t_G(A)$ is likely to be 1.

Theorem 28. *If a subset $A \subset G$ is selected randomly, where $|A| < \frac{3|G|}{2(2d-1)} + 1/2$, then $t_G(A) > 1$ with probability at most*

$$\frac{2d^2\,|A|\,(|A|-1)}{|G|}.$$

Proof. Let $n = |G|$. Let set A be $\{u_1, u_2, \ldots, u_k\}$. Since A is a random subset of G, we can assume that the nodes $u_1, u_2, \ldots, u_k$ are chosen sequentially from G at random without replacement.

Let $N(A)$ be the set of neighbors of A in G. Let T_i be the event that $t_G(\{u_1, \ldots, u_i\}) = 1$. Given T_i, if node u_{i+1} is not in $N(N(\{u_1, \ldots, u_i\})) \setminus \{u_1, \ldots, u_i\}$, then none of the nodes in $G \setminus \{u_1, \ldots, u_{i+1}\}$ have two neighbors in $\{u_1, \ldots, u_{i+1}\}$. Since the set $N(N(\{u_1, \ldots, u_i\})) \setminus \{u_1, \ldots, u_i\}$ contains at most $2d(2d-1)i$ nodes, we have

$$\begin{aligned}
\Pr\{T_{i+1} \mid T_i\} &\geq \frac{n - 2d(2d-1)i}{n-i} \\
&\geq \left(1 - 2d(2d-1)\frac{i}{n}\right)\left(1 + \frac{i}{n}\right) \\
&= 1 - \left((2d(2d-1)-1)\frac{i}{n} + 2d(2d-1)\frac{i^2}{n^2}\right).
\end{aligned}$$

Since $\Pr\{T_1\} = 1$, we have

$$\begin{aligned}
&\Pr\{T_k\} \\
&= \prod_{i=1}^{k-1} \Pr\{T_{i+1} \mid T_i\} \\
&\geq \prod_{i=1}^{k-1} 1 - \left((2d(2d-1)-1)\frac{i}{n} + 2d(2d-1)\frac{i^2}{n^2}\right) \\
&\geq 1 - \left(\frac{(2d(2d-1)-1)k(k-1)}{2n} + \frac{2d(2d-1)(k-1)k(2k-1)}{6n^2}\right) \\
&= 1 - \left(\frac{2d^2k(k-1)}{n} - \frac{(2d+1)k(k-1)}{2n} + \frac{2d(2d-1)(k-1)k(2k-1)}{6n^2}\right).
\end{aligned}$$

Since $k < \frac{3n}{2(2d-1)} + 1/2$, we have

$$\frac{2d(2d-1)(k-1)k(2k-1)}{6n^2} - \frac{(2d+1)k(k-1)}{2n} < 0.$$

Then,

$$\Pr\{T_k\} \geq 1 - \frac{2d^2k(k-1)}{n}.$$

□

Corollary 29. *Given an* $\mathbb{H}$*-graph* $G \in \mathbb{H}_{n,2d}$ *and a randomly-selected subset* $A \subset G$*, if* $\mathbb{H}$*-graph* G_A *is obtained by removing nodes in* A *from* G *by* LEAVE*, then*

$$\lambda(G_A) \leq \lambda(G) + 1$$

with probability at least

$$1 - \frac{2d^2\,|A|\,(|A|-1)}{n}.$$

Corollary 29 shows that when a set A of leaving nodes is selected randomly, the effect on the eigenvalue of G is likely to be limited. However, the bound given by Theorem 27 becomes trivial when $t_G(A) \geq \lambda(G)$. An on-line adversary can possibly choose the set A such that $t_G(A) = \min(2d, |A|)$. Thus we cannot provide a strong guarantee against on-line adversaries.

Nevertheless, we believe it is very difficult to launch such on-line attack on a large network. The adversary needs to know the topology of the network, evaluate the effects of removing subsets of limited size, and then attack those nodes in the selected subset. Given that the network topology is changing all the time, an adversary will face significant challenges in learning the topology and launching its attack in time.

In the following, we suggest a heuristic that can reduce the effect of a leaving set $A \subset G$ on the second largest eigenvalue $\lambda(G)$:

> (Rule of Reinsertion) Whenever k neighbors of a node v leave the network, node v should reinsert itself into the network by calling LEAVE and JOIN.

When an $\mathbb{H}$-graph follows the rule of Reinsertion, it can prevent an adversary from reducing $\lambda(G)$ drastically in a short time. The rule of Reinsertion does not work if an adversary can remove a significant fraction of the network in a short time. In the extreme case, rule of Reinsertion could lead to a disintegration of the network. Nevertheless, if an adversary can disable such a large number of nodes, it can easily partition the network anyway. In addition, if an adversary is able to remove selected nodes over an extended time, we will still need a regeneration algorithm to recover from such attack.

In short, although we cannot prove the robustness of $\mathbb{H}$-graph against persistent or large-scaled on-line adversarial attacks, we believe these attacks are difficult to be implemented in practice.

2.12 Related Work

Study of random graphs was introduced by Erdős and Rényi [23]. In recent years, there have been efforts in applying random graphs in many areas of computer science. For example, in the area of switching networks, random graphs had been used in ATM networks [26] and Internet telephony signaling networks [90].

Our protocol can be viewed as a membership tracking protocol. Probabilistic broadcasts, with application for membership management, have been studied by Kermarrec, Massoulie, and Ganesh [57] and Eugster et al. [24]. The analysis in [24] is based on the assumption each node having "uniformly distributed random view" of a constant size. This is analogous to having a fixed degree in a random graph. The assumption is justified by a good correlation with the simulation results. However, it is not clear how fast their graph will converge to a topology with property close to the assumption when nodes join or leave. In [57], the random views of the nodes are provided by a set of dedicated servers.

There has been much research on membership service in the area of group communications [2]. But little has been studied that how such groups can be formed in the first place.

Our work benefits from the Friedman's result that random regular graphs are likely to have small eigenvalues [33]. Relations between eigenvalues and expanders have been studied by Alon [7] and Kahale [52].

There are geometric interpretations of a graph's eigenvalues [19]. The eigenvalues has been linked to a graph's expander property [7, 52]. Expanders have applications in diverse areas including switching networks and error-correcting codes [8, 42].

Little has been done on distributed algorithms for random graphs. Frieze and McDiarmid [35]'s survey on random graphs algorithms included several parallel algorithms but no distributed algorithms. There were several sequential algorithms for generating random regular graphs [69, 91, 56]. Random graphs constructed by centralized algorithms were also used for communication networks [26, 90].

Bauer and Wang [11] compared distributed search algorithms based on depth-first-search and flooding on three special topologies by simulations.

Kermarrec, Massoulié, and Ganesh [57] and Eugster et al. [24] have studied randomized networks for probabilistic broadcasts and membership management. The analysis in [24] is based on the assumption each node having "uniformly distributed random view" of a constant size, justified by some simulation results. It is not clear how fast their graphs converge to a topology satisfying the assumption. In [57], the random views of the nodes are provided by a set of dedicated servers that require $O(n)$ storage space.

Pandurangan, Raghavan, and Upfal [77] proposed an algorithm for building low-diameter peer-to-peer networks with bounded degrees. Although their random topology also achieves $O(\log n)$ diameter with high probability, their approach and techniques are different from ours. First, they assume a stochastic model for the arrivals and departures of nodes, while

$\mathbb{H}$-graph assumes arbitrary sequence of arrivals and departures. Second, their protocol uses a special server that is known to all nodes in the network, which is not required in $\mathbb{H}$-graphs. Third, their network is connected with high probability instead of with certainty.

2.13 Concluding Remarks

This chapter introduces a distributed approach for constructing overlay networks as random graphs. Using random walks as a sampling subroutine, we have demonstrated a scalable construction of expander graphs without any centralized server support.

Previous studies of random regular graphs indicate that they are robust against failures [97, 75]. We expect that most isolated faults would have limited effects on the probability spaces. Further work is required to devise efficient schemes for recovering from node failures and schedules for the regeneration algorithm.

Our protocol constructs one particular family of random regular graphs. It would be of interest if some other models of regular graphs can also be constructed distributedly.

In a large network, there can be a significant variation of network distance between pairs of nodes, which can be a function of the network delay, bandwidth, or reliability. We can have a hierarchy of $\mathbb{H}$-graphs based on the underlying network topology. For example, there can be an $\mathbb{H}$-graph for each regional network, an $\mathbb{H}$-graph for each country, and then a global $\mathbb{H}$-graph. Each search query will first try to search within the smaller $\mathbb{H}$-graphs before querying nodes in the larger $\mathbb{H}$-graphs.

In a heterogeneous network, a powerful machine can serve as several nodes in the $\mathbb{H}$-graphs, thus increasing the number of its effective neighbors by a constant factor. Such scheme effectively utilizes machines of different capacities and is transparent to our algorithms.

We consider the case where the JOIN requests are not well separated. If we assume an $O(1)$ arrival rate of new nodes, the number of nodes may increase from n to $n + O(\log n)$ during the an execution of JOIN. Therefore, the protocol effectively samples from a subset of n nodes in a graph of $n + O(\log n)$ nodes. Therefore, the probability of any node being sampled is at most increased by a factor of $1 + O(\log n)/n$. With perfect sampling, we have

$$\Pr\{\text{SAMPLE returns } v\} = \frac{1}{n}\left(1 + \frac{O(\log n)}{n}\right).$$

With approximated sampling, we have

$$\Pr\{\text{SAMPLE-RW returns } v\}$$
$$= \left(\frac{1}{n} + \epsilon_n\right)\left(1 + \frac{O(\log n)}{n}\right).$$

In both cases, we can show that the probability of bad graphs are increased by a constant factor, with a proof similar to that of Theorem 8.

At last, we expect that the distributed sampling algorithms and estimation algorithms discussed in this chapter could lead to interesting further investigations and be useful in other applications.

Chapter 3

Layered Expander Networks

An $\mathbb{H}$-graph is efficient for locating a member of a subset, when the size of the subset is not too small when compared with the size of the entire graph. However, if we want to locate a particular node, the expected time is $\Theta(n)$. In this section, we will improve the performance for such query by overlaying multiple $\mathbb{H}$-graphs on the same set of nodes. We introduce layered $\mathbb{H}$-graphs in Section 3.1. Degeneration into complete graphs is discussed in Section 3.2. We then discuss two variants that reduce lookup time in Section 3.3. We study the effects of random and sampling identifiers on load-balancing in Section 3.4. We discuss related work in Section 3.5.

3.1 Layered $\mathbb{H}$-Graphs

Let us start with two layers. Let G be connected as an $\mathbb{H}$-graph by our protocol. We can randomly color half of the nodes red and half of the nodes green. All the red nodes are then connected by an $\mathbb{H}$-graph consisting of red nodes only. And there is an $\mathbb{H}$-graph for the green nodes. In this case, if we want to find a particular red node v, we first find any red node using the original $\mathbb{H}$-graph, and then locate node v following the edges of the red $\mathbb{H}$-graph.

The next natural step is to consider having layers of $\mathbb{H}$-graphs until the deepest layer consists of graphs of $O(1)$ size. We call such topology a layered $\mathbb{H}$-graph. Let layer 0 refer to the $\mathbb{H}$-graph connecting the entire graph.

In an layered $\mathbb{H}$-graph with at most m layers, we can assign an identifier to each node. Let an identifier be a string $z_1 z_2 \cdots z_m$ such that $z_i \in \{0, \ldots, q_i - 1\}$ for all $i \in \{1, \ldots, m\}$. We will assume that $q_i = q$ in this chapter, although our results can be extended to the general case of distinct q_i's. Let $\mathsf{id}(u)$ be node u's identifier.

We first describe how identifiers can be used to construct a layered $\mathbb{H}$-graph. Consider a set of n nodes in which each node is assigned an identifier. The nodes are connected by

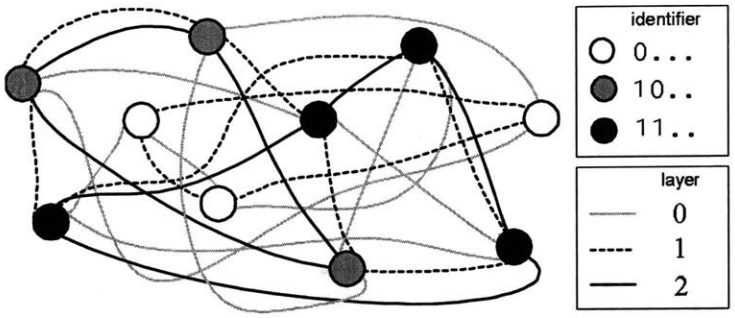

Figure 3.1: A layered $\mathbb{H}$-graph with $d = 1$.

layers of $\mathbb{H}$-graphs. In layer 0, the nodes are connected as an $\mathbb{H}$-graph. In addition, for each layer i and each prefix $z_1 \cdots z_i$, the nodes with this prefix are connected as an $\mathbb{H}$-graph, if there are at least three such nodes. For example, nodes with $z_1 = 0$ should be connected as an $\mathbb{H}$-graph, and all nodes with $z_1 = 1$ should be connected as a separate $\mathbb{H}$-graph. Figure 3.1 shows an example of a 3-layer $\mathbb{H}$-graph.

To search for a node with identifier z in a layered $\mathbb{H}$-graph, we first search for a node u with $\mathsf{id}(u)_1 = z_1$ using the layer-0 links. After that, we can use the layer-1 links of the z_1-subgraph to search for a node v with $\mathsf{id}(v)_2 = z_2$. We can repeat this process until we reach a subset of $O(1)$ size, which can be connected as a complete graph.

We will consider two possible assumptions on the assignments of the identifiers. In the first case, the identifiers are determined before the nodes join the graph. These are called *fixed identifiers* . This is a common assumption made by many other peer-to-peer networks. As another possibility, we can allow the node identifiers to be changed after the nodes have joined the network. We call this case *flexible identifiers* . Most of the algorithms that we will present are the same for both variants. We will indicate wherever flexible identifiers can be exploited to improve the constructed layered $\mathbb{H}$-graphs.

For any node u in a layered $\mathbb{H}$-graph G, we define

$$\mathsf{layer}(u, i) \equiv \{\, v \in G \mid \mathsf{id}(v)_h = \mathsf{id}(u)_h \text{ for } 0 < h \leq i \,\}\,.$$

We note that $\mathsf{layer}(u, 0) = G$ for any $u \in G$. In other words, if $v, w \in \mathsf{layer}(u, i)$, then v and w share the same i-prefix identifier. Let $|\mathsf{layer}(u, i)|$ be the number of nodes in $\mathsf{layer}(u, i)$.

The number of layers of a graph is not necessarily the length of the identifiers. When the graph is small, few layers are sufficient.

```
IDSEARCH(z, i)
1  return a node u such that id(u)_i = z_i
```

Procedure IDSEARCH can be implemented with algorithms such as SEARCH-CYCLEWALK and SEARCH-TREEWALK.

Now we present the algorithms for joining and leaving a layered $\mathbb{H}$-graph.

```
LAYERJOIN(u, i)
1  if layer(self, i) is connected as a complete graph
2     then insert u into the complete graph
3          TOREGULAR()
4     else JOIN(u, i)
5          IDSEARCH(id(u), i + 1) ⇒ LAYERJOIN(u, i + 1)
```

For any node u in a layered $\mathbb{H}$-graph G, let

$$\mathsf{depth}(u) \equiv \min \{\, i \mid \mathsf{layer}(u,i) \text{ is connected as a complete graph} \,\}.$$

Let $\mathsf{depth}(G)$ be the maximum depth of the nodes in G:

$$\mathsf{depth}(G) \equiv \max \{\, \mathsf{depth}(u) \mid u \in G \,\}.$$

```
LAYERLEAVE(i)
1  TOCOMPLETE()
2  for each j ∈ {i, ..., depth(self)} in parallel
3  do LEAVE(j)
```

Procedures JOIN(u,i) and LEAVE(i) are similar to the original JOIN and LEAVE procedures except that they only use the edges in layer i.

The LAYERSEARCH algorithm moves from layer 0 to the deepest layer of the target node.

```
LAYERSEARCH(z, i)
1  if id(self) is a prefix of z
2     then return self
3     else return IDSEARCH(z, i + 1) ⇒ LAYERSEARCH(z, i + 1)
```

The major advantage of layered $\mathbb{H}$-graphs over simple $\mathbb{H}$-graphs is that any node can be located efficiently given its identifier.

Given a layered $\mathbb{H}$-graph G and a node $u \in G$. We say that a non-complete graph $\mathsf{layer}(u,i)$ is *δ-balanced* if

$$\min_{0 \le j < q} \frac{|\{\, v \in \mathsf{layer}(u,i) \mid \mathsf{id}(v)_{i+1} = j \,\}|}{|\mathsf{layer}(u,i)|} \ge \frac{\delta}{q}.$$

If for all $u \in G$ and all $i \in \{0, \ldots, \mathsf{depth}(G)\}$ where $\mathsf{layer}(u,i)$ is not a complete graph, $\mathsf{layer}(u,i)$ is δ-balanced, then we say G is δ-balanced.

Theorem 30. *If G is δ-balanced, then* LAYERSEARCH *using* SEARCH-CYCLEWALK *takes at most $\beta(q-1)\log_q n$ hops in expectation, where*

$$\beta = \left(1 - \left(1 - \frac{\sigma}{q}\right)\log_q \delta - \frac{\sigma}{q}\log_q \sigma\right)^{-1}$$

and

$$\sigma = q - \delta q + \delta.$$

Proof. We shall prove by induction. Let $T(n)$ be the expected time to search for a target node when the graph size is n.

Base Case When the graph is connected as a complete graph. The target node can be reached in $1 - 1/n$ steps in expectation. We need to show that

$$T(n) = 1 - 1/n \leq \beta(q-1)\log_q n,$$

which can be written as

$$\frac{1 - 1/n}{\log n} \leq \frac{\beta(q-1)}{\log q}. \tag{3.1}$$

First, Equation (3.1) is satisfied when $n = 1$. For $n \geq 2$, we have

$$\frac{1 - 1/n}{\log n} \leq 2/3.$$

The right hand side of Equation (3.1) $\frac{\beta(q-1)}{\log q}$ is minimized when $q = 2$. Let

$$f(\delta) = 1 - \log\delta + \frac{\delta}{2}\left(\log\delta - \log\sigma\right).$$

We can see that $f(1) = 1$ and

$$\begin{aligned} f'(\delta) &= \left(\frac{\delta}{2} - 1\right)\frac{1}{\delta} + \frac{\log\delta}{2} - \left(\frac{\delta}{2\sigma}(1-q) + \frac{\log\sigma}{2}\right) \\ &= \frac{1}{2}\log\frac{\delta}{2-\delta} - \frac{(4-\delta)(1-\delta)}{2(2-\delta)\delta} \end{aligned}$$

for $0 < \delta < 1$. Thus $f(\delta) \geq 1$ for $0 < \delta < 1$. Therefore, Equation (3.1) is satisfied for all $q \geq 2$ and $n \geq 1$.

Inductive Step Assuming that $T(k) \leq \beta(q-1)\log_q k$ for all $k < n$, we have

$$T(k) = 1 - 1/k \leq \beta(q-1)\log_q k.$$

Let v be the target node. Consider the graph layer(u, i). It contains q layer-$i+1$ graphs $L_0, \ldots, L_{q-1}$. Let the sizes of these graphs be $a_0 n, \ldots, a_{q-1} n$. The probability that v is in L_j is a_j. The expected time it takes to reach a node in L_j is $1/a_j$ if $\mathsf{id}(v)_{i+1} \neq \mathsf{id}(u)_{i+1}$.

$$\begin{aligned}
T(n) &= -1 + \sum_{0 \leq j < q} a_j \left(1/a_j + T(a_j n)\right) \\
&= q - 1 + \sum_{0 \leq j < q} a_j T(a_j n) \\
&\leq q - 1 + \beta(q-1) \sum_{0 \leq j < q} a_j \log_q a_j n \\
&\leq \beta(q-1)\log_q n + (q-1)\left(1 + \beta \sum_{0 \leq j < q} a_j \log_q(a_j)\right)
\end{aligned}$$

We need to show that

$$\beta \sum_{0 \leq j < q} a_j \log_q(a_j) + 1 \leq 0.$$

Since $x \log x$ is a convex function, $\sum_{0 \leq j < q} a_j \log_q(a_j)$ is maximized at when a_j's has the extreme values. Since $a_j \geq \delta/q$, we have

$$\sum_{0 \leq j < q} a_j \log_q(a_j) \leq (q-1)\frac{\delta}{q}\log_q \frac{\delta}{q} + \left(1 - (q-1)\frac{\delta}{q}\right)\log_q\left(1 - (q-1)\frac{\delta}{q}\right)$$

As $\sigma = q - \delta q + \delta$, then $\delta = (q - \sigma)/(q-1)$. Then we have

$$\begin{aligned}
\sum_{0 \leq j < q} a_j \log_q(a_j) &\leq \left(1 - \frac{\sigma}{q}\right)\log_q \frac{\delta}{q} + \frac{\sigma}{q}\log_q \frac{\sigma}{q} \\
&= -1 + \left(1 - \frac{\sigma}{q}\right)\log_q \delta + \frac{\sigma}{q}\log_q \sigma
\end{aligned}$$

Thus, if we let

$$\beta = \left(1 - \left(1 - \frac{\sigma}{q}\right)\log_q \delta - \frac{\sigma}{q}\log_q \sigma\right)^{-1},$$

then $T(n) \leq \beta(q-1)\log_q n$.

□

The best possible expected number of hops required for a layered search is $\log_2 n$ on a 1-balanced layered $\mathbb{H}$-graph when $q = 2$. In this case, $\delta = \sigma = 1$, thus $\beta = 1$ as well.

3.2 Transitions Between Complete Graphs and $\mathbb{H}$-Graphs

Define integers regularsize and completesize such that

- regularsize $>$ completesize > 0,
- regularsize $\geq q$,

and for any $F = \mathsf{layer}(u, i)$ of a layered $\mathbb{H}$-graph,

- if $|F| \leq$ completesize, then F is connected as a complete graph and does not contain deeper layers;
- if $|F| \geq$ regularsize, then F is connected as a layered $\mathbb{H}$-graph and may contain deeper layers.

In a complete graph, each node has complete information of the entire graph. Thus it is easy to attain good load-balancing among the nodes when they are used for distributed lookup service. To control the size of the complete graphs, we need to implement procedures ToComplete and ToRegular efficiently.

Here we suggest a set of choices of completesize, regularsize, and procedures ToComplete and ToRegular. Let completesize $= 2d$ and regularsize $= 4d$.

```
ToComplete()
1  i ← depth(self) − 1
2  if |layer(self, i)| > completesize
3     then return
4  F ← layer(self, i)
5  Learn about all the nodes in F by walking along any Hamilton cycle
6  Contact all nodes in F to make F a complete graph
7  For all j > i, layer-j edges in F are removed
```

Let $F = \mathsf{layer}(u, \mathsf{depth}(u) - 1)$. Then the call $u \Rightarrow$ ToComplete() requires $|F| + 1$ steps and $2|F| - 1$ messages.

```
ToRegular()
1  i ← depth(self)
2  if |layer(self, i)| < regularsize
3     then return
4  F ← layer(self, i)
5  for each l in {1, ..., d}
6  do Select a random Hamilton cycle C_l for the nodes in F
7        Notify all nodes in F about their neighbors on the d Hamilton cycles
8  For all u ∈ F, graphs layer(u, i + 1) remain connected as complete graphs
```

The procedure $u \Rightarrow$ ToRegular() can be completed in a single step, because the graph $F = \mathsf{layer}(u, \mathsf{depth}(u))$ was a complete graph just before the transition. The number of messages required is $|F| - 1$.

We should note that the innocent-looking condition check $|\mathsf{layer}(u, i)| < \mathsf{regularsize}$ in procedure ToRegular can be tricky in practice. Since during the check $\mathsf{layer}(u, i)$ is still an $\mathbb{H}$-graph, node u may not know the size of $\mathsf{layer}(u, i)$. Although we presented some estimation algorithms in Section 2.10, it is probably not necessary to use those complicated algorithms for these graphs with limited size. A simple solution is for each node to keep track of the size of the $\mathbb{H}$-graph F until there are at least two layers above graph F. An even simpler heuristic is to guess the size of the graph by observing one's own neighbors. When the graph size is small, we can expect that the number of distinct neighbors of a node u is likely to drop below $2d$. And this must be true once the total number of nodes is at most $2d$.

For the case of flexible identifiers, we should choose regularsize so that it is divisible by q. And then a modified version of procedure ToRegular can rearrange the node identifiers to obtain an ideal layered $\mathbb{H}$-graph.

3.3 Better Recursive Searches

In this section we explore two ways to improve the recursive lookup time of layered $\mathbb{H}$-graphs.

3.3.1 Search with Broadcasts

One approach to improve the performance of recursive search is to broadcast on every layer. In Section 2.9, we showed that search by broadcast is $O(\log \psi^{-1})$, where ψ is the fraction of the target set. Let procedure IDSearch-B(z, i) be Search-TreeWalk(ϕ, ∞, ∞) with criterion function ϕ set to be $\mathsf{id}(u)_i = z_i$. Therefore, IDSearch-B(z, i) broadcasts on layer $i - 1$ and stops once a node u such that $\mathsf{id}(u)_i = z_i$ is found.

In the rest of this section, $\log n$ means $\log_2 n$.

We first establish a bound on the performance of IDSearch-B on any layer.

Lemma 31. *Consider an ideal layered* $\mathbb{H}$*-graph* G*. Procedure* IDSEARCH-B(z,i) *finds a node* u *with* $\mathsf{id}(u)_i = z_i$ *in*

$$2\log_{2(d-1)}(2\log n)$$

steps with probability at least $1 - 1/n^{1.25}$.

Proof. First, we assume $i = 1$. By Theorem 11, the number of nodes reached by a broadcast after t steps is

$$\frac{n}{1 + 4\frac{n-1}{(2(d-1))^t}}.$$

Now let t be the number of steps that satisfies

$$\frac{n}{1 + 4\frac{n-1}{(2(d-1))^t}} = 1.25 \ln n \log n.$$

Then by Lemma 15, a broadcast of t steps finds a node u with $\mathsf{id}(u)_i = z_i$ with probability at least $1 - 1/n^{1.25}$ when it covers $1.25 \ln n \log n$ nodes. We have

$$\begin{aligned}
t &= \log_{2(d-1)} \frac{4(1-\frac{1}{n})}{\frac{1}{1.25 \ln n \log n} - \frac{1}{n}} \\
&\leq \log_{2(d-1)}((1.25)4 \ln n \log n) + \frac{1.25 \ln n \log n}{n \ln 2(d-1)} \\
&= \log_{2(d-1)}((1.25 \ln 2)4 \log n \log n) + \frac{1.25 \ln n \log n}{n \ln 2(d-1)} \\
&= \log_{2(d-1)}(4 \log^2 n) + \frac{\log(1.25 \ln 2)}{\log 2(d-1)} + \frac{1.25 \log^2 n}{n \log 2(d-1)} \\
&= 2\log_{2(d-1)}(2 \log n) + \frac{\log(1.25 \ln 2)}{\log 2(d-1)} + \frac{1.25 \log^2 n}{n \log 2(d-1)}.
\end{aligned}$$

If $i > 1$, the graph size should be $n/\log^{i-1} n$ instead of n. Therefore

$$t \leq 2\log_{2(d-1)}(2 \log n) + \frac{\log(1.25 \ln 2)}{\log 2(d-1)} + \frac{\log^{i+1} n}{n \log 2(d-1)}.$$

However, the size of the deepest $\mathbb{H}$-graph layer is at least $2 \log n$, because any complete graph has size at least 2, and the next layer will have size $2 \log n \geq 2 \log 16 = 8$.

And then we have

$$\begin{aligned}
&\frac{\log(1.25 \ln 2)}{\log 2(d-1)} + \frac{1.25 \log^{i+1} n}{n \log 2(d-1)} \\
&\leq \frac{1}{\log 2(d-1)} \left(\log(1.25 \ln 2) + \frac{1.25}{8} \right) \\
&< 0.
\end{aligned}$$

□

Theorem 32. *Consider an ideal layered* $\mathbb{H}$*-graph* G *with* $d = 1+\log^{\epsilon} n$, $\epsilon < 1$, *and* $q = \log n$, *so that each node has*

$$(2 + 2\log^{\epsilon} n)\frac{\log n}{\log\log n} = 2\frac{\log^{1+\epsilon} n}{\log\log n} + 2\frac{\log n}{\log\log n}$$

neighbors. A LAYERSEARCH *using* IDSEARCH-B *takes at most*

$$\frac{2\log n}{\epsilon\log\log n}$$

steps with probability at least

$$1 - \frac{1}{n}.$$

Proof. Since $q = \log n$, then there are $\log n/\log\log n$ layers. The overall time required by LAYERSEARCH using IDSEARCH-B is

$$\begin{aligned}
&2\log_{2(d-1)}(2\log n)\frac{\log n}{\log\log n}\\
&= 2\log_{2\log^{\epsilon} n}(2\log n)\frac{\log n}{\log\log n}\\
&\leq 2\frac{1+\log\log n}{1+\log\log^{\epsilon} n}\frac{\log n}{\log\log n}\\
&= 2\frac{1+\log\log n}{1+\epsilon\log\log n}\frac{\log n}{\log\log n}\\
&< \frac{2\log n}{\epsilon\log\log n}
\end{aligned}$$

with probability at least

$$1 - \frac{\log n}{\log\log n}\frac{1}{n^{1.25}} > 1 - \frac{1}{n}$$

by the union bound, when $n \geq 16$. □

Corollary 33. *Consider an ideal layered* $\mathbb{H}$*-graph* G *with*

$$2d = 2 + 2(\log n)^{\log\log\log n/\log\log n} = 2\log\log n + 2$$

and $q = \log n$, *so that each node has*

$$2\log n\left(1 + \frac{1}{\log\log n}\right)$$

neighbors. A LAYERSEARCH *using* IDSEARCH-B *takes at most*

$$\frac{2\log n}{\log\log\log n}$$

steps with probability at least

$$1-\frac{1}{n}.$$

Proof. The result follows by setting $\epsilon = \log\log\log n/\log\log n$ in Theorem 32. □

3.3.2 Search with Layered Broadcasts

We further optimize the results we obtained in Section 3.3.1 by neighbors in multiple layers.

In algorithm IDSEARCH-BL(i), we first contact all neighbors in layers at least $i-1$, and then perform IDSEARCH-B on those nodes. For example, when $i=4$, the initial node first contact all nodes at layers at least 3, and issue a IDSEARCH-B(i) request to each of these nodes. Algorithm IDSEARCH-BL exploits the situation that on the shallow layers (when i is small), a node has access to around $2d\log_q n$ neighbors. Thus we can contact a much lager number of nodes within a short time.

In the following we analyze the performance of LAYERSEARCH using IDSEARCH-BL. We will show that this technique achieves the best possible asymptotic look up time as limited by the node degrees.

We first analyze the number of broadcasting steps required for each layer.

Lemma 34. *Consider an ideal layered $\mathbb{H}$-graph G. Let b be the size of the smallest layer considered. Let $j=\log_q(n/b)-i+1$. Let $\delta>1$. With probability at least*

$$1-\left(\frac{e^{\delta-\mu}}{(\frac{\delta}{\mu})^{\delta}}+\frac{1}{e^{\log^{1-\epsilon}n}}\right),$$

where

$$\mu=\frac{1}{b}\frac{d-1}{2+\frac{d-3}{b}}\frac{q}{(q-1)^2},$$

procedure IDSEARCH-BL(z,i) *finds a node u with* $\mathrm{id}(u)_i=z_i$ *in*

$$\log_{2(d-1)}\frac{q\log^{1-\epsilon}n}{j}+\log_{2(d-1)}16e^{1/4}-\log_{2(d-1)}\left(1-\frac{\delta}{mj}\right)+\frac{\log^{1-\epsilon}n}{bq^{j-1}\ln 2(d-1)}+O\left(\frac{\log_d n}{bq^{j-1}}\right)^2$$

steps

Proof. According to Theorem 11, after the first step, a broadcast can reach

$$\frac{b}{1+4\frac{b-1}{(2(d-1))}} = \frac{d-1}{2+\frac{d-3}{b}}$$

nodes.

Let

$$m = \frac{d-1}{2+\frac{d-3}{b}}.$$

Let $j = \log_q(n/b) - i + 1$ be the number of layers at or above layer i. The number of nodes covered after the first step is at most jm. Let random variable X be the number of overlapping nodes in the jm nodes covered by the first step. When there are $k+1$ layers at or above i, the expected number of overlapping nodes with previous layers is at most $mk\frac{1}{bq^k}$ because the size of the layer is bq^k. Therefore, random variable X is a sum of independent hypergeometric variables with expectation:

$$\begin{aligned}
\mu &< \sum_{k=1}^{j-1} \frac{mk}{bq^k} \\
&= \frac{m}{b} \sum_{k=1}^{j-1} \frac{k}{q^k} \\
&< \frac{m}{b} \frac{1/q}{(1-1/q)^2} \\
&= \frac{m}{b} \frac{q}{(q-1)^2}.
\end{aligned}$$

It is known that [93, page 29] we can apply the Chernoff bound on X:

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

for $\delta > 1$. We have

$$\Pr\{X > \delta\} = \Pr\left\{mj - X < mj\left(1 - \frac{\delta}{mj}\right)\right\}.$$

Let

$$\alpha = 1 - \frac{\delta}{mj}.$$

Now assume that $X > \delta$. This means that there number of nodes reached after the first step, across all layers above i, is at least $mj\alpha$.

Then by an argument similar to Theorem 11, we can show that after $t-1$ steps, the number of nodes reached by the broadcast is at least

$$\frac{q^j}{\frac{1}{b} + 4\frac{q^j/jm\alpha - 1}{(2(d-1))^t}}.$$

By Lemma 15, a broadcast of t steps finds a node u with $\mathrm{id}(u)_i = z_i$ with probability at least $1 - 1/e^{\log^{1-\epsilon} n}$ when it covers $q\log^{1-\epsilon} n$ nodes. Then we have

$$\begin{aligned}
t &\le 1 + \log_{2(d-1)} 4\frac{\frac{1}{jm\alpha} - \frac{1}{q^j}}{\frac{1}{q\log^{1-\epsilon} n} - \frac{1}{bq^j}} \\
&= 1 + \log_{2(d-1)} 4\left(\frac{q\log^{1-\epsilon} n}{jm\alpha} - \frac{1}{q^j}\right) + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)}\right)^2 \\
&\le 1 + \log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j\alpha} + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_{2(d-1)} n}{bq^{j-1}}\right)^2 + \log_{2(d-1)}\left(2 + \frac{d-3}{b}\right) \\
&\quad - \log_{2(d-1)}(d-1) + \log_{2(d-1)} 4 \\
&= \log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j\alpha} + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_{2(d-1)} n}{bq^{j-1}}\right)^2 \\
&\quad + \log_{2(d-1)}\left(1 + \frac{d-3}{2b}\right) + \log_{2(d-1)} 16 \\
&\le \log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j\alpha} + \log_{2(d-1)} 16 + \frac{d-3}{2(2d)\ln 2(d-1)} \\
&\quad + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_{2(d-1)} n}{bq^{j-1}}\right)^2 \\
&< \log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j\alpha} + \log_{2(d-1)} 16 + \frac{\log e}{4\log 2(d-1)} \\
&\quad + \frac{\log_{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_{2(d-1)} n}{bq^{j-1}}\right)^2 \\
&= \log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j\alpha} + \log_{2(d-1)} 16e^{1/4} + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_{2(d-1)} n}{bq^{j-1}}\right)^2
\end{aligned}$$

because $b \ge 2d$.

□

Next we analyze the performance of LAYERSEARCH using IDSEARCH-BL up to a layer of size b.

Lemma 35. *Consider an ideal layered $\mathbb{H}$-graph G. A* LAYERSEARCH *using* IDSEARCH-BL *from layer* 1 *to layer* $h = \log_q(n/b)$ *takes at most*

$$h\left(\log_{2(d-1)} 16e^{1/4} + \log_{2(d-1)} \frac{qe\log^{1-\epsilon} n}{h}\right) + \log_{2(d-1)} \frac{1}{\sqrt{2h\pi}}$$
$$+ \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} + O\left(\frac{\log_d^2 n}{b^2}\right) + \frac{\delta H_h}{m\ln 2(d-1)} + O\left(\frac{\delta^2}{m^2 \log d}\right)$$

steps with probability

$$1 - h\left(\frac{e^{\delta-\mu}}{(\frac{\delta}{\mu})^\delta} + 1/e^{\log^{1-\epsilon} n}\right)$$

where

$$\mu = \frac{1}{b}\frac{d-1}{2+\frac{d-3}{b}}\frac{q}{(q-1)^2}.$$

Proof. Let $h = \log_q(n/b)$ be the number of layers. The expected total time required is at most

$$\sum_{j=1}^{h} \mathrm{E}\left[\text{time required for IDSEARCH-BL}(h-j+1)\right]$$
$$= h\left(\log_{2(d-1)} 16e^{1/4} + \log_{2(d-1)} \log^{1-\epsilon} n\right)$$
$$+ \sum_{j=1}^{h}\left(\log_{2(d-1)} \frac{q\log^{1-\epsilon} n}{j} + + \frac{\log^{1-\epsilon} n}{bq^{j-1}\ln 2(d-1)} + O\left(\frac{\log_d n}{bq^{j-1}}\right)^2 + \log_{2(d-1)}\left(1-\frac{\delta}{mj}\right)^{-1}\right)$$
$$= h\left(\log_{2(d-1)} 16e^{1/4} + \log_{2(d-1)} \log^{1-\epsilon} n\right) + \log_{2(d-1)} \frac{\left(q\log^{1-\epsilon} n\right)^h}{h!} + \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)}$$
$$+ O\left(\frac{\log_d^2 n}{b^2}\right) + \frac{\delta H_h}{m\ln 2(d-1)} + O\left(\frac{\delta^2}{m^2\log d}\right)$$
$$< h\left(\log_{2(d-1)} 16e^{1/4} + \log^{1-\epsilon}_{2(d-1)} n\right) + \log_{2(d-1)} \frac{\left(q\log^{1-\epsilon} n\right)^h}{\sqrt{2\pi h}(h/e)^h}$$
$$+ \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} + O\left(\frac{\log_d^2 n}{b^2}\right) + \frac{\delta H_h}{m\ln 2(d-1)} + O\left(\frac{\delta^2}{m^2\log d}\right)$$
$$= h\left(\log_{2(d-1)} 16e^{1/4} + \log_{2(d-1)} \frac{qe\log^{1-\epsilon} n}{h}\right) + \log_{2(d-1)} \frac{1}{\sqrt{2h\pi}}$$
$$+ \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} + O\left(\frac{\log_d^2 n}{b^2}\right) + \frac{\delta H_h}{m\ln 2(d-1)} + O\left(\frac{\delta^2}{m^2\log d}\right).$$

□

To prepare for Theorem 39, we first need to prove three technical lemmas.

Lemma 36. *For $n \geq 16$,*

$$\log_{2(d-1)} \frac{1}{\sqrt{2h\pi}} + \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} < 0$$

where

$$h = \frac{\log n}{\epsilon \log\log n}\left(1 - \frac{\log\log\log n}{\log\log n}\right),$$

and

$$b = n^{\frac{\log\log\log n}{\log\log n}}.$$

Proof. First, we can rewrite the expression as

$$\begin{aligned}
&\frac{1}{\ln 2(d-1)}\left(\ln\frac{1}{\sqrt{2h\pi}} + \frac{\log^{1-\epsilon} n}{b(1-1/q)}\right) \\
&= \frac{1}{\ln 2(d-1)}\left(\ln\frac{1}{\sqrt{2h\pi}} + \ln e^{\frac{\log^{1-\epsilon} n}{b(1-1/q)}}\right) \\
&= \frac{1}{\ln 2(d-1)} \ln\left(\frac{e^{\frac{\log^{1-\epsilon} n}{b(1-1/q)}}}{\sqrt{2h\pi}}\right).
\end{aligned}$$

For $n \geq 16$, $\epsilon \leq 0.5$, and $q \geq 2$. we have

$$\frac{b(1-1/q)}{\log^{1-\epsilon} n} = n^{\frac{\log\log\log n}{\log\log n}} \frac{1-1/q}{\log^{1-\epsilon} n} \geq \frac{16^{1/2}(1-1/2)}{2} = 1.$$

And, for $n \geq 16$, we have

$$\sqrt{2\pi h} = \sqrt{2\pi \frac{\log n}{\epsilon \log\log n}\left(1 - \frac{\log\log\log n}{\log\log n}\right)} \geq \sqrt{4\pi}.$$

Since $e < \sqrt{4\pi}$, we conclude that for $n \geq 16$,

$$\frac{e^{\frac{1}{b(1-1/q)\log^{1-\epsilon} n}}}{\sqrt{2h\pi}} < 1,$$

which then implies

$$\frac{1}{\ln 2(d-1)}\left(\ln\frac{1}{\sqrt{2h\pi}} + \ln e^{\frac{\log^{1-\epsilon} n}{b(1-1/q)}}\right) < 0.$$

□

Lemma 37. *Let*

$$\begin{aligned}\delta &= \frac{\log\log n}{\log\log\log n}\\ h &= \frac{\log\log n}{\log\log\log n}\log_q n\\ q &= \log^{\epsilon} n\\ d &= 1+(\log\log n)/2,\end{aligned}$$

where

$$\frac{1}{\log\log n} \le \epsilon \le 0.5.$$

For $n \ge 16$,

$$\frac{\delta H_h}{m\ln 2(d-1)} < 4.4\frac{\log\log n}{(\log\log\log n)^2} + 1.1$$

Proof.

$$\begin{aligned}&\frac{\delta H_h}{m\ln 2(d-1)}\\ &< \frac{\delta(\ln h+\gamma+\frac{1}{2h})}{m\ln\log\log n}\\ &< \frac{\delta}{m\ln\log n}\left(\ln\left(\frac{\log n}{\epsilon\log\log n}\left(1-\frac{\log\log\log n}{\log\log n}\right)\right)+0.58+\frac{1}{2h}\right)\\ &= \frac{\delta}{m\log\log\log n}\left(\log\log n+\log\left(1-\frac{\log\log\log n}{\log\log n}\right)+(0.58+\frac{1}{2h})\log e\right)\\ &< \frac{\delta}{m\log\log\log n}\left(\log\log n+\log\left(1-\frac{\log\log\log n}{\log\log n}\right)+(0.58+\frac{1}{4})\log e\right)\\ &< 1.1\frac{\log\log n}{\log\log\log n}\frac{\log\log n}{m\log\log\log n}\\ &= 1.1\left(\frac{\log\log n}{\log\log\log n}\right)^2\frac{2+\frac{d-3}{b}}{d-1}\\ &< 1.1\left(\frac{\log\log n}{\log\log\log n}\right)^2\left(\frac{4}{\log\log n}+\frac{1}{b}\right)\\ &= 4.4\frac{\log\log n}{(\log\log\log n)^2}+\frac{1.1(\log\log n)^2}{n^{\frac{\log\log\log n}{\log\log n}}(\log\log\log n)^2}\\ &\le 4.4\frac{\log\log n}{(\log\log\log n)^2}+1.1\end{aligned}$$

for $n \ge 16$. □

Lemma 38. *For $n \geq 16$,*

$$-\frac{\log\log\log n}{\log\log n}\log 8e^{5/4} + \log\frac{1}{1-\frac{\log\log\log n}{\log\log n}} < 0.$$

Proof. We need to show that

$$\frac{\log\log n}{\log\log\log n}\log\frac{1}{1-\frac{\log\log\log n}{\log\log n}} < \log 8e^{5/4}.$$

When $n = 16$,

$$\frac{\log\log n}{\log\log\log n}\log\frac{1}{1-\frac{\log\log\log n}{\log\log n}} = 2 < \log 8e^{5/4}.$$

The value of this expression decreases when n increases. □

Theorem 39 shows that a layered search with IDSEARCH-BL can achieve asymptotically degree-optimal routing.

Theorem 39. *Consider an ideal layered $\mathbb{H}$-graph G of size $n \geq 16$, $d = 1 + (\log\log n)/2$, $q = \log^{\epsilon} n$, $b = n^{\frac{\log\log\log n}{\log\log n}}$, $\epsilon \in [1/\log\log n, 0.5]$. A* LAYERSEARCH *using* IDSEARCH-BL *from layer 1 to layer $h = \log_q(n/b)$ takes at most*

$$\frac{\log n}{\log\log n}\left(1 + \frac{7.2}{\log\log\log n}\right) + 1.1 + O\left(\frac{1}{(\log\log\log n)^3}\right).$$

steps, with probability at least

$$1 - \left(\frac{\log^2 n}{4n} + \frac{\log n}{e^{\log^{1-\epsilon} n}}\right).$$

Proof. Since $q = \log^{\epsilon} n$, $h = \log_q(n/b)$, and $b = n^{\frac{\log\log\log n}{\log\log n}}$, we have

$$h = \frac{\log n}{\epsilon\log\log n}\left(1 - \frac{\log\log\log n}{\log\log n}\right)$$

layers.

From Lemma 35, the number of steps required can be broken into five terms:

$$\begin{aligned}
&h\left(\log_{2(d-1)} 16e^{1/4} + \log_{2(d-1)} \frac{qe}{h}\right) \\
&+ \log_{2(d-1)} \frac{1}{\sqrt{2h\pi}} + \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} \\
&+ \frac{\delta H_h}{m \ln 2(d-1)} \\
&+ O\left(\frac{\delta^2}{m^2 \log d}\right) \\
&+ O\left(\frac{\log_d^2 n}{b^2}\right),
\end{aligned}$$

where

$$m = \frac{d-1}{2 + \frac{d-3}{b}}$$

and

$$\delta = \frac{\log\log n}{\log\log\log n}.$$

The second term is bounded by Lemma 36, which shows that

$$\log_{2(d-1)} \frac{1}{\sqrt{2h\pi}} + \frac{\log^{1-\epsilon} n}{b(1-1/q)\ln 2(d-1)} < 0$$

for $n \geq 16$.

The third term is bounded by Lemma 37, which shows that

$$\frac{\delta H_h}{m \ln 2(d-1)} < 4.4 \frac{\log\log n}{(\log\log\log n)^2} + 1.1.$$

For the fourth term, we have

$$\begin{aligned}
O\left(\frac{\delta^2}{m^2 \log d}\right) &= O\left(\frac{(\log\log n)^2}{(\log\log\log n)^2 (\log\log n)^2 \log\log\log n}\right) \\
&= O\left(\frac{1}{(\log\log\log n)^3}\right).
\end{aligned}$$

For the fifth term, we can also show that

$$O\left(\frac{\log_d^2 n}{b^2}\right) = O\left(\frac{1}{(\log\log\log n)^3}\right).$$

And for the first term,

$$
\begin{aligned}
& h\left(\log_{2(d-1)} 16e^{5/4} + \log_{2(d-1)} \frac{q \log^{1-\epsilon} n}{h}\right) \\
&= h\left(\log_{2(d-1)} 16e^{5/4} + \log_{2(d-1)} \frac{\log n}{h}\right) \\
&= \frac{\log(n/b)}{\epsilon \log\log\log n \log\log n}\left(\log 16e^{5/4} + \log \frac{\epsilon \log\log n}{1 - \frac{\log\log\log n}{\log\log n}}\right) \\
&< \frac{\log n}{\epsilon \log\log\log n \log\log n}\left(1 - \frac{\log\log\log n}{\log\log n}\right) \\
&\quad \left(\log 16e^{5/4} + \log\epsilon + \log\log\log n + \log \frac{1}{1 - \frac{\log\log\log n}{\log\log n}}\right) \\
&< \frac{\log n}{\epsilon \log\log n}\left(1 + \frac{\log 8e^{5/4}}{\log\log\log n}\right) \\
&< \frac{\log n}{\epsilon \log\log n}\left(1 + \frac{5}{\log\log\log n}\right).
\end{aligned}
$$

In the last inequality, we had applied Lemma 38:

$$
-\frac{\log\log\log n}{\log\log n} \log 8e^{5/4} + \log \frac{1}{1 - \frac{\log\log\log n}{\log\log n}} < 0.
$$

Therefore, the overall time required is

$$
\begin{aligned}
& \frac{\log n}{\epsilon \log\log n}\left(1 + \frac{5}{\log\log\log n}\right) + 4.4 \frac{\log\log n}{(\log\log\log n)^2} + 1.1 + O\left(\frac{1}{(\log\log\log n)^3}\right) \\
&\leq \frac{\log n}{\epsilon \log\log n}\left(1 + \frac{7.2}{\log\log\log n}\right) + 1.1 + O\left(\frac{1}{(\log\log\log n)^3}\right)
\end{aligned}
$$

for $n \geq 16$ and $\epsilon < 0.5$.

The probability of not reaching the time constraint in each layer is at most

$$
\frac{e^{\delta-\mu}}{(\frac{\delta}{\mu})^{\delta}} + \frac{1}{e^{\log^{1-\epsilon} n}}.
$$

We have

$$\begin{aligned}\mu &= \frac{m}{b}\frac{q}{(q-1)^2} \\ &= \frac{d-1}{2b+d-3}\frac{q}{(q-1)^2} \\ &< \frac{\log\log n}{4b}\frac{q}{(q-1)^2} \\ &= \frac{\log\log n}{4bq'},\end{aligned}$$

where $q' = (q-1)^2/q$.

We can verify that $\delta - \mu < 0$ for $n \geq 5$, thus $e^{\delta-\mu} < 1$.

Then we have

$$\begin{aligned}\frac{h}{(\frac{\delta}{\mu})^\delta} &= \frac{\log n}{\log\log^\epsilon n}\left(1 - \frac{\log\log\log n}{\log\log n}\right)\left(\frac{\log\log n}{4bq'}\right)^{\log\log n/\log\log\log n} \\ &< \frac{\log n}{n}\left(\frac{\log\log n}{4q'}\right)^{\log\log n/\log\log\log n} \\ &\leq \frac{\log n}{n}\left(\frac{\log\log n}{2}\frac{1}{4(1-1/2)^2}\right)^{\log\log n/\log\log\log n} \\ &< \frac{\log^2 n}{4n}.\end{aligned}$$

For the second term, we have

$$\frac{h}{e^{\log^{1-\epsilon} n}} < \frac{\log n}{e^{\log^{1-\epsilon} n}}$$

for $n \geq 16$.

The total probability of not reaching the time expectation is at most

$$\frac{\log^2 n}{4n} + \frac{\log n}{e^{\log^{1-\epsilon} n}}.$$

□

Theorem 40. *Consider an ideal layered $\mathbb{H}$-graph G of size $n \geq 16$ and*

$$2d = 2 + (4\log\log n)^{\log\log\log n/\log\log n},$$

so that each node has

$$\log\log n + O(\log\log\log n)$$

neighbors. Let $q = \log^{\epsilon} n$, *such that* $(\log\log n)^{-1} \leq \epsilon \leq 0.5$.

We perform recursive search using IDSEARCH-BL *with* $b = n^{\frac{\log\log\log n}{\log\log n}}$, *and switch to* IDSEARCH-B *after layer* $h = \log_q(n/b)$. *The overall time required is*

$$\frac{(2+\epsilon^{-1})\log n}{\log\log n}\left(1+\frac{7.2}{\log\log\log n}\right)+1.1+O\left(\frac{1}{\log\log\log n}\right)$$

with probability

$$1-\left(\frac{1}{n}+\frac{\log^2 n}{4n}+\frac{\log n}{e^{\log^{1-\epsilon} n}}\right).$$

Proof. From Theorem 39, the IDSEARCH-BL recursive search with $b = n^{\frac{\log\log\log n}{\log\log n}}$ is

$$\frac{\log n}{\log\log n}\left(1+\frac{7.2}{\log\log\log n}\right)+1.1+O\left(\frac{1}{(\log\log\log n)^3}\right).$$

When the size of the graph is $n^{\frac{\log\log\log n}{\log\log n}}$ the time to IDSEARCH-B recursive search takes

$$\frac{\log n^{\frac{\log\log\log n}{\log\log n}}}{\log\log\log n}+O\left(\frac{1}{\log\log\log n^{\frac{\log\log\log n}{\log\log n}}}\right)=\frac{\log n}{\log\log n}+O\left(\frac{1}{\log\log\log n}\right).$$

□

3.4 Balancing of Layered $\mathbb{H}$-Graphs

3.4.1 Random Identifiers

In this subsection we assume that identifiers are chosen uniformly at random. A hash function can map the name of a node into an identifier. Alternatively, each node can pick an identifier randomly. We will assume that the identifiers have sufficient number of bits so that the probability of collisions is negligible.

One can expect that a uniform hash function or a good random number generator should be able to produce well-balanced layered $\mathbb{H}$-graphs. We will verify this intuition in the following.

We first show that identifiers of length $O(\log n)$ are sufficient.

Theorem 41. *Let* completesize $= 2d$ *and* $d \geq 2$. *Consider a layered* $\mathbb{H}$*-graph* G *with uniformly distributed identifiers. The depth of* G *is at most* $(1+\epsilon)\log_q n$ *with probability at least* $1 - 1/n^{2d\epsilon-1}$.

Proof. For any $u \in G$, $\mathsf{depth}(u)$ is more than $(1+\epsilon)\log_q n$ only if

$$\left|\mathsf{layer}(u, (1+\epsilon)\log_q n)\right| \geq \mathsf{completesize}.$$

We have

$$\mathrm{E}\left[\left|\mathsf{layer}(u, (1+\epsilon)\log_q n)\right|\right] = n/q^{(1+\epsilon)\log_q n} = 1/n^{\epsilon}.$$

Since $\mathsf{completesize} = 2d$, and that the identifiers are picked independently, by the Chernoff bound, we have

$$\begin{aligned}\Pr\left\{\left|\mathsf{layer}(u, (1+\epsilon)\log_q n)\right| \geq 2d\right\} &\leq \left(\frac{e^{2dn^{\epsilon}-1}}{(2dn^{\epsilon})^{2dn^{\epsilon}}}\right)^{1/n^{\epsilon}} \\ &\leq 1/n^{2d\epsilon}.\end{aligned}$$

By the union bound, the probability that any $u \in G$ has depth more than $(1+\epsilon)\log_q n$ is at most $1/n^{2d\epsilon-1}$. □

We then show that layered $\mathbb{H}$-graphs of sufficient size are likely to be 1/2-balanced.

Lemma 42. *For any $u \in G$ and non-complete-graph $\mathsf{layer}(u,i)$, $\mathsf{layer}(u,i)$ is 1/2-balanced with probability at least $1 - 2e^{-n/12q^{i+1}}$.*

Proof. Consider the case that we are in a layer-i $\mathbb{H}$-graph trying to search for a layer-$(i+1)$ node. Let random variable X be the size of the layer-i $\mathbb{H}$-graph. Let random variable Y be the size of the layer-$(i+1)$ $\mathbb{H}$-graph that we are trying to reach. Therefore, $\mathrm{E}[X] = n/q^i$, and $\mathrm{E}[Y \mid X] = X/q$. First, we derive the probability that Y is smaller than a linear fraction of X.

$$\begin{aligned}&\Pr\{Y < \delta X/q\} = \\ &\Pr\left\{X < \frac{(1-\epsilon)n}{q^i}\right\}\Pr\left\{Y < \frac{\delta X}{q} \mid X < \frac{(1-\epsilon)n}{q^i}\right\} \\ &+\Pr\left\{X \geq \frac{(1-\epsilon)n}{q^i}\right\}\Pr\left\{Y < \frac{\delta X}{q} \mid X \geq \frac{(1-\epsilon)n}{q^i}\right\}.\end{aligned}$$

Random variable X can be considered as a sum of n independent Poisson trials with success probability $1/q^i$. Thus, by Chernoff bound [73, p. 70], we have

$$\Pr\left\{X < (1-\epsilon)n/q^i\right\} < e^{-n\epsilon^2/2q^i}.$$

Next, we have $\mathrm{E}\left[Y \mid X \geq (1-\epsilon)n/q^i\right] \leq (1-\epsilon)n/q^{i+1})$. Thus, by Chernoff bound,

$$\Pr\left\{Y < \delta X/q \mid X \geq (1-\epsilon)n/q^i\right\} < e^{-(1-\delta)^2(1-\epsilon)n/2q^{i+1}}.$$

Setting $\delta = 1/2$ and $\epsilon = 1/3$, we have

$$\begin{aligned}\Pr\{Y < \delta X/q\} &< e^{-(1-\delta)^2(1-\epsilon)n/2q^{i+1}} + e^{-\epsilon^2 n/2q^i} \\ &= e^{-n/12q^{i+1}} + e^{-n/18q^i} \\ &< 2e^{-n/12q^{i+1}},\end{aligned}$$

because $q \geq 2$ and $1/12q < 1/18$.

□

Once we have Lemma 42, we can prove Theorem 43, which gives the time complexity of LAYERSEARCH. Analyses of message complexities are omitted for brevity.

Theorem 43. *If the identifiers of a layered $\mathbb{H}$-graph G are randomly distributed, then with high probability (with respect to the topology G),* LAYERSEARCH$(z, 0)$ *returns in expected time $O(q \log n)$.*

Proof. First consider those $\mathsf{layer}(u, i)$ where

$$\mathrm{E}\left[|\mathsf{layer}(u, i)|\right] = n/q^i > 12q \ln n.$$

According to Lemma 42, such $\mathsf{layer}(u, i)$ is not $1/2$-balanced with probability at most

$$2e^{-n/12q^{i+1}} < 2/n.$$

By Theorem 41, there are at most $2\log_q n$ layers with probability at least $1 - 1/n^{4d-1}$. By the union bound, the probability that any of these $\mathbb{H}$-graph is not $1/2$-balanced is at most $2\log_q n/n$.

When LAYERSEARCH reaches a $\mathsf{layer}(v, j)$ with expected size at least $12q \ln n$, the actual size of $\mathsf{layer}(v, j)$ is at most $24q \ln n$ with high probability:

$$\Pr\{|\mathsf{layer}(v, j)| > (1+1)12q \ln n\} < \left(\frac{e}{4}\right)^{12q \ln n} = n^{-12(\ln 4 - 1)}.$$

Then we only need to traverse at most $24q \ln n$ nodes to reach our target. Therefore, the total expected time of the search is $O(q \log n)$.

□

The time complexity of LAYERJOIN is the same as LAYERSEARCH because at each level, JOIN can be performed with IDSEARCH in parallel.

3.4.2 Sampling Identifiers

Sometimes a solution based on random identifiers is unsatisfactory. If identifiers are generated by a hash function, an adversary can potentially select the logical locations of new nodes by choosing the 'name' of the nodes accordingly. Even if we assume that joining nodes have uniformly distributed identifiers, the leaving nodes can be picked by an adversary to make the graph unbalanced.

For certain applications, we can construct a layered $\mathbb{H}$-graph that is "self-correcting" by allowing the new nodes choose their own identifiers with the goal of enhancing the balancedness of the graph.

We shall illustrate this approach with an example where $q = 2$. To balance the "tree" of $\mathbb{H}$-graphs, a new node should try to join the smaller $\mathbb{H}$-graphs. For example, in a graph with 1000 nodes, if there are 700 nodes with $z_1 = 0$ and 300 nodes with $z_1 = 1$, a new node joining the graph should try to set its z_1 to 1. It is expensive to actually count the number of nodes with $z_1 = 0$ or $z_1 = 1$. However, it is not difficult for a node to guess intelligently which set is larger. If node u samples one node in the graph, there is a probability of 7/10 observing a $z_1 = 0$ node and a probability of 3/10 observing a $z_1 = 1$ node. So if u observes a $z_1 = 0$ node, then it should set its z_1 to 1, and vice versa. For a more prudent decision, u can sample more nodes and pick the one with the least-frequently observed prefix. This method can be generalized to arbitrary q:

```
SAMPLING-IDENTIFIER(q, i)
1  u ← SAMPLE(i − 1) // using layer-(i − 1) edges only
2  return UNIFORM({0, ..., q − 1} \ id(u)_i)
```

We first consider 2-layer $\mathbb{H}$-graphs. For any q, let random variable $S(q)_n$ be the number of $z_1 = 0$ nodes in a layered $\mathbb{H}$-graph of n nodes if identifiers are chosen by SAMPLING-IDENTIFIER. Let random variable $X(q)_n$ be the number of $z_1 = 0$ nodes when identifiers are chosen uniformly at random.

It is natural to expect that this scheme will lead to more balanced graphs, when compared to graphs with uniformly distributed identifiers. In the following we will analyze $S(q)_n$ and show that the variance of $S(q)_n$ is smaller than the variance of $X(q)_n$.

Initially, when $n = 1$, we have

$$\Pr\{S(q)_0 = 0\} = 1.$$

First, we state the recurrence relation of $S(q)_n$:

$$\begin{aligned}
&\Pr\{S(q)_{n+1} = i\} \\
&= \Pr\{\, S(q)_{n+1} = i \mid S(q)_n = i \,\} \Pr\{S(q)_n = i\} + \\
&\quad \Pr\{\, S(q)_{n+1} = i \mid S(q)_n = i-1 \,\} \Pr\{S(q)_n = i-1\} \\
&= \left(\frac{i}{n} + \frac{q-2}{q-1}\frac{n-i}{n}\right) \Pr\{S(q)_n = i\} + \\
&\quad \frac{1}{q-1}\frac{n-(i-1)}{n} \Pr\{S(q)_n = i-1\}.
\end{aligned}$$

When $q = 2$, let $S_n \equiv S(2)_n$. The recurrence is simplified to

$$\Pr\{S_{n+1} = i\} = \frac{i}{n} \Pr\{S_n = i\} + \frac{n-(i-1)}{n} \Pr\{S_n = i-1\}. \tag{3.2}$$

Let sgn be the sign function:

$$\operatorname{sgn}(x) \equiv \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

We can show that

$$\Pr\{S_n = i\} = \frac{1}{2(n-1)!} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (i-k)^{n-1} \operatorname{sgn}(i-k)$$

satisfies the recurrence Equation (3.2):

$$\begin{aligned}
&\Pr\{S_{n+1} = i\} \\
&= \frac{i}{n} \Pr\{S_n = i\} + \frac{n-(i-1)}{n} \Pr\{S_n = i-1\} \\
&= \frac{1}{2(n-1)!} \left(\frac{i}{n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (i-k)^{n-1} \operatorname{sgn}(i-k) \right. \\
&\quad \left. + \frac{n-(i-1)}{n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (i-1-k)^{n-1} \operatorname{sgn}(i-1-k) \right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2n!}\left(i\sum_{k=0}^{n}(-1)^k\binom{n}{k}(i-k)^{n-1}\operatorname{sgn}(i-k) \right. \\
&\quad -(n+1-i)\sum_{k=1}^{n}(-1)^k\binom{n}{k-1}(i-k)^{n-1}\operatorname{sgn}(i-k) \\
&\quad \left. +(-1)^{n+1}(i-(n+1))^{(n+1)-1} \right) \\
&= \frac{1}{2n!}\left(i\sum_{k=1}^{n}(-1)^k\binom{n+1}{k}(i-k)^{n-1}\operatorname{sgn}(i-k) \right. \\
&\quad -\sum_{k=1}^{n}(-1)^k k\binom{n+1}{k}(i-k)^{n-1}\operatorname{sgn}(i-k) \\
&\quad \left. +i^n+(-1)^{n+1}(i-(n+1))^{(n+1)-1} \right) \\
&= \frac{1}{2n!}\left(\sum_{k=1}^{n}(-1)^k\binom{n+1}{k}(i-k)^{(n+1)-1}\operatorname{sgn}(i-k) \right. \\
&\quad \left. +i^n+(-1)^{n+1}(i-(n+1))^{(n+1)-1} \right) \\
&= \frac{1}{2n!}\sum_{k=0}^{n+1}(-1)^k\binom{n+1}{k}(i-k)^{(n+1)-1}\operatorname{sgn}(i-k).
\end{aligned}
$$

Theorem 44 derives an upper bound of the moment generating function of $S(2)_n$. Although in practice, an $\mathbb{H}$-graph is bootstrapped by a conversion from a complete graph, we assume, for our analysis, that the regular graph start from scratch and the first node picks its identifier uniformly at random (this will only weaken the results).

Theorem 44. *Let S_n be the number of $z_1 = 0$ nodes in a layered $\mathbb{H}$-graph of n nodes when identifiers are selected by* SAMPLING-IDENTIFIER$(q = 2, i = 1)$. *We have*

$$\mathrm{E}\left[e^{tS_n}\right] < \left(\frac{e^t-1}{t}\right)^n$$

for $n \geq 2$.

Proof. We shall prove by induction. Let $S_n = S(2)_n$. First, when $n = 2$, we have

$\Pr\{S_2 = 1\} = 1$ and $\mathrm{E}\left[e^{tS_2}\right] = e^t$. And since

$$\begin{aligned}
\left(\frac{e^t-1}{t}\right)^2 &= \frac{e^{2t}-2e^t+1}{t^2} \\
&= \frac{1}{t^2}\left(\sum_{i=0}^{\infty}\frac{(2t)^i}{i!} - 2\sum_{i=0}^{\infty}\frac{t^i}{i!} + 1\right) \\
&= \frac{1}{t^2}\sum_{i=2}^{\infty}\frac{(2t)^i - 2t^i}{i!} \\
&= \sum_{i=2}^{\infty}\frac{(2^i-2)t^{i-2}}{i!} \\
&= \sum_{i=0}^{\infty}\frac{(2^{i+2}-2)t^i}{(i+2)!} \\
&> \sum_{i=0}^{\infty}\frac{t^i}{i!} \\
&= e^t,
\end{aligned}$$

we have $\mathrm{E}\left[e^{tS_2}\right] < \left(\frac{e^t-1}{t}\right)^2$.

Assuming $\mathrm{E}\left[e^{tS_n}\right] < \left(\frac{e^t-1}{t}\right)^n$, we can show that

$$\begin{aligned}
&\mathrm{E}\left[e^{tS_{n+1}}\right] \\
&= \sum_{i=0}^{n+1}\Pr\{S_{n+1} = i\}\, e^{it} \\
&= \sum_{i=0}^{n+1}\Big(\Pr\{\, S_{n+1} = i \mid S_n = i\,\}\Pr\{S_n = i\} \\
&\qquad + \Pr\{\, S_{n+1} = i \mid S_n = i-1\,\}\Pr\{S_n = i-1\}\Big)\, e^{it} \\
&= \sum_{i=0}^{n+1}\Big(\Pr\{\, S_{n+1} = i \mid S_n = i\,\}\Pr\{S_n = i\} \\
&\qquad + e^t\Pr\{\, S_{n+1} = i+1 \mid S_n = i\,\}\Pr\{S_n = i\}\Big)\, e^{it} \\
&= \sum_{i=0}^{n+1}\left(\frac{i}{n}\Pr\{S_n = i\} + e^t\frac{n-i}{n}\Pr\{S_n = i\}\right)e^{it}
\end{aligned}$$

$$
\begin{aligned}
&= e^t \sum_{i=0}^{n} \Pr\{S_n = i\} e^{it} + \frac{1-e^t}{n} \sum_{i=0}^{n} i \Pr\{S_n = i\} e^{it} \\
&< e^t \left(\frac{e^t-1}{t}\right)^n + \frac{1-e^t}{n} \frac{d}{dt} \sum_{i=0}^{n} \Pr\{S_n = i\} e^{it} \\
&< e^t \left(\frac{e^t-1}{t}\right)^n + \frac{1-e^t}{n} \frac{d}{dt} \left(\frac{e^t-1}{t}\right)^n \\
&= e^t \left(\frac{e^t-1}{t}\right)^n + (1-e^t) \left(\frac{e^t-1}{t}\right)^{n-1} \frac{te^t - e^t + 1}{t^2} \\
&= \left(\frac{e^t-1}{t}\right)^{n+1}.
\end{aligned}
$$

□

Theorem 45 derives the limit of the variance of $S(q)_n$ as n becomes large.

Theorem 45.

$$\lim_{n\to\infty} \frac{\mathrm{Var}\left[S(q)_n\right]}{n} = \frac{(q-1)^2}{q^2(q+1)}.$$

Proof. Let $P_{n,i} = \Pr\{S(q)_n = i\}$ and $S_n = S(q)_n$. Let $\mathrm{E}\left[S_n^2\right] = \frac{n^2}{q^2} + v_n n$. We have

$$
\begin{aligned}
&\mathrm{E}\left[S_{n+1}^2\right] \\
&= \sum_{i=0}^{n+1} P_{n+1,i} i^2 \\
&= \sum_{i=0}^{n+1} \left(\Pr\{S_{n+1} = i \mid S_n = i\} P_{n,i} + \Pr\{S_{n+1} = i \mid S_n = i-1\} P_{n,i-1}\right) i^2 \\
&= \sum_{i=0}^{n} \left(\frac{i}{n} + \frac{n-i}{n}\frac{q-2}{q-1}\right) P_{n,i} i^2 + \sum_{i=1}^{n+1} \frac{1}{q-1} \frac{n+1-i}{n} P_{n,i-1} i^2 \\
&= \sum_{i=0}^{n} \left(i^2 \left(1 - \frac{2}{(q-1)n}\right) + i\frac{1}{q-1}\left(2 - \frac{1}{n}\right) + \frac{1}{q-1}\right) P_{n,i} \\
&= \left(\frac{n^2}{q^2} + v_n n\right)\left(1 - \frac{2}{n}\right) + \frac{n}{q}\left(2 - \frac{1}{n}\right) + \frac{1}{q-1} \\
&= \frac{(n+1)^2}{q^2} + v_n(n+1) - \frac{(q+1)v_n}{q-1} + \frac{q-1}{q^2}.
\end{aligned}
$$

Since $\mathrm{E}\left[S_{n+1}^2\right] = \frac{n^2}{q^2} + (n+1)v_{n+1}$, we have

$$\begin{aligned} v_{n+1} &= v_n + \frac{1}{n+1}\left(-\frac{(q+1)v_n}{q-1} + \frac{q-1}{q^2}\right) \\ &= v_n + \frac{q+1}{(q-1)(n+1)}\left(\frac{(q-1)^2}{q^2(q+1)} - v_n\right). \end{aligned}$$

Therefore, $v_{n+1} < v_n$ if $v_n > \frac{(q-1)^2}{q^2(1+q)}$. If, in addition, $\frac{q+1}{(q-1)(n+1)} < 1$, then $v_{n+1} > \frac{(q-1)^2}{q^2(1+q)}$.

When $q = 2$ and $n = 3$, we have $\Pr\{S(2)_3 = 1\} = \Pr\{S(2)_3 = 2\} = 1/2$, and thus

$$\mathrm{E}\left[S(2)_3^2\right] = \frac{5}{2} = \frac{3^2}{4} + \frac{3}{12}.$$

When $q = 3$ and $n = 2$, we have $\Pr\{S(3)_2 = 0\} = 1/3$, $\Pr\{S(3)_2 = 1\} = 2/3$, and thus

$$\mathrm{E}\left[S(3)_2^2\right] = 2/3 = \frac{2^2}{9} + \frac{2}{9}.$$

When $q \geq 4$ and $n = 1$, we have

$$\begin{aligned} \Pr\{S(q)_1 = 0\} &= 1 - 1/q, \\ \Pr\{S(q)_1 = 1\} &= 1/q. \end{aligned}$$

Thus $\mathrm{E}\left[S(q)_1^2\right] = 1/q$ and $\mathrm{Var}\left[S(q)_1\right] = 1/q - 1/q^2$. We can verify that $1/q - 1/q^2 > \frac{(q-1)^2}{q(q+1)}$ and $n + 1 > \frac{q+1}{q-1}$ for any $q \geq 4$ and $n = 1$.

Therefore, $\mathrm{Var}\left[S(q)_n\right]$ tends to $\frac{n(q-1)^2}{q^2(1+q)}$ in the limit.

□

The random variable $S(2)_n$ is a special case of "The Safety Campaign" in Friedman's urn model [31]. However, Friedman's expression of the moment generating function for his more general urn model was not in a closed form. It is also a special case ($p = -1$) of the balls and bins models with feedback in [22].

Freedman [29] gave an analysis of Friedman's urn when n is large. He shows that the distribution of $S(2)_n$ converges to the normal distribution with mean $n/2$ and variance $n/12$.

The following comparisons of $X(q)_n$ and $S(q)_n$ confirm our intuition that sampling

identifiers should lead to better load-balancing.

$$\mathrm{E}\left[X(q)_n\right] = n/q \qquad \mathrm{E}\left[S(q)_n\right] = n/q$$

$$\mathrm{Var}\left[X(q)_n\right] = \frac{(q-1)n}{q^2} \qquad \mathrm{Var}\left[S(q)_n\right] \approx \frac{(q-1)^2 n}{q^2(q+1)}$$

$$\mathrm{E}\left[e^{tX(2)_n}\right] = \left(\frac{e^t+1}{2}\right)^n \qquad \mathrm{E}\left[e^{tS(2)_n}\right] < \left(\frac{e^t-1}{t}\right)^n$$

$$\Pr\left\{X(2)_n < \frac{n}{4}\right\} < (0.883)^n \qquad \Pr\left\{S(2)_n < \frac{n}{4}\right\} < (0.665)^n$$

Applications with this self-balancing approach is more restrictive than the random identifiers approach. Since the identifier of a node depends on the graph topology when it joins, the identifier cannot be derived from the "name" of the node. Therefore, a node has to announce its identifier before others are able to locate it. Certain applications might find this limitation undesirable.

3.5 Related Work

Napster [74] is a well-known file-sharing system based on a centralized directory. Gnutella [39] is distributed for both indexing and file-sharing. However, it is based on ad hoc protocols that could lead to unbalanced topology and bottlenecks easily. Gnutella searches with controlled broadcast which is similar to SEARCH-TREEWALK. Freenet [30] gives high priority to anonymity. Freenet uses depth-first-search with caching to improve performance.

The fault-tolerant content addressable network by Saia et al. [86] is robust against adversarial removals of nodes and supports strong guarantees of data availability. However, it is not very dynamic because it relies on a static butterfly network of supernodes. It also has higher message and space complexities. The adversary is limited—during any period of time the number of new nodes must exceed the number of deleted nodes.

CAN [83], Chord [92], Pastry [85], and Tapestry [99] are recently proposed systems designed mainly for distributed storage and lookup service. These systems have $O(\log n)$ routing time and $O(\log n)$ neighbor size.

Chord [92] is a recent design of a distributed lookup service. Chord uses a consistent hash function [54] to map the nodes randomly on a virtual ring. In the default protocol each node points to $O(\log n)$ nodes. (In a variant with better load-balancing, each node points to $O(\log^2 n)$ other nodes.) A look up takes $O(\log n)$ times, and nodes joining or leaving the network take $O(\log^2 n)$ messages. Therefore, Chord and our layered $\mathbb{H}$-graphs features essentially the same asymptotic results except that leaving a node in a layered $\mathbb{H}$-graph takes only $O(\log n)$ messages. Both Chord and layered $\mathbb{H}$-graphs have extensions to improve look up time to $O(\log n/\log\log n)$, albeit via different techniques as well.

Content-Addressable Network (CAN) [83] is another hash-table based solution. Instead of hashing the nodes to points on a ring, CAN hashes the nodes to points on a d-dimensional torus. Each node owns a 'zone' in the torus and stores the items with hashed value in this zone. The average routing path is $(d/4)(n^{1/d})$ hops, where each node maintains $2d$ neighbors. By setting $d = (\log_2 n)/2$, this becomes $(\log_2 n)/2$ hops and $\log_2 n$ neighbors. In this case, CAN has the same asymptotic performance as Chord and layered $\mathbb{H}$-graphs. However, CAN's protocol for node departures is complicated and could be costly.

Comparing to these systems, it is likely that layered $\mathbb{H}$-graphs will require more overheads in terms of larger constant factors in the number of neighborhoods per node and the number of messages required during JOIN. However, we note that layered $\mathbb{H}$-graphs are particularly efficient and robust for node departures. layered $\mathbb{H}$-graphs do not need a background process to fix broken routing pointers, because there are no explicit routing tables. Fault-tolerance support is relatively easy to implement because we can simply apply ring-based solutions on the Hamiltion cycles independently. In addition, layered $\mathbb{H}$-graphs support a unique 'self-balancing' approach by sampling the current state of the network.

Loguinov et al. [66] compare the properties of de Bruijn topologies with other existing topologies, mostly with Chord and CAN. The metrics they used include diameter, eigenvalue, and expansion. In those metrics, our H-graphs lie between the de Bruijn and other topologies that they had compared in the paper. This is because de Bruijn is an ideal construction with best possible diameter and expansion. Another interesting result is that they found that both Chord and CAN are not expanders. One reason that they can obtain even better eigenvalues than our $\mathbb{H}$-graphs is that they use directed graphs instead of undirected graphs. This make their graphs more efficient in terms of the number of edges used, as we count each undirected edges twice in our work.

Although de Bruijn graphs are the ideal expander graphs, in practice it is probably not feasible to construct such graphs exactly. In [66], a dynamic construction approximates the de Bruijn graphs. It is not clear how the effect of the approximation would affect the constants of the performance measures.

Kaashoek and Karger [50] proposed Koorde as a de Bruijn version of Chord. Koorde is degree-optimal as it can look up in $O(\log n)$ steps with $O(1)$ neighbors, or in $O(\log n/\log\log n)$ steps with $O(\log n)$ neighbors.

Chapter 4

Resource Discovery and Connectivity

4.1 Introduction

The resource discovery problem of networked machines seeking and learning about each others was introduced by Harchol-Balter, Leighton, and Lewin in [45]. In a distributed network, if machine u knows the address of machine v, machine u can connect machine to v and inform it of other machines known to u. A resource discovery algorithm specifies how the machines should communicate with each other so that, eventually, each machine will be aware of all other machines.

There are three performance measures for a resource discovery algorithm:

1. time complexity – number of time steps taken;
2. message complexity – number of messages sent (called connection communication complexity in [45]) ; and
3. pointer complexity – number of pointers (machine addresses) passed (called pointer communication complexity in [45]).

Prior algorithms for resource discovery include the Flooding algorithm, the Swamping algorithm, and the Random Pointer Jump algorithm (all described in Harchol-Balter *et al.* [45]). However, it was shown in [45] that these algorithms do not perform well in certain network topology.

Their paper [45] introduced the NAME-DROPPER algorithm: During each round, each machine picks a neighbor randomly and passes the neighbor all its known pointers. It was shown that the machines can learn about one another in $O(\log^2 n)$ time, $O(n \log^2 n)$ message complexity, and $O(n^2 \log^2 n)$ pointer complexity, all with high probability.

Kutten, Peleg, and Vishkin [60] proposed a deterministic resource discovery algorithm with $O(\log n)$ time $O(n \log n)$ message complexity, and $O(n^2 \log n)$ pointer complexity.

There has been much research on connected component algorithms on the Parallel Random Access Machine (PRAM) model. The algorithm KUTTEN-PELEG-VISHKIN is an extension of a parallel algorithm by Shiloach and Vishkin [89]. Awerbuch and Shiloach [10] also proposed a parallel connected component using similar techniques. Greiner [41] compared these two algorithms with an algorithm based on Reif [84] and Philips [79]. More recently, Karger, Nisan, and Parnas [55], and Halperin and Zwick [44] gave fast randomized connected component algorithm for this computation model.

All algorithms considered in [45] and this chapter assume synchronous networks. Kutten and Peleg [59], and Abraham and Dolev [1] studied resource discovery algorithms on asynchronous networks.

In this chapter, we introduce two related randomized algorithms, ABSORPTION and ASSIMILATION. Algorithm ABSORPTION's performance analyses assume strongly-connected graphs. Both algorithms run in $O(\log n)$ time with $O(n \log n)$ message complexity. The pointer complexities of ABSORPTION and ASSIMILATION are $O(n^2)$ and $O(n^2 \log n)$ respectively. A variant of ABSORPTION with $O(\log^2 n)$ running time achieves $O(n)$ message complexity.

We observe that resource discovery can be considered in two parts: from weakly-connected graph to strongly-connected graph, and then from strongly-connected graph to complete graph.

In Section 4.2, we describe a graph-theoretic model for the resource discovery problem. We introduce and analyze ABSORPTION algorithm in Section 4.3. Section 4.4 discusses some variants of ABSORPTION. We study ASSIMILATION algorithm in Section 4.5 and conclude Section 4.6.

4.2 Resource Discovery

In this section, we describe a model of the resource discovery problem by studying distributed algorithms that evolve a connected directed graph into a complete graph. We will also give simple lower bounds on the three performance measures defined in Section 4.1.

A network can be modeled by a directed graph (V, E), such that each machine is a node in V. If machine u knows about machine v, then there is an edge $(u, v) \in E$.

For any node u, we let $\Gamma(u)$ be the set of machines known to u:

$$\Gamma(u) \equiv u \cup \{v \in V \mid (u, v) \in E\}.$$

A *message* is a set of nodes. A node u can send messages to any node in $\Gamma(u)$. When a node u receives a message M from v, node u's known set $\Gamma(u)$ is updated to $\Gamma(u) \cup M$. In this case, we also say node u passes a set M of pointers to node v.

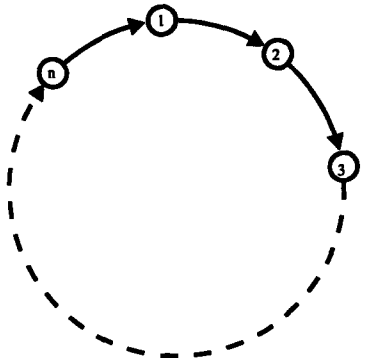

Figure 4.1: A worst-case strongly-connected graph for Remark 46.

The goal of a resource discovery algorithm is to evolve a connected graph into a complete graph. In a complete graph, every node knows all other nodes: $\Gamma(u) = V$ for all $u \in V$. The algorithm needs to be distributed in the sense that each node runs the algorithm without knowledge of the global state. For the convenience of analysis, we assume that a global clock is available such that the algorithm on each node can run in discrete steps.

Remark 46. *Any resource discovery algorithm on a strongly-connected graph requires at least* $\log_2(n-1)$ *time steps,* n *messages, and* $n(n-2)$ *pointers passed in the worst case.*

Proof. Consider Figure 4.1, in which node i knows about node $i+1 \bmod n$ only.

Time In the worst case, $\log_2(n-1)$ time steps are needed because the graph has diameter $n-1$, but after each time step, the diameter of the graph can at most reduced by half.

Messages We need at least n connections because each node must receive at least one message.

Pointers Each node needs to receive at least $n-2$ pointers because each node knows two nodes initially, but needs to know n nodes at the end. Therefore, a total of $n(n-2)$ pointers have to be passed. □

4.3 The ABSORPTION Algorithm

We now describe the algorithm ABSORPTION.

For any node v, let $l(v)$ denote the leader of v. Naturally, we have $l(v) \in \Gamma(v)$ for any node v. A node v is a leader if and only if $l(v) = v$.

A graph can be partitioned into disjoint clusters such that two nodes are in the same cluster if they share the a leader. Each cluster contains exactly one leader.

For any leader u, let $\mathcal{C}(u)$ be the set of nodes that have u as their leader:

$$\mathcal{C}(u) \equiv \{w \mid l(w) = u\}.$$

We will show in Lemma 47 that a leader's known set is a superset of the union of its members' known sets.

In the beginning, there are n clusters such that every node is the leader of its single-node cluster. Each round consists of the following three Steps.

1. Each leader u becomes *active* with probability p if $\Gamma(u) \neq \mathcal{C}(u)$. If leader u is active, it randomly chooses a node $v \in \Gamma(u) \setminus \mathcal{C}(u)$ and then passes the set $\Gamma(u)$ to v. We call these messages seek messages.

2. Each node v that was contacted by some leaders in the previous Step now passes the seek messages to its leader $l(v)$.

3. (a) Any active leader u remains idle in this Step.

   (b) Any inactive leader that had not received any seek message in Step 2 remains idle in this Step.

   (c) For each leader u inactive in Step 1 and had received $k > 0$ messages in Step 2, we will construct a bigger cluster led by u. Let $v_1, \ldots, v_k$ be the leaders that had contacted some members of $\mathcal{C}(u)$ in Step 1.

   The new cluster is a merge of the original cluster of u and the clusters of these retiring leaders $v_1, \ldots, v_k$. After the merge, $\mathcal{C}(u)$ is updated to contain all these clusters:

   $$\mathcal{C}'(u) = \mathcal{C}(u) \cup \bigcup_{1 \leq i \leq k} \mathcal{C}(v_i),$$

   and $\Gamma(u)$ will absorb the pointers received from $v_1, \ldots, v_k$:

   $$\Gamma'(u) = \Gamma(u) \cup \bigcup_{1 \leq i \leq k} \Gamma(v_i).$$

   Leader u informs all nodes $w \in \mathcal{C}'(u)$ of their new leader u and $\Gamma'(u) \setminus \Gamma(w)$. We call these messages update messages.

In the first Step, each node independently decides to be active or inactive. This is somewhat similar to the symmetry-breaking techniques used in parallel algorithms [47].

Now we proceed to show that ABSORPTION terminates and produces a complete graph. We will also analyze the complexities of ABSORPTION on strongly-connected graphs.

First, we need to extend our notation of Γ to take in a set of nodes:

$$\Gamma(W) \equiv \bigcup_{w \in W} \Gamma(w).$$

Lemma 47 states that the known set of a leader is the union of the known sets of the members of its cluster.

Lemma 47. *During the* ABSORPTION *algorithm, we have*

$$\Gamma(\mathcal{C}(u)) = \Gamma(u)$$

for any leader u.

Proof. Initially, each cluster consists of a single node. Therefore, Equation (47) is satisfied because $\mathcal{C}(u) = \{u\}$ for all $u \in V$.

We now need to show that Equation (47) is maintained during any merging in Step 3. During any round, assume that leader u becomes the new leader of the clusters of $v_1, v_2, \ldots, v_k$. Let $\Gamma'(u)$ and $\mathcal{C}'(u)$ be the known set and the cluster of u after the merge.

It is sufficient to show that $\Gamma'(\mathcal{C}'(u)) \supseteq \Gamma'(u)$.

Since u receives the known pointers from $v_1, v_2, \ldots, v_k$, we have

$$\Gamma'(u) = \Gamma(u) \cup \bigcup_{i=1}^{k} \Gamma(v_i).$$

Applying our inductive hypothesis, we have

$$\Gamma'(u) = \Gamma(\mathcal{C}(u)) \cup \bigcup_{i=1}^{k} \Gamma(\mathcal{C}(v_i)).$$

We also have

$$\mathcal{C}'(u) = \mathcal{C}(u) \cup \bigcup_{i=1}^{k} \mathcal{C}(v_i).$$

At last, since $\Gamma'(A) \supseteq \Gamma(A)$ for any set A,

$$\begin{aligned}
\Gamma'(\mathcal{C}'(u)) &= \Gamma'\left(\mathcal{C}(u) \cup \bigcup_{i=1}^{k} \mathcal{C}(v_i)\right) \\
&\supseteq \Gamma\left(\mathcal{C}(u) \cup \bigcup_{i=1}^{k} \mathcal{C}(v_i)\right) \\
&= \Gamma(\mathcal{C}(u)) \cup \bigcup_{i=1}^{k} \Gamma(\mathcal{C}(v_i)) \\
&= \Gamma'(u).
\end{aligned}$$

□

Lemma 48. *At the end of each round, for any node v,*

$$\Gamma(v) = \Gamma(l(v)).$$

Proof. This follows from messages containing $\Gamma'(u) \setminus \Gamma(v)$ from the new leader before the end of each round. □

Lemma 49. *In each round of the* ABSORPTION *algorithm, each leader u where $\Gamma(u) \setminus \mathcal{C}(u) \neq \emptyset$ is retired with probability at least $p(1-p)$.*

Proof. A leader u is retired in a round if and only if

1. leader u is active; and
2. leader u selects a node v such that leader $l(v)$ is not active.

By Lemma 51, leader u can always pick a node v not in the cluster of u.

The probability that leader u being active in any round is p. The probability that the leader $l(v)$ being inactive is $1-p$. Therefore the probability that leader u is retired in any given round is $p(1-p)$. □

Theorem 50. *On a weakly-connected graph, the* ABSORPTION *algorithm terminates in $O(n)$ rounds with high probability.*

Proof. In each round there exists at least one leader u where $\Gamma(u) \setminus \mathcal{C}(u) \neq \emptyset$. Thus in each round, with probability at least $p(1-p)$, one leader retires. Let X be the number of successes in kn trials with success probability $p(1-p)$. Let $k = 2/p(1-p)$. The probability that X is smaller than n is

$$\Pr\{X < n\} = \Pr\{X < (1/2)kp(1-p)n\} < e^{-n/4}.$$

□

If the graph is strongly-connected, than any leader knows about at least one node not among its own cluster.

Lemma 51. *Running* ABSORPTION *on a strongly-connected graph, unless there is only one leader left,*

$$\Gamma(u) \neq \mathcal{C}(u)$$

for any leader u.

Proof. Assume there is a leader u such that $\Gamma(u) = \mathcal{C}(u)$ and there is another leader v where $v \neq u$. By Lemma 47, for any node $w \in \mathcal{C}(u)$, we have $\Gamma(w) \subseteq \Gamma(\mathcal{C}(l(w))) = \Gamma(u)$. This means that none of the nodes in $\mathcal{C}(u)$ points to any node not in $\mathcal{C}(u)$. Thus there is no path from any node in $\mathcal{C}(u)$ to v, contradicting our assumption that the graph is strongly-connected in the beginning. □

Using Lemma 51, we can show that the expected number of leaders in the graph is reduced by a constant factor in each round.

Now we are ready to state our main results.

Theorem 52. *On a strongly-connected graph, the* ABSORPTION *algorithm terminates in*

$$2\log_{\frac{1}{1-p+p^2}} n$$

rounds with probability greater than $1 - 1/n$.

Proof. The main loop of the ABSORPTION algorithm terminates when we are left with only one leader.

Let G be any strongly connected directed graph with n nodes.

We start with n leaders. By Lemma 49, at each round, any leader has a probability of $p(1-p)$ of retiring if there are more than 1 leader in the graph. Consider any leader u, the probability that u is not retired after $2\frac{\log n}{\log \frac{1}{1-p+p^2}}$ rounds is

$$(1-p+p^2)^{2\log_{\frac{1}{1-p+p^2}} n} = \frac{1}{n^2}.$$

Therefore, the probability that there are more than one leader after $2\log_{\frac{1}{1-p+p^2}} n$ rounds is at most $1/n$. □

Corollary 53. *With high probability, the message complexity of the* ABSORPTION *algorithm is* $O(n \log n)$ *and the pointer complexity of the* ABSORPTION *algorithm is* $O(n^2 \log n)$.

Next, we derive the upper bounds on the constants of the asymptotic bounds.

Theorem 54. *Let* $q = 1/(1-p+p^2)$. *The expected time steps of the* ABSORPTION *algorithm is* $3\log_q n$; *the expected total number of messages is at most* $n\log_q n + 2n/(1-p)$; *and the expected total number of pointers passed is at most* $(1 + 2/(1-p))n(n-1)$.

Proof. By Lemma 49, the number of leaders is reduced by $p(1-p)$ in each round. Therefore, there are $\log_q n$ rounds in expectation, where $q = 1/\log(1-p+p^2)$. Since each round takes 3 steps, the expected total running time is $3\log_q n$ steps.

As introduced in Section 4.3, there are two types of messages:

1. the seek messages are those sent by leader u in Step 1 to some node not in $C(u)$, and the forwardings of those messages in Step 2;

2. the update messages are those sent by a leader u in Step 3 to members of the new merged cluster.

We can count the number of messages seek messages originated from each node.

Let u be any node. Let X_0 is the total number of messages originated from node u. Let X_i be the total expected number of seek messages originated from node u after the ith round. We have

$$\begin{aligned} X_i &\le \Pr\{u \text{ is inactive in round } i\} X_{i+1} \\ &\quad + \Pr\{u \text{ is active in round } i\} \left(2 + \Pr\{u \text{ does not retire}\} X_{i+1}\right) \\ &= (1-p)X_{i+1} + p(2 + pX_{i+1}) \\ &= 2p + (1 - p + p^2)X_{i+1}. \end{aligned}$$

Therefore, we have

$$X_0 = \frac{2p}{p - p^2} = \frac{2}{1-p}.$$

Thus there are $2n/(1-p)$ seek messages. The size of each seek message is at most $n-1$, therefore, the expected total number of pointers in seek messages is at most $2n(n-1)/(1-p)$.

In Step 3, the new leader informs its members in update messages. Since the clusters are disjoint, the number of update messages passed in each round is at most n. Therefore, there are $n \log_q n$ messages in total. In these update messages, any node only receives addresses not known before. Therefore, the total number of pointers in all update messages is at most $n(n-2)$.

□

Corollary 55. *When $p = 1/2$. The expected time steps of the* ABSORPTION *algorithm is $3\log_{4/3} n$; the expected total number of messages is at most $n\log_{4/3} n + 4n$; and the expected total number of pointers passed is at most $4n(n-1) + n(n-2)$.*

To estimate how far we are from the optimal algorithm, we can compare our bounds derived in Theorem 54 with the best possible centralized algorithm.

Corollary 56. *When $p = 1/2$, the ratio of the expected running time of* ABSORPTION *to the worst-case running time of the optimal centralized algorithm is at most 11.5 for strongly-connected network with at least 3 nodes. The ratio approaches $3\log 2/\log(4/3) \approx 7.23$ as network size grows.*

If the given graph is weakly-connected, we can first run the NAME-DROPPER algorithm for $O(\log n)$ rounds. According to [45], the NAME-DROPPER algorithm makes $O(n)$ connections per round, and passes $O(n^2)$ pointers per round in high probability.

At last, we demonstrate a nice property of ABSORPTION— equal probability leader election.

Theorem 57. *On a strongly-connected graph with n nodes, each node has probability $1/n$ to become the ultimate leader of the* ABSORPTION *algorithm.*

Proof. We shall prove by induction on the number of leaders. We will show that in any graph with n leaders, each leader has probability $1/n$ to become the ultimate leader.

For the base case, if there is only one leader, than it has probability 1 to become the ultimate leader.

Consider a graph with n leaders, where $n > 1$. In each round, each leader has the same probability $p(1-p)$ of retiring. Assume that k leaders are retired in the next round. By symmetry, the probability that any leader does not retire is $(n-k)/n$. Therefore the overall probability of becoming the ultimate leader for any leader is

$$\frac{n-k}{n}\frac{1}{n-k} = \frac{1}{n}.$$

□

4.4 Variants of ABSORPTION

In this section we discuss two variants of the ABSORPTION algorithm.

4.4.1 Optimizing Pointers

We can see from Table 4.2 that, on a weakly-connected graph, the subroutine of evolving the graph to be strongly-connected is the bottleneck of the ABSORPTION algorithm's pointer complexity. We describe a method to improve the pointer complexity at the cost of higher message complexity. Instead of NAME-DROPPER, we can use the following simple algorithm to obtain a strongly-connected graph.

DOUBLE-LINK: Each node sends a message about itself to each node in its known set. Thus, after one time step, the graph is strongly-connected.

Algorithm DOUBLE-LINK takes 1 time step, sends at most $n(n-1)$ messages, and passes at most $n(n-1)$ pointers. On a weakly-connected graph, if DOUBLE-LINK is executed before ABSORPTION, the overall pointer complexity is improved to $O(n^2)$ in expectation. However, the message complexity would degrade to $O(n^2)$ with high probability.

4.4.2 Optimizing Messages

We describe a variant of the ABSORPTION algorithm that reduces the expected message complexity of the ABSORPTION algorithm to $O(n)$, at the cost of higher time complexity. We call this variant "ABSORPTION-M", for optimizing the message complexity.

We now describe the changes required on the ABSORPTION algorithm.

In Step 3, the new leader u does not notify all nodes in the new cluster. Instead, leader u only tells the retiring leaders $v_1, \ldots, v_k$ to update their leader pointers. The retired leader will forward future seek messages to its new leader. Therefore, during Step 2, a retired leader u will forward the seek messages received from its members to u's leader. The side effect of this modification is that Step 2 can no longer be finished in a single time step. In fact, during the ith round, Step 2 needs $i-1$ time steps for the chain of retired leaders to forward the seek messages to the current leaders.

There will be a final stage where the last remaining leader notifies all nodes with the entire graph.

Theorem 58. *With high probability, algorithm* ABSORPTION-M *runs in* $O(\log^2 n)$ *time.*

Proof. This follows immediately from Theorem 52 that ABSORPTION finishes in $O(\log n)$ rounds in high probability, and that the ith round in ABSORPTION-M takes $i-1+2 = i+1$ time steps. □

Theorem 59. *Let* $q = 1/(1-p+p^2)$. *Running the* ABSORPTION-M *algorithm on a strongly-connected graph, the expected time steps is* $\frac{1}{2}\log_q^2 n + \frac{3}{2}\log_q n + 1$; *the expected total number of messages is at most* $\left(\frac{1+p^2-p}{p(1-p)^2} + 2\right) n - 2$; *and the expected total number of pointers passed is at most* $\left(\frac{1+p^2-p}{p(1-p)^2} + 1\right) n(n-1)$.

Proof. Applying the analysis of ABSORPTION, the expected number of rounds of ABSORPTION-M is $\log_q n$. Since the ith round takes $i+1$ time steps and final stage takes 1 time step, the total expected time is

$$1 + \sum_{i=1}^{\log_q n} (i+1) = \frac{\log_q n}{2}(\log_q n + 3) + 1$$
$$= \frac{1}{2}\log_q^2 n + \frac{3}{2}\log_q n + 1.$$

As introduced in Section 4.3, there are two types of messages:

1. the seek messages are those sent by leader u in Step 1 to some node not in $C(u)$, and the forwardings of those messages in Step 2;
2. the update messages are those sent by a leader u in Step 3 to notify the retiring leaders that u is their new leader.

We can count the number of messages seek messages originated from each node.

Let u be any node. Let X_0 is the total number of messages originated from node u. Let X_i be the total expected number of seek messages originated from node u after the ith round.

Since,

$$\begin{aligned} X_i &= \Pr\{u \text{ is inactive in round } i\} X_{i+1} \\ &\quad + \Pr\{u \text{ is active in round } i\} \left(i + \Pr\{u \text{ does not retire}\} X_{i+1}\right) \\ &= (1-p)X_{i+1} + p(i + pX_{i+1}) \\ &= pi + (1 - p + p^2)X_{i+1}. \end{aligned}$$

Therefore, we have

$$X_0 = \frac{1+p^2-p}{p(1-p)^2}.$$

Since each seek message contains at most $n-1$ pointers, the expected total number of pointers is $\frac{1+p^2-p}{p(1-p)^2}n(n-1)$.

In the final stage, there are $n-1$ messages and $(n-1)^2$ pointers.

In summary, the expected total number of messages is at most $\frac{1+p^2-p}{p(1-p)^2}n + (n-1) + (n-1) = \left(\frac{1+p^2-p}{p(1-p)^2} + 2\right)n - 2$; and the expected total number of pointers is at most

$$\begin{aligned} &\frac{1+p^2-p}{p(1-p)^2}n(n-1) + (n-1) + (n-1)^2 \\ &= \frac{1+p^2-p}{p(1-p)^2}n(n-1) + n(n-1) \\ &= \left(\frac{1+p^2-p}{p(1-p)^2} + 1\right)n(n-1). \end{aligned}$$

□

Corollary 60. *When $p = \frac{3-\sqrt{5}}{2}$, the message complexity of the* ABSORPTION-M *algorithm is $(5+\sqrt{5})n - 2$. The time complexity is $\frac{1}{2}\log_q^2 n + \frac{3}{2}\log_q n + 1$, where $q = \frac{3+\sqrt{5}}{4}$. The message complexity is $(4+\sqrt{5})n(n-1)$.*

4.4.3 Optimizing Time

We can also optimize the constant of the time complexity with the cost of slightly larger constant for the message complexity.

We introduce another variant, the ABSORPTION-T algorithm. In Step 1, an active leader u notifies all nodes in $\mathcal{C}(u)$. Step 2 is skipped because all nodes in $\mathcal{C}(u)$ already knows if that cluster is active in that round. In Step 3, a inactive node contacted in Step 1 send update messages to the nodes of the cluster being merged. In this variant, we also need a final stage where the last leader broadcasts the entire graph.

Theorem 61. *Let* $q = 1/(1-p+p^2)$. *The expected time steps of the* ABSORPTION-T *algorithm is* $2\log_q n + 1$*; the expected total number of messages is at most* $2n\log_q n + (1 + 2/(1-p))n$*; and the expected total number of pointers passed is at most* $(1+2/(1-p))n(n-1) + n\log n$.

Proof. Let $q = 1/(1-p^2+p)$.

We only need to consider the difference between the complexities between ABSORPTION and ABSORPTION-T.

First, there is an extra broadcast of the activeness decision in each round. There are at most n such messages in each round.

Instead of the $n(n-2)$ pointers for the update messages in total, we now have n pointers for the update messages for each round. However, at the final stage, there are $n-1$ messages and $(n-1)^2$ pointers.

The total difference in messages is $n\log_q n + n - 1$. The total difference in pointers is $n\log_q n + (n-1)^2 - n(n-2)$. □

	Time	Messages	Pointers
ABSORPTION	$3\log_{4/3} n$	$n\log_{4/3} n + 4n$	$5n(n-1)$
ABSORPTION-M	$\frac{1}{2}\log^2_{\frac{3+\sqrt{5}}{4}} n \frac{3}{2}\log_{\frac{3+\sqrt{5}}{4}} n + 1$	$(5+\sqrt{5})n$	$(4+\sqrt{5})n(n-1)$
ABSORPTION-T	$2\log_{4/3} n + 1$	$2n\log_{4/3} n + 5n$	$5n(n-1) + n\log n$

Table 4.1: Performance of ABSORPTION, ABSORPTION-M, and ABSORPTION-T on strongly-connected graphs.

4.5 The ASSIMILATION Algorithm

While ABSORPTION is an efficient algorithm with both time complexity and pointer complexity very close to be optimal, the performance of ABSORPTION depends on the graph being strongly-connected. Coupling ABSORPTION with a strongly-connecting algorithm could tricky in implementation and is not an entirely satisfactory solution.

In this section we introduce algorithm ASSIMILATION that achieves $O(\log n)$ time on weakly-connected graphs. Algorithm ASSIMILATION is a "union" of NAME-DROPPER and ABSORPTION, because it contains the properties of both algorithms. There are five Steps in each round of ASSIMILATION:

1. For each leader u, if $\Gamma(u) \neq C(u)$, u becomes *active* for this round with probability $p > 1/2$.

If leader u is active, it randomly chooses up to $|\mathcal{C}(u)|$ nodes in $\Gamma(u) \setminus \mathcal{C}(u)$ and then passes the set $\Gamma(u)$ to them. These are the seek messages.

2. Each node v contacted by some leaders in the previous Step now passes the seek messages to its leader $l(v)$.

3. Let u be any inactive leader that had received $k > 0$ seek messages in Step 2. Let $v_1, \ldots, v_k$ be the leaders that had contacted some members of $\mathcal{C}(u)$ in Step 1. The known set $\Gamma(u)$ will absorb the pointers received from $v, \ldots, v_k$:

$$\Gamma'(u) = \Gamma(u) \cup \bigcup_{1 \le i \le k} \Gamma(v_i).$$

Leader u sends notify messages to $v_1, \ldots, v_k$ containing pointer to u only.

4. Each leader v received notify messages from $u_1, \ldots, u_m$ in Step 3 returns reply messages that partition $\mathcal{C}(v)$. (Our analysis does not depend on the way that $\mathcal{C}(v)$ is partitioned.)

5. For any leader u having received reply messages, the new cluster is a merge of the original cluster of u and the reply messages received from retiring leaders $v_1, \ldots, v_k$. After the merge, $\mathcal{C}(u)$ is updated to contain all these clusters:

$$\mathcal{C}'(u) = \mathcal{C}(u) \cup \bigcup_{1 \le i \le k} v_i.\text{reply}.$$

Leader u informs all nodes $w \in \mathcal{C}'(u)$ of their new leader u and $\Gamma'(u) \setminus \Gamma(w)$. These are the update messages.

Lemma 62 ([45]). *Let a, b be two neighboring nodes in G. They will be in the same connected component after $O(c \log n)$ rounds of* ASSIMILATION *with probability $1 - \frac{1}{n^{O(c)}}$.*

Proof. This proof is an adaptation of the proof of Lemma 2.2 in [45].

Without loss of generality, we assume there is an edge from a to b in the initial graph.

Let set A be the set of nodes having pointers to both a and b. For any node $u \in A$, we have $\{a, b\} \subset \Gamma(u)$. Initially, we have $a \in A$. Our goal is to bound the time before some node in A contacts b successfully. Because once that happens, a and b will be strongly connected.

The key observation is that algorithm ASSIMILATION simulates NAME-DROPPER. In each round, with probability p, each leader u randomly picks $|\mathcal{C}(u)|$ nodes. This is better than having each node in $\mathcal{C}(u)$ picking independently because the leader does not pick a node in $\mathcal{C}(u)$, which will be useless, nor does have duplicate picks. Therefore, we can apply

the analysis of NAME-DROPPER and only adjust certain probabilities by factors of p or $(1-p)$.

Let $d(u)$ be number of outgoing edges of u.

Let A' be the set of active nodes in A. Since $|A'|$ is a binomial variable with expectation $p|A|$, we have

$$\Pr\{X < (1/2p)p|A|\} \leq e^{-|A|(1-1/2p)^2/2} \leq e^{-(1-1/2p)^2/2}$$

Since $p > 1/2$, we have $e^{-(1-1/2p)^2/2} < 1$. Therefore, with probability at least $1 - e^{-(1-1/2p)^2/2}$, we have $|A'| \geq |A|/2$.

$$\begin{aligned}\Pr\{b \text{ is contacted by a node in } A' \text{ while inactive}\} &= 1-(1-p)\prod_{u\in A'}\left(1-\frac{1}{d(u)}\right)\\ &\geq 1-(1-p)e^{-\sum_{u\in A'}\frac{1}{d(u)}}.\end{aligned}$$

If

$$\Pr\{b \text{ is contacted by a node in } A' \text{ while inactive}\} < 1-(1-p)e^{-1/16},$$

then,

$$\sum_{u\in A'}\frac{1}{d(u)} \leq \frac{1}{16}.$$

This implies that there are at least $|A'|/4$ with degree greater than $4|A'|$. Then there are at least $|A|/2$ nodes with degree greater than $2|A|$ with such probability.

Since each of these nodes points to at least $|A|$ nodes not in A, there is a probability of $1/4$ that each of these nodes points to a new node. The expected number of new nodes is then at least $|A|/8$. Following the proof of Lemma 2.2 in [45], with a bounded Markov argument, it can be shown that with probability at least $1/15$, the size of A grows by $|A|/16$.

In summary, during each round, either there is a probability $\left(1-e^{-(1-1/2p)^2/2}\right)(1-p)e^{-1/16}$ that b is contacted and merged, or there is probability $\left(1-e^{-(1-1/2p)^2/2}\right)\frac{1}{15}$ that $|A|$ grows by a fraction of $1/16$. Thus, after $O(c\log n)$ rounds, the probability that b is still not contacted and merged is at most $\frac{1}{n^{O(c)}}$. □

Theorem 63. *The* ASSIMILATION *algorithm terminates after* $O(\log n)$ *rounds with high probability.*

Proof. By Lemma 62, for any $a, b \in G$ where a points to b in the beginning, node b will point to a in $O(c\log n)$ rounds with probability at least $1-\frac{1}{n^{O(c)}}$. Since there are at most $n(n-1)/2$ such pairs in G, we can pick c such that with high probability, the graph is strongly-connected in $O(c\log n)$ rounds.

	Time	Messages	Pointers
NAME-DROPPER [45]	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n^2 \log^2 n)$
KUTTEN-PELEG-VISHKIN [60]	$O(\log n)$	$O(n \log n)$	$O(n^2 \log^2 n)$
ABSORPTION	$O(\log n)$	$O(n \log n)$	$O(n^2)$
ASSIMILATION	$O(\log n)$	$O(n \log n)$	$O(n^2 \log n)$

Table 4.2: Performance of NAME-DROPPER, KUTTEN-PELEG-VISHKIN, ABSORPTION, and ASSIMILATION on three complexity measures. The bounds on row "ABSORPTION" assume a strongly-connected graph.

	Time	Messages	Pointers
NAME-DROPPER	$O(\log n)$	$O(n \log n)$	$O(n^2 \log n)$
DOUBLE-LINK	$O(1)$	$O(n^2)$	$O(n^2)$
ASSIMILATION	$O(\log n)$	$O(n \log n)$	$O(n^2 \log n)$

Table 4.3: Asymptotic bounds of strong-connecting algorithms.

We note that Theorem 52 is also applicable for ASSIMILATION on strongly-connected graphs. Therefore, once the graph is strongly-connected, the graph becomes a single cluster in $O(\log n)$ time with high probability. □

Corollary 64. *The message complexity of algorithm* ASSIMILATION *is* $O(n \log n)$. *The pointer complexity of algorithm* ASSIMILATION *is* $O(n^2 \log n)$.

Proof. The theorem follows from the observation that there are $O(n)$ messages and $O(n^2)$ pointers in each Step of the ASSIMILATION algorithm. □

4.6 Concluding Remarks

In this chapter, we describe and analyze the ABSORPTION algorithm, variants ABSORPTION-M and ABSORPTION-T, and the ASSIMILATION algorithm. We proved strong bounds on the performance measures. Table 4.2 compares the performance of ABSORPTION and ASSIMILATION with NAME-DROPPER and KUTTEN-PELEG-VISHKIN. We note that these algorithms except KUTTEN-PELEG-VISHKIN are randomized algorithms. Both ABSORPTION and ASSIMILATION are Las Vegas algorithms while NAME-DROPPER is a Monte Carlo algorithm.

A weakness of ABSORPTION and ASSIMILATION is their reliance on few machines (those leaders that retire late) to distribute most of the pointers. However, we note that fault

tolerance can be easily obtained by running independent instances of ABSORPTION in parallel. In particular, we have shown that any node in a strongly-connected graph has equal probability to become the ultimate leader.

The open question is whether there exists a resource discovery algorithm achieving optimal asymptotic bounds on all three complexity measures.

Of related interest is the problem of transforming a weakly-connected graph to a strongly-connected graph. For this problem, open question also remains that whether there exists an $O(\log n)$ time algorithm with $O(n)$ messages or $O(n^2)$ pointers (Table 4.3).

Chapter 5

Bluetooth Scatternet Formation

5.1 Introduction

Bluetooth [14, 43, 17, 72] is an emerging low-cost and low-power short-range radio technology. Many useful applications can be supported by an ad hoc network over Bluetooth. For example, in a conference room, a special announcement can be broadcast to the Bluetooth-enabled mobile phones and hand-held computers through an ad hoc network. Bluetooth ad hoc networks can also be used for rapid deployment of electromagnetic identification readers [9].

The area of ad hoc networking has gathered significant research interests in recent years. Many studies have concentrated on the routing issues of ad hoc networks [78, 95]. These studies usually assume that any two in-range nodes can communicate with each other. Therefore, an ad hoc network can be modeled as a graph such that the in-range nodes are adjacent. For example, simulation-based studies [18, 21] of ad hoc routing protocols have been conducted with a link-layer model based on or similar to the IEEE 802.11b standard.

An ad hoc network based on Bluetooth, however, brings new challenges. There are specific Bluetooth constraints not present in other wireless networks. For example, a Bluetooth network is composed of *piconets*. Each piconet contains one master and up to seven slaves. Piconets can be connected into a larger *scatternet* (Figure 5.1) by sharing slaves. As shown by Miklos et al. [71] and Zurbes [100], the configuration of a scatternet has significant impact on the performance of the network. For instance, when a scatternet contains more piconets, the rate of packet collisions increases. Before we can make effective use of Bluetooth ad hoc networking, we must first devise an efficient protocol to form a scatternet from isolated Bluetooth devices.

In this chapter, we study the problem of scatternet formation in the situation where the devices are in-range of one another. The communication range is at least 10 meters according to the current Bluetooth specification. This means that our formation algorithm

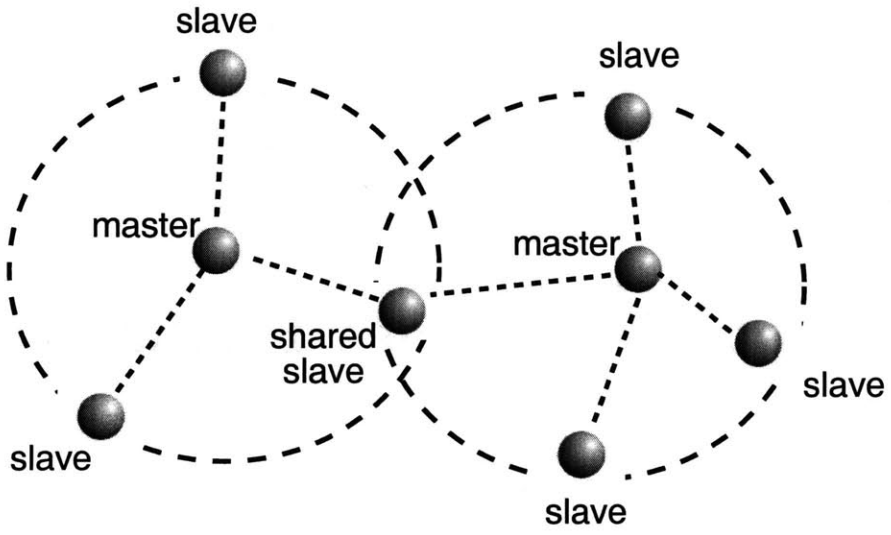

Figure 5.1: A Bluetooth scatternet.

should work when the maximum distance between any two devices is at most 10 meters. We will discuss in Section 5.8 how the algorithm should adapt if the assumption is not satisfied.

We adopt a two-layer approach to this problem. First, we investigate how these devices can be organized into scatternets. We design and evaluate the performance of a new scatternet formation protocol. Second, as a subroutine of the formation protocol, we study how the devices can discover each other efficiently.

This chapter is organized as follows: In Section 5.2, we discuss the related research on Bluetooth scatternets. To get a better understanding of how our results differ from prior work, readers may skip this section and come back to it after going through the results of this chapter. In Section 5.3, we introduce the problem of scatternet formation. Our new scatternet formation protocol is presented in Section 5.4. We present theoretical analyses and simulation results of our protocol in Section 5.5. We discuss device discovery with simulation results in Section 5.6. In Section 5.7 we estimate the overall performance of the protocol. We discuss several variations and extensions to our protocol in Section 5.8 and conclude in Section 5.9. Theoretical results in this chapter have appeared in a conference paper [64]. Simulation results in this chapter have appeared in a conference paper [61] and a thesis [70].

5.2 Related Work

Miklos et al. [71] apply heuristics to generate scatternets with some desirable properties. They evaluate these scatternets of different characteristics through simulations. Johansson et al. [49] perform link-layer simulations of piconets. Raman, Bhagwat, and Seshan [82] argue for cross-layer optimization in Bluetooth Scatternets.

Aggarwal et al. [3] introduce a scatternet formation algorithm. Their algorithm first partitions the network into independent piconets, and then elects a 'super-master' that knows about all the nodes. However, the resulting network is not a scatternet, because the piconets are not inter-connected. A separate phase of re-organization is required.

Salonidis et al. [87] discuss the issues of symmetric connection between a pair of Bluetooth devices. In their symmetric protocol, the devices switch states (INQUIRY and INQUIRY SCAN) with a random schedule. In contrast, in our work, the devices switch states periodically, but pick the states randomly.

Salonidis et al. [88] introduce a scatternet formation algorithm—Bluetooth Topology Construction Protocol (BTCP). BTCP has three phases: (I) a coordinator is elected with a complete knowledge of all devices, (II) this coordinator determines and tells other masters how a scatternet should be formed, and (III) the scatternet is formed according to the instructions. A formation scheme is presented in [88] for up to 36 devices. In contrast, our algorithm has only one phase. Since the topology is decided by a single device (the coordinator), BTCP has more flexibility in constructing the scatternet. However, if the coordinator fails, the formation protocol has to be restarted. BTCP's timeout value for the first phase would affect the probability that a scatternet is formed. Our protocol's timeout value for each round only affects the overall performance of the protocol—the scatternet will be formed with certainty. In addition, BTCP is not suitable for dynamic environments where devices can join and leave after the scatternet is formed.

The algorithms in [3, 88] depend on a single device to design the scatternet topology and notify other devices. Therefore these algorithms will have time complexity $\Omega(n/k)$, where n is the number of nodes, and k is the maximum number of slaves in a piconet. In comparison, our algorithm consists of a single phase and has $O(\log n)$ time complexity. However, as pointed out in [88], the coordinator election phase dominates the total time requirement. Thus, the advantage of our protocol's $O(\log n)$ time complexity might not be relevant in practice unless the number of devices is very large. Moreover, we note that at least the phase II of BTCP can be modified to run in $O(\log n)$ time, if the topological information is distributed along a tree. However, a tree-based distribution scheme will increase the complexity of the protocol.

Tan [94] gives a distributed Tree Scatternet Formation (TSF) protocol. The extensive simulation results indicate relatively short scatternet formation latency. However, TSF is not designed to minimize the number of piconets. The simulation results suggest that each master usually has fewer than 3 slaves. In comparison, our protocol guarantees that all but

one of masters have at least 6 slaves. Thus, our protocol leads to fewer number of piconets and fewer collisions among the piconets.

Bluetree [98] and Bluenet [96] are scatternet formation protocols for larger-scale Bluetooth networks, in which the devices can be out of range with one another. Simulation results of the routing properties of the scatternets were presented in [98, 96]. However, there were no simulation or theoretical analyses on the performance of the scatternet formation process.

5.3 Preliminaries

In this section we introduce some terminologies and performance measures for the scatternet formation problem.

Bluetooth devices share 79 channels of 1 MHz bandwidth in the 2.4 GHz band using frequency hopping. When two Bluetooth devices are connected, one of the devices acts as a *master* and the other device acts as a *slave*. Any Bluetooth device can perform the role of a master or a slave.

A Bluetooth device can discover other devices by the inquiry process. A master in INQUIRY state hops 3,200 times per second according to a 32-channel inquiry hopping sequence. At the same time, a slave in INQUIRY SCAN state changes its listening frequency every 1.28 seconds, along the same sequence.

If the inquiry process succeeds, the master learns the address (which is unique for each Bluetooth device) and the clock of the slave. In the page process, the master in PAGE state contacts the slave with a 32-channel page hopping sequence, which is a function of the slave's address and (estimated) clock. Similarly, the PAGE SCAN slave hops with the period of 1.28 seconds along the same sequence. After the master and the slave are connected, they communicate with a hopping sequence over all 79 channels at the rate of 1600 hops per second. This hopping sequence is determined by the master's clock and address.

A *piconet* consists of 1 master and 1 to k active slaves[1]. All packets are exchanged between a master and its slaves within a piconet. There is no direct master-master or slave-slave communication. A device can be a slave in several piconets but be a master in only one piconet. The *degree* of a device is the number of piconets to which the device belongs. A device is *unshared* if its degree is 0 or 1. Otherwise, it is *shared*. A *scatternet* is a set of piconets connected through shared devices.

The problem of scatternet formation: How does a collection of isolated devices form a scatternet? The devices are isolated in the beginning; each device is not aware of the other devices. Therefore, the scatternet formation protocol must be distributed. We assume that the devices are in the communication range of each other. Thus, potentially, any two devices can be connected directly.

[1] k is 7 in Bluetooth Specification 1.1 [15]

A scatternet formation protocol has two major performance measures:

- time complexity — amount of time to form a scatternet. A scatternet should be formed as fast as possible to minimize the delay experienced by the users.
- message complexity — number of messages sent between the devices. This is important because Bluetooth devices usually operate with limited power. By reducing the number of messages sent, power consumption is conserved.

Furthermore, it is also crucial to have scatternets of good quality. It is not very useful to have scatternets leading to poor communication performance. Thus, we should aim to form a scatternet that facilitates inter-piconet communications. It is not easy to quantify the quality of a scatternet, but we believe the following measures are good indicators.

- number of piconets — a measurement of a scatternet's efficiency. Since all piconets share the same set of 79 channels, there will be more collisions when there are more piconets. As shown in [100], the burst failure rate increases with the number of piconets.
- maximum degree of the devices — the maximum number of piconets that any device belongs to. Since the piconets communicate through shared slaves, if a slave belongs to many piconets, then this slave could become the bottleneck of inter-piconet communications. A shared slave has to be time multiplexed between the piconets that it belongs to. Therefore, a shared slave of high degree could become overloaded.
- network diameter — maximum number of hops between any pair of devices. This will provide us with an estimation of the maximum routing delay of the scatternet.

A good balance among the quality measures is desirable. Consider, for example, a star topology: a single "central" slave is shared by all piconets. In such a scatternet of n devices with every piconet containing k slaves, there are $\lceil (n-1)/k \rceil$ piconets. Although the number of piconets is minimized (see Remark 65), this scatternet probably would not perform very well in practice because the shared slave will be overwhelmed, unless the network is small.

Remark 65. *Let k be the maximum number of slaves in a piconet. A scatternet of n devices must contain at least $\lceil (n-1)/k \rceil$ piconets.*

Proof. We need to show that a scatternet of n devices has at least $\lceil (n-1)/k \rceil$ piconets. Let $p(n)$ be the minimum number of piconets for a scatternet of n devices. We will show that $p(n) \geq \lceil (n-1)/k \rceil$ by induction on $p(n)$.

First, if $p(n) = 1$, then $n \leq k+1$ by our assumption that each piconet can have at most $k+1$ devices. Therefore

$$p(n) = 1 = \lceil k/k \rceil \geq \lceil (n-1)/k \rceil .$$

Next, assume that if $p(n) \leq m$, then $p(n) \geq \lceil (n-1)/k \rceil$. Then we consider the case that $p(n') = m+1$. Given a scatternet of n' devices, we pick a master and remove its piconet so that the rest of the scatternet is still connected. We can at most remove k devices because this piconet was connected with the rest of the scatternet. Therefore, after this removal, we are left with m piconets, and at least $n' - k$ devices. By the inductive hypothesis, we have $p(n) \geq \lceil (n-1)/k \rceil$ if $p(n) \leq m$. Since $n \geq n' - k$, we have $p(n) \geq \lceil (n'-k-1)/k \rceil = \lceil (n'-1)/k \rceil - 1$. Thus, $p(n') \geq p(n) + 1 \geq \lceil (n'-1)/k \rceil$.

□

5.4 Scatternet Formation

In this section, we first present our scatternet formation protocol and then evaluate its performance and properties by analyses and simulations. The development of this algorithm was inspired by our research on resource discovery algorithms in general networks [63]. The main idea is to merge pairs of connected components until one component is left. Each component has a leader. In each round, a leader either tries to contact another component or waits to be contacted. The decision of each leader is random and independent. Our protocol in [63] forms a complete graph in $O(\log n)$ rounds. In this paper, we apply the same idea to connect Bluetooth devices in $O(\log n)$ rounds.

5.4.1 Algorithm

Initially, we are given a set of isolated but in-range devices. During the execution of the algorithm, the devices are partitioned into *components*. A component is a set of interconnected devices, and can be a single device, a piconet, or a scatternet. There is one *leader* in each component. For a single-device component, the only member is the leader. For a piconet, the master is the leader. For a scatternet, one of the masters is the leader. When a leader *retires*, it stops being a leader and will be inactive for the rest of the algorithm (unless it becomes a leader again). For any device v, let $\mathcal{S}(v)$ be the set of v's slaves. If v is not a master or has no slaves, then $\mathcal{S}(v) = \emptyset$. Let $k \geq 2$ be the maximum number of slaves allowed in a piconet. Thus $|\mathcal{S}(v)| \leq k$ for any v.

In Lemma 66, we will prove the following invariants for the algorithm:

- Each leader either has no slave, or has at least one unshared slave in its piconet.
- Each leader has fewer than k slaves in its piconet, i.e., $|\mathcal{S}(u)| < k$ for any leader u.

All leaders execute procedure MAIN in the beginning of each round. We assume a constant δ, such that procedure MAIN and the procedures called by it can be completed in δ seconds. A good choice of δ can be found by simulations (see Section 5.6) and by

prototyping. We assume that all leaders call procedure MAIN at time $t_0+i\delta$, for $i = 0, 1, \ldots$, where t_0 is the start time.

In the beginning, all devices are leaders. In procedure MAIN, a leader calls SEEK with probability $p \in (1/3, 2/3)$. Otherwise, the leader calls SCAN or asks an unshared slave to call SCAN. During each round, only one device in each component should call SEEK or SCAN.

```
MAIN(leader u)
1  x ← a random number in [0, 1)
2  if x < p   (1/3 < p < 2/3)
3     then SEEK(u)
4     else if S(u) = ∅
5             then SCAN(u)
6             else v ← an unshared slave of u
7                  SCAN(v)
```

When a leader executes SEEK, it tries to acquire a new slave (which is running SCAN). However, the leader may not always succeed, because, in any given round, the number of devices running SCAN can be smaller than the number of devices running SEEK. Therefore, if a leader is not able to contact a slave after certain time, it should give up and run MAIN again in the next round. Similarly, SCAN might also fail in any given round. During each round, a matching is found between the SEEK devices and SCAN devices. The number of connections established (size of the matching) is the smaller of the number of SEEK devices and the number of SCAN devices.

```
SEEK(u)
1  u performs INQUIRY
2  if a slave v is found
3     then u connects to slave v by PAGE
4          // S(u) ← S(u) ∪ {v}
5          CONNECTED(u, v)
```

```
SCAN(v)
1  v performs INQUIRY SCAN
2  if v is contacted by a master u
3     then v waits for u in PAGE SCAN
```

We note that SEEK and SCAN devices will go into PAGE and PAGE SCAN modes respectively after all inquiries are completed. The amount of time required is investigated in Section 5.6. In general, we make sure that each master is matched to only one slave, and

vice versa. When a leader u running SEEK connects to a slave v running SCAN, procedure CONNECTED(u, v) is called.

Procedure CONNECTED(u, v) merges the component of u and the component of v. There are several cases:

1. If v is an isolated leader, then v would become a slave of u unless the piconet of u has become full, in which case a new piconet with master v and unshared slave y is created, as shown in Figure 5.2. This is necessary because otherwise u's piconet would be full, violating the second invariant. To satisfy the first invariant, we also need to give the new master v an unshared slave y.

2. If v is not an isolated leader, then let w be the master of v.

   (a) If the piconet of u and the piconet of w can fit into a single piconet (with at most $k-1$ slaves because of the second invariant), then w and its slaves join the piconet of u, as shown in Figure 5.3.

   (b) Otherwise, we cannot merge the two piconets, and thus we should let w retire.

      i. If u was an isolated master, u would not have an unshared slave. Thus, we will let u become the slave of retiring master w and let v become an unshared slave of u (Figure 5.4).

      ii. If the merged piconet is just full, violating the second invariant, we will need to let v become a master and give it an unshared slave y (Figure 5.5).

      iii. Otherwise, we will just try to move as many nodes allowed by the invariants as possible from the piconet of u to the piconet of retiring master w, so as to reduce the overall number of piconets (Figure 5.6).

```
CONNECTED(leader u, slave v)
 1  if v is a leader
 2    then // v was an isolated leader
 3         if |S(u)| < k
 4           then v retires
 5           else u retires // Figure 5.2
 6                y ← an unshared slave of u
 7                MOVE({y}, u, v)
 8    else w ← the other master of v
 9         w retires
10         switch
11           case |S(u) ∪ S(w)| + 1 < k : // Figure 5.3
12                MERGE(u, v, w); return
13           case |S(u)| = 1 : // Figure 5.4
14                MOVE({u}, NIL, w)
15                v disconnects from w ; return
16           case |S(u) ∪ S(w)| + 1 = k : // Figure 5.5
17                u retires
18                y ← an unshared slave of u
19                MERGE(u, v, w)
20                MOVE({y}, u, v)
21                v becomes a leader ; return
22           case default : // Figure 5.6
23                MIGRATE(u, v, w); return
```

Communications between u and w in CONNECTED, MERGE, MIGRATE, and MOVE are via their common slave v.

Procedure MERGE(u, v, w) makes w and all its slaves become u's slaves.

```
MERGE(master u, slave v, master w)
1  v disconnects from w
2  MOVE(S(w) \ v, w, u)
3  MOVE({w}, NIL, u)
```

Procedure MIGRATE(u, v, w) moves slaves from $\mathcal{S}(u)$ to $\mathcal{S}(w)$ until $\mathcal{S}(w)$ is full or when only two slaves are left in $\mathcal{S}(u)$.

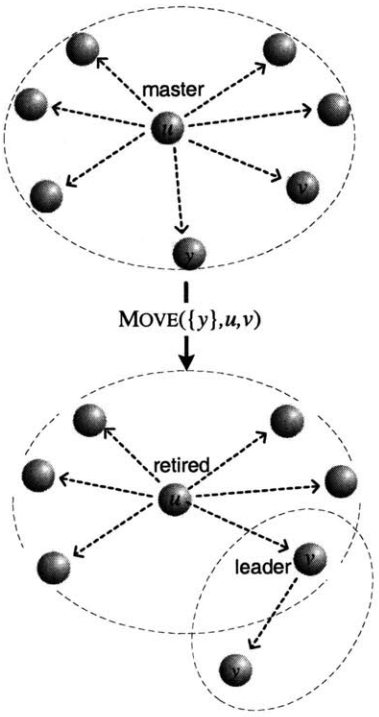

Figure 5.2: Lines 5–7 in procedure CONNECTED for $k = 7$.

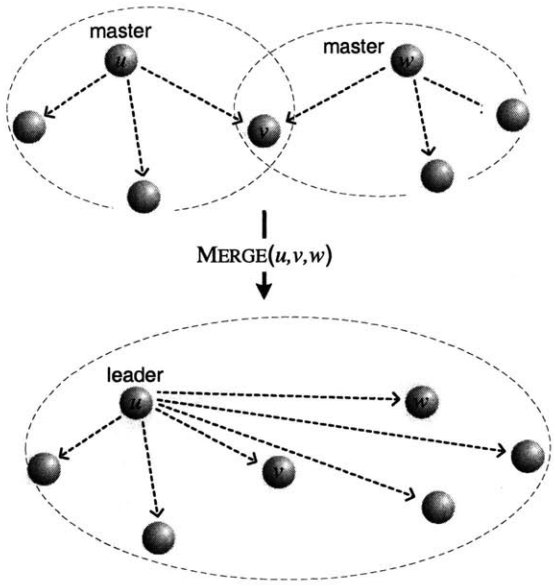

Figure 5.3: Lines 11–12 ($|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 < k$) in procedure CONNECTED for $k = 7$.

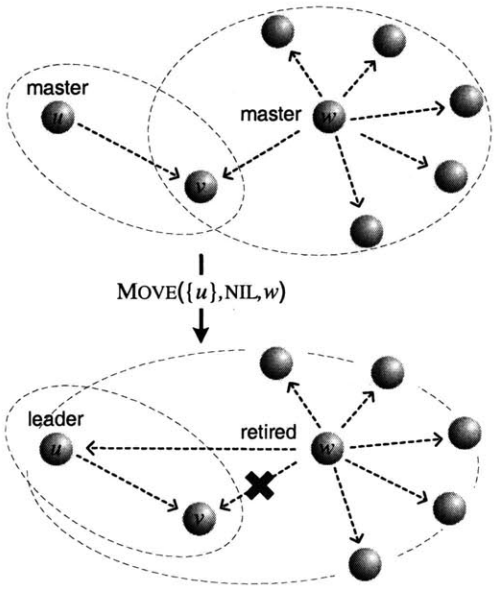

Figure 5.4: Lines 13–15 ($|\mathcal{S}(u)| = 1$) in procedure CONNECTED for $k = 7$.

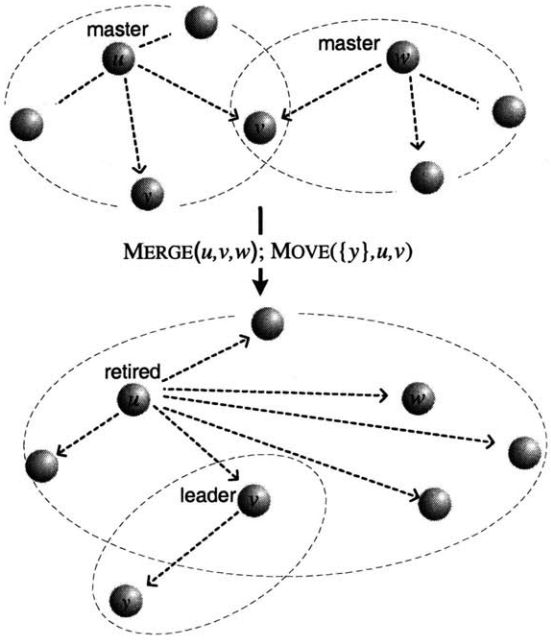

Figure 5.5: Lines 16–21 ($|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 = k$) in procedure CONNECTED for $k = 7$.

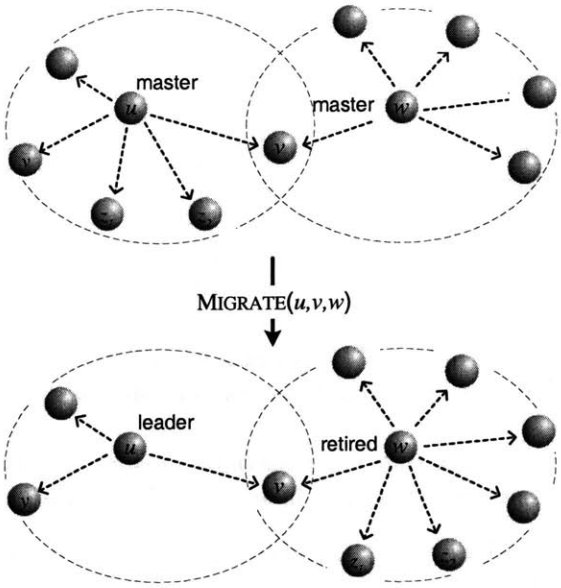

Figure 5.6: Lines 22–23 (default) in procedure CONNECTED for $k = 7$.

```
MIGRATE(master u, slave v, master w)
1  i ← min(k − |S(w)|, |S(u)| − 2)
2   // i is the number of slaves to migrate
3  if i > 0
4     then y ← an unshared slave of u
5          Z ← { i slaves in S(u) \ {y, v} }
6          MOVE(Z, u, w)
```

Procedure MOVE is a subroutine called by CONNECTED, MERGE, and MIGRATE. All devices in set Z disconnect from u and become slaves of w.

```
MOVE(set Z, master u, master w)
1  devices in Z disconnect from u
2  devices in Z wait for w in PAGE SCAN
3  w connects to devices in Z by PAGE
```

Lemma 66 proves the invariants of the algorithm.

Lemma 66. *During the execution of the algorithm, the following invariants hold:*

- *Each leader has either no slave, or has at least one unshared slave.*

- *Each leader has fewer than k slaves.*

Proof. We will prove the invariants by induction.

In the beginning, all devices are leaders without slaves. So our invariants are satisfied.

Assuming a state satisfying the claim, we only have to make sure that CONNECTED and its calls to MERGE, MIGRATE, and MOVE preserve the invariants, because no piconet is formed or modified in MAIN, SEEK, and SCAN.

Let $\mathcal{S}'(u)$ and $\mathcal{S}'(v)$ be the slaves of u and v after CONNECTED(u, v) is returned.

First, we consider the case that v is a leader (lines 1–7). If v is a leader, it means that v does not have any slave. If $|\mathcal{S}(u)| < k$ (lines 3–4), v would become an unshared slave of u. If u has exactly k slaves (lines 5–7), then one unshared slave y is moved from $\mathcal{S}(u)$ to $\mathcal{S}(v)$. Thus, $\mathcal{S}'(v)$ contains an unshared slave. In this case, u is retired so that it does not need an unshared slave.

Second, we consider the case that v is a slave of a leader w. Master w will no longer be a leader, so it does not have to satisfy the invariants. There are four cases:

($|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 < k$): All devices in $\mathcal{S}(w)$ become slaves of u. Device v was an unshared slave in $\mathcal{S}(w)$. After the merge, u is the only master of v, so v becomes an unshared slave of u. Also, we note that $|\mathcal{S}'(u)| = |\mathcal{S}(u) \cup \mathcal{S}(w)| + 1$ is smaller than k by assumption.

($|\mathcal{S}(u)| = 1$): Leader u will have v as its only slave. Slave v is unshared because it was w's unshared slave.

($|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 = k$): In this case, u retires so it does not need to satisfy the invariants. When slave v becomes a leader, it obtains an unshared slave y from u.

default: Slaves in $\mathcal{S}(u)$ are migrated to $\mathcal{S}'(w)$ until $|\mathcal{S}'(w)|$ is k or $\mathcal{S}'(u)$ contains only two slaves (one of them is v). Procedure MIGRATE will always reserve an unshared slave y for $\mathcal{S}'(u)$. By assumption, w had at most $k-1$ slaves before CONNECTED is called. Therefore, we can move at least one slave from $\mathcal{S}(u)$ to $\mathcal{S}'(w)$. Therefore, $|\mathcal{S}'(u)|$ is at most $k-1$, because at least one slave is removed after u has obtained slave v. □

This algorithm does not minimize the absolute number of messages passed between the devices. This is not crucial in practice, as Section 5.6 will show that most of the packets are sent during the inquiry processes. The current design is a compromise between simplicity of the algorithm and the constant factors of the message complexity of the algorithm.

The last leader will keep calling MAIN even after the scatternet is formed. It is because the leader cannot be certain that all devices are already connected unless it knows the total number of devices. In practice, we can let the leader stop after it has failed to find any device for certain number of rounds. The probability that n leaders fail to make any connections for l rounds is $(p^n + (1-p)^n)^l$, which is less than $(5/9)^l$ for $n \geq 2$ and $1/3 < p < 2/3$.

5.5 Performance and Properties

5.5.1 Theoretical Results

Scatternet Properties

We show that the scatternet formed possesses two useful properties: small degrees for shared devices and small number of piconets.

Lemma 67. *At most one piconet in the scatternet formed by the algorithm contains fewer than $k-1$ slaves.*

Proof. When a scatternet is formed, only one component is left. Therefore, except for one piconet, the masters of the other piconets have retired. We will show that a retired master has at least $k-1$ slaves. Therefore, when the scatternet is formed, all but one of the masters have at least $k-1$ slaves.

We only need to make sure that if a leader becomes a retired master in a round, it should have at least $k-1$ slaves, because a retired master will not lose any slave in subsequent rounds. There are four places in CONNECTED that a leader is retired.

line 4: v becomes a slave.

line 5: The test $|\mathcal{S}(u)| < k$ is false. Thus u had at least k slaves before line 7: MOVE($\{y\}, u, v$), which reduces the number of u's slaves by 1. Thus u would have $k-1$ slaves when retired.

line 9: We must show that for all the four cases in lines 11–23, w will have at least $k-1$ slaves when CONNECTED returns if w remains a master.

In the first and third cases, w loses all of its slaves in the procedure MERGE and becomes a slave itself.

In the second case (lines 13–15), we have $|\mathcal{S}(u)| = 1$ but $|\mathcal{S}(u) \cup \mathcal{S}(w)|+1 \geq k$ because the condition of the first case is not satisfied. Since u and w share one slave, we have $|\mathcal{S}(u)|+|\mathcal{S}(w)|-1+1 \geq k$. Thus, $\mathcal{S}(w) \geq k-1$ before MOVE is called. Master w loses slave v, but gains a new slave u, so w still has at least $k-1$ slaves when procedure CONNECTED returns.

In the last case (lines 22–23), we have $|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 \geq k+1$, and MIGRATE will move all devices in the piconet of u to the piconet of w until w has k slaves or u has only 2 slaves left. In the latter case, only one slave in $\mathcal{S}(u) \cup \mathcal{S}(w)$ will not become a slave of w. Thus w would have at least $k-1$ slaves after the MIGRATE operation.

line 17: All slaves of w and w itself become slaves of u in line 19. We note that u and w had $k-1$ slaves in total (line 16), thus u should have k slaves after MERGE (line 19).

MOVE (line 20) would remove one slave from u. Therefore, u still has $k-1$ slaves when CONNECTED returns. □

Lemma 68. *The algorithm forms a scatternet with $m-1$ devices of degree 2 and $n-m+1$ devices of degree 1, where n is the number of devices and m is the number of piconets.*

Proof. First, we will show that any device has maximum degree 2. We will verify that the shared slaves and shared masters have degrees at most 2.

Shared Slaves We observe that only unshared slaves may participate in SCAN. Thus, a shared slave will not be shared again through SCAN. A shared save might become unshared in a MERGE or become a degree-2 shared master in lines 16–21 of procedure CONNECTED.

Shared Masters A shared master can only be a slave of a retired master. Therefore, a shared master will never be shared again with another master through SCAN. In addition, a shared master v is always created from an unshared slave or an isolated master. This means that v had no slave before becoming a shared master. Therefore, a device can only become a shared master once.

We can now consider the topological graph of the scatternet, in which each piconet is a node and each degree-2 device is an edge. We can show that this topological graph is a tree. Initially, each component is a tree (a single node). During each CONNECTED call, at most one edge is created between the two merging components. And since each component only participates in one CONNECTED process during each round, the components remain trees throughout the protocol.

In a tree of m nodes, there are exactly $m-1$ edges. Therefore, there are $m-1$ degree-2 devices, and $n-(m-1)$ degree-1 devices. □

Theorem 69. *The scatternet formed by the algorithm contains at most $\lfloor (n-2)/(k-1) \rfloor + 1$ piconets.*

Proof. Consider a scatternet produced by the algorithm. Let n be the number of devices and m be the number of piconets. By Lemma 67, at most one piconet has size less than k. (A piconet has size less than k if and only if it has fewer than $k-1$ slaves.) Such piconet has size at least 2. By Lemma 68, $m-1$ devices have degree 2 and the rest of the devices have degree 1. Therefore, we can conclude that the scatternet contains at least

$$k(m-1) + 2 - (m-1) = (k-1)(m-1) + 2$$

devices. Thus, $n \geq (k-1)(m-1)+2$. Since m is an integer, $m \leq \lfloor (n-2)/(k-1) \rfloor + 1$. □

Comparing Theorem 69's upper bound $\lfloor (n-2)/(k-1) \rfloor + 1$ with Remark 65's lower bound $\lceil (n-1)/k \rceil$, we note that our bound is very close to be optimal. For example, when $n = 100$ and $k = 7$, our algorithm forms a scatternet containing at most 17 piconets, while the lower bound requires 15 piconets.

Asymptotic Complexities

We first derive the algorithm's time complexity and then its message complexity.

Lemma 70. *During a round with at least 2 leaders, the number of leaders is reduced by a constant fraction with a constant probability.*

Proof. Let $m \geq 2$ be the number of leaders. Let p be the probability that a leader chooses to run SEEK. The algorithm specifies that $1/3 < p < 2/3$. We will assume $p \leq 1/2$, because if $p > 1/2$, we can switch the roles of SEEK and SCAN and the proof follows similarly.

During each round, we have a matching between the SEEK devices and the SCAN devices. Let random variable X be the number of SEEK devices in a given round. Since X is distributed binomially with parameter p, we have $\mathrm{E}[X] = pm$ and $\mathrm{Var}[X] = mp(1-p)$.

Let α be a real number between 0 and 1. If $(1-\alpha)pm \leq X \leq (1+\alpha)pm$, then at least $(1-\alpha)pm$ connections are made between the SEEK devices and SCAN devices because: (1) there are at least $(1-\alpha)pm$ SEEK devices; and (2) there are at least

$$m - (1+\alpha)pm = (1-p-\alpha p)m \geq (1-\alpha)pm$$

SCAN devices since $(1-p) \geq p$ if $p \leq 1/2$.

Thus, the probability of having at least $(1-\alpha)pm$ connections (size of the matching between SEEK devices and SCAN devices) is

$$\begin{aligned}
&\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} \\
&= \Pr\{(1-\alpha)pm \leq X \leq (1+\alpha)pm\} \\
&= \Pr\{|X - pm| \leq \alpha pm\} \\
&= 1 - \Pr\{|X - pm| > \alpha pm\}.
\end{aligned}$$

The Chebyshev's inequality states that

$$\Pr\{|X - \mathrm{E}[X]| > t\} < t^{-2}\mathrm{Var}[X].$$

By setting $t = \alpha pm$, $\mathrm{E}[X] = pm$, and $\mathrm{Var}[X] = mp(1-p)$, we have

$$\Pr\{|X - pm| > \alpha pm\} < \frac{mp(1-p)}{(\alpha pm)^2} = \frac{1-p}{m\alpha^2 p}.$$

Since $m \geq 2$ and $p > 1/3$, we have $(1-p)/pm < 1$. Thus we can pick α so that $\alpha^2 > (1-p)/2p \geq (1-p)/mp$. Then $c = (1-p)/(2\alpha^2 p)$ is a constant smaller than 1. Therefore,

$$\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} > 1 - c.$$

Each connection reduces the number of leaders by 1. Therefore, with probability at least $1 - c$, the number of leaders is reduced by a fraction $(1-\alpha)p$. □

Theorem 71. *The algorithm forms a scatternet in $O(\log n)$ rounds with probability at least $1 - O(1/n)$,*

Proof. We note that a scatternet is formed when there is only one leader left. By Lemma 70, when there are at least two leaders, the number of leaders is reduced by a fraction with some probability q. The probability that the algorithm takes more than $O(\log n)$ rounds to reduce the number of leaders to 1 is at most $q^{O(\log n)} = O(1/n)$. □

Theorem 72. *The expected message complexity of the algorithm is $O(kn)$.*

Proof. We first consider the message complexity of each invocation of the procedures. We note that each of the procedures MAIN, SCAN, SEEK, CONNECTED, MERGE, MIGRATE sends $O(1)$ messages. Procedure MOVE moves at most k devices. Thus it sends $O(k)$ messages.

To analyze the message complexity of MAIN, SEEK, and SCAN, it is sufficient to find the expected number of times that MAIN is called, because each call to MAIN leads to a call to SEEK or a call to SCAN.

First, we can assume that when a leader w chooses SCAN such that if it or its slave is contacted by another leader u, then w will retire. This is true except that if u has k slaves, then u will retire instead. See lines 5-7 in procedure CONNECTED. However, for simplicity of the analysis, we can assume that w retires instead of u. In other words, we can assume that w and u swap their identities whenever we are in this case. This will not affect our result because we only care about the total number of messages sent by these leaders. The high-level algorithm does not rely on an identifier of the device. (Device address is used by low-level Bluetooth INQUIRY and PAGE. But these processes are independent between different rounds in the algorithm.)

During any round, each leader chooses SCAN with probability $1 - p$. Assume that a leader w has chosen SCAN. Leader w or w's unshared slave will definitely be contacted by another leader if the total number of SCAN devices is not more than the number of SEEK devices.

Let X_i be the random variable of the number of SCAN devices over i components. Thus, $\mathrm{E}[X_i] = (1-p)i$. Let $m \geq 2$ be the number of components.

We now assume that $p \geq 1/2$ because if otherwise, we can switch the roles of SCAN and SEEK. Assume w has chosen SCAN.

$$\begin{aligned}&\Pr\{w \text{ or } w\text{'s slave is contacted by a leader}\}\\&= \Pr\{X_{m-1} \leq m/2 - 1\}.\end{aligned}$$

By Markov inequality, we have

$$\begin{aligned}\Pr\{X_{m-1} > m/2 - 1\} &= \Pr\{X_{m-1} \geq m/2 - 1/2\}\\&\leq \mathrm{E}[X_{m-1}]/(m/2 - 1/2)\\&= 2(1-p).\end{aligned}$$

Thus,

$$\Pr\{X_{m-1} \leq m/2 - 1\} \geq 1 - 2(1-p) = 2p - 1.$$

Thus, each leader retires with probability at least $(1-p)(2p-1)$, which is positive except when $p = 1/2$.

We now consider the case where $p = 1/2$. In the proof of Lemma 70, we have

$$\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} > 1 - (1-p)/(\alpha^2 pm).$$

Let $p = 1/2$, $\alpha = 1/2$, and $m \geq 5$, then

$$\Pr\{\text{at least } m/4 \text{ connections}\} > 1/5.$$

Therefore, with probability at least $1/5$, at least $m/4$ connections are made. And when that happens, each device has a probability of at least $1/4$ to be the slave of a connection being made. This proves our argument for $m \geq 5$. The cases where $m = 2, 3, 4$ can be easily verified.

We have shown that any leader has a constant probability of retiring during each round. This means that each leader is active for $O(1)$ rounds. Thus MAIN is called $O(n)$ times in total, and the overall message complexity for procedures MAIN, SEEK, SCAN is $O(n)$.

Procedure CONNECTED is called exactly $n-1$ times. Thus, the message complexity of CONNECTED is $O(n)$. Each call to CONNECTED could result in at most 1 call to MERGE, at most 1 call to MIGRATE, and at most 3 calls to MOVE. Thus the overall message complexity of MERGE and MIGRATE is $O(n)$, and the overall message complexity of MOVE is $O(kn)$. □

Corollary 73. *If k is a constant, then the message complexity of the algorithm is $O(n)$.*

We note that $O(n)$ is the optimal asymptotic message complexity because each device needs to send at least one message.

5.5.2 Simulation Results

In this subsection, we investigate the properties and performance of our scatternet formation protocol.

We simulate our scatternet formation algorithm with `simjava` [46], a discrete event simulation package for Java. The probability p that each leader chooses to execute SEEK in each round is $1/2$ in our simulations. Following the Bluetooth specification, we set $k = 7$. We start with 2, 4, and 8 nodes, and then increase by increments of 8 nodes, up to 128 nodes. This allows us to present the results against the number of nodes in linear scale and in logarithmic scale. Each data point in the graphs of this section represents an average of 50 trials.

Scatternet Properties

First, we found that the maximum degree of the scatternet formed is 1 when there are fewer than 8 nodes and is 2 when there are at least 8 nodes. This means that the maximum degree is optimal except when there are 8 nodes, in which case a maximum degree of 1 is possible.

As we discussed in Section 5.3, it is important to minimize the number of piconets because piconets interfere with each other. Figure 5.7 shows that the number of piconets formed lies between the protocol's theoretical upper bound $\lfloor (n-2)/(k-1) \rfloor + 1$ and the universal lower bound $\lceil (n-1)/k \rceil$. The largest difference between our simulation result and the lower bound is 2.2 piconets.

The network diameter, which is the maximum shortest path length between any pair of devices, captures the maximum routing delay between any pair of nodes in the scatternet. Although we do not have a theoretical analysis of the network diameter, Figure 5.8 shows that the network diameter grows about logarithmically with the number of devices (x axis is in logarithmic scale).

Performance

First, it is crucial that the scatternet is formed as fast as possible, because this translates to the latency experienced by the users. In Figure 5.9, we can see that the number of rounds required to form the scatternet is around $1.2 \log_2 n + 2$. This validates the $O(\log n)$ time complexity theoretical result. In Section 5.6, we will investigate the amount of time required in each round.

Second, as most mobile Bluetooth devices are expected to run on batteries, it is important to minimize the number of messages sent in order to conserve battery power. We can put the messages into three categories:

Inquiries – Bluetooth INQUIRY and INQUIRY RESPONSE packets.

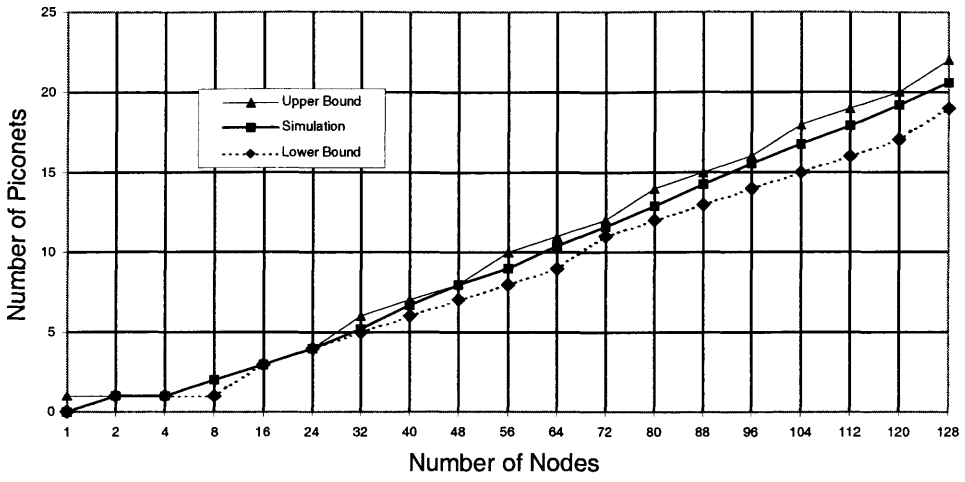

Figure 5.7: Number of piconets in the scatternet formed, compared to upper bound $\lfloor (n-2)/(k-1) \rfloor + 1$ and lower bound $\lceil (n-1)/k \rceil$, where $k = 7$.

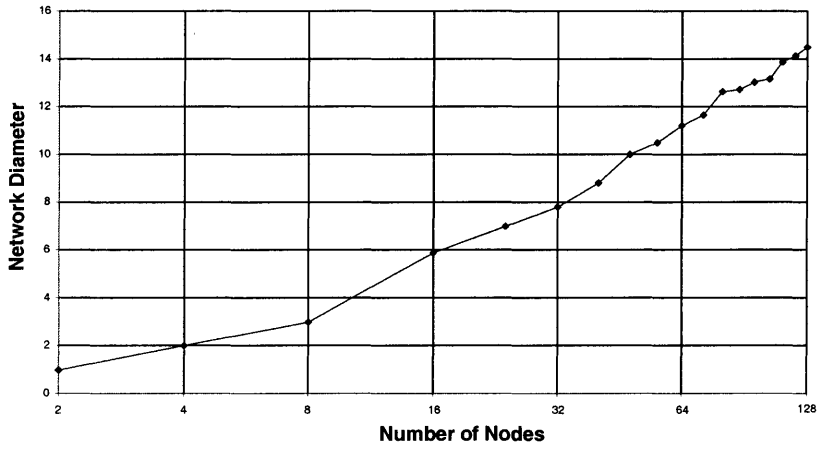

Figure 5.8: Network diameter of the scatternet formed.

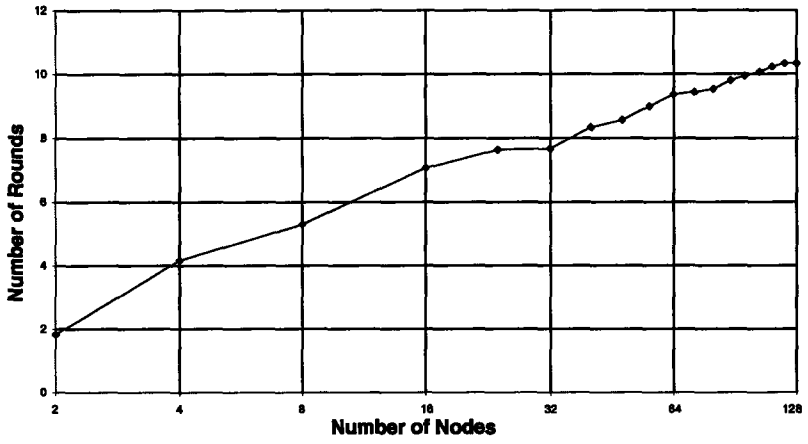

Figure 5.9: Number of rounds to form a scatternet.

Pages – Bluetooth PAGE and PAGE RESPONSE packets.

Algorithmic Messages – other messages used by our scatternet formation algorithm.

Figure 5.10 presents the total number of the Algorithmic Messages, Inquiries, and Pages. We can verify that the numbers of all three types of messages increase linearly with the number of devices. This agrees with the $O(n)$ message complexity theoretical result.

In Figure 5.11, we can see that the maximum number of messages sent by any device increases logarithmically with the number of nodes. This implies that the power requirement of the "unluckiest" device is $O(\log n)$. A likely candidate of such unlucky device is the last remaining leader in the protocol. Since the last leader is not retired, the number of messages sent by this leader is $\Theta(\log n)$.

In the next section, we will find out how long it takes to finish one round of inquiry and page. We will also see how many packets are sent during the inquiry and page processes.

5.6 Device Discovery

In this section, we investigate the performance of the device discovery protocol used during each round of the scatternet formation algorithm.

During scatternet formation, there are many devices trying to get connected at the same time, so the inquiry and page processes will interfere with each other. We call this the problem of device discovery when a set of in-range devices try to connect with each other. In the following, we discuss our approach and present the simulation results.

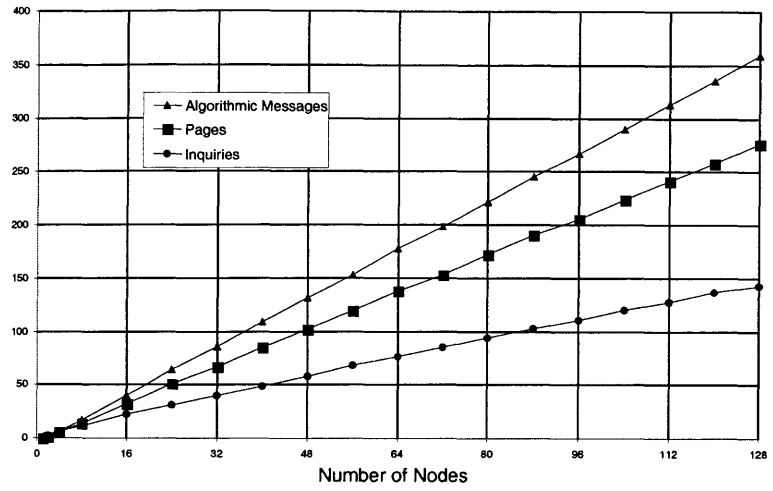

Figure 5.10: Total number of Algorithmic Messages, Pages, and Inquiries.

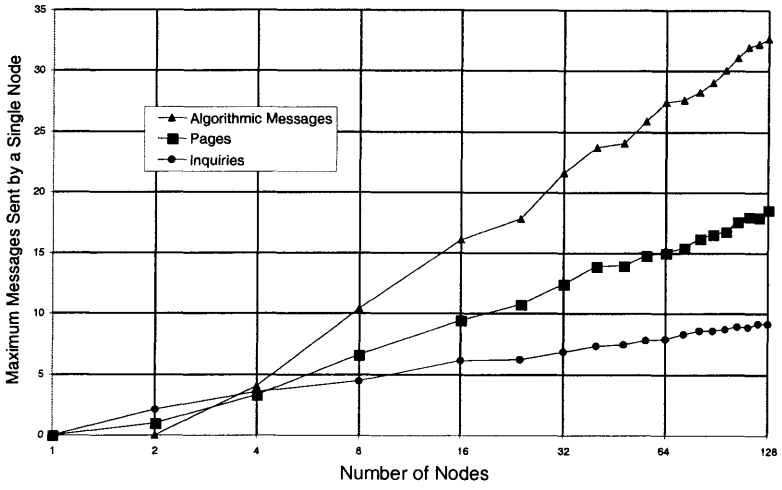

Figure 5.11: Maximum number of Algorithmic Messages, Pages, and Inquiries sent by any single node.

5.6.1 Protocol

We describe a simple randomized protocol for the problem of device discovery. This protocol is repeated during each round of the scatternet formation algorithm introduced in Section 5.4. We are given n devices that are not aware of each other. Our goal is to establish as many connections as possible. We are not concerned with exactly which of the devices are connected.

First, each device independently decides to be a SEEK node (with probability p) or a SCAN node (with probability $1-p$).

The protocol contains two phases - the inquiry phase and the page phase. In the inquiry phase, all the SCAN devices stay in the INQUIRY SCAN state. Each SEEK device will try to contact a SCAN device. However, a SEEK device may not always succeed in finding a slave because the number of SCAN devices can be smaller than the number of SEEK devices. Therefore, if a SEEK device is not able to contact a slave after certain amount of time, it will simply give up. Similarly, a SCAN device might also fail to be connected. In the page phase, the already paired devices are connected with PAGE and PAGE SCAN.

This protocol makes sure that each SEEK device is connected to at most one SCAN device and each SCAN device is connected to at most one SEEK device. In other words, we obtain a one-to-one matching between the SCAN devices and SEEK devices. The number of connections established is the smaller of the number of SEEK devices and the number of SCAN devices.

5.6.2 Simulation Results

We also used `simjava` [46] to simulate this protocol[2]. Each Bluetooth device is simulated by a thread. In each time slot, all devices first send messages, which include the frequency channel numbers, to a special object `Air`. Object `Air` detects the collisions in each of the 79 frequency channels, and only delivers the uncollided messages to those devices listening on the respective frequency channels. Inquiry and page frequency hopping sequences are implemented according to the Bluetooth specification. Since the overall time scale of the simulation is small, we did not simulate the clock drift. Each data point in the figures of this section is an average of 10 trials. In each trial, the devices are assigned addresses and clocks randomly.

Figure 5.12 shows the running time of the inquiry phase with three different master-to-slave ratios. For example, when there are 16 devices in total, a 50%–50% split leads to 8 masters and 8 slaves, and a 25%–75% split leads to 4 masters and 12 slaves. In the simulations of our algorithm in Section 5.5.2, we set p to $1/2$. Thus, we expect to see 50% masters and 50% slaves in each round. The actual outcomes at each round are

[2]IBM's BlueHoc simulator [13] is not used because we began implementing our simulator before BlueHoc was released in public.

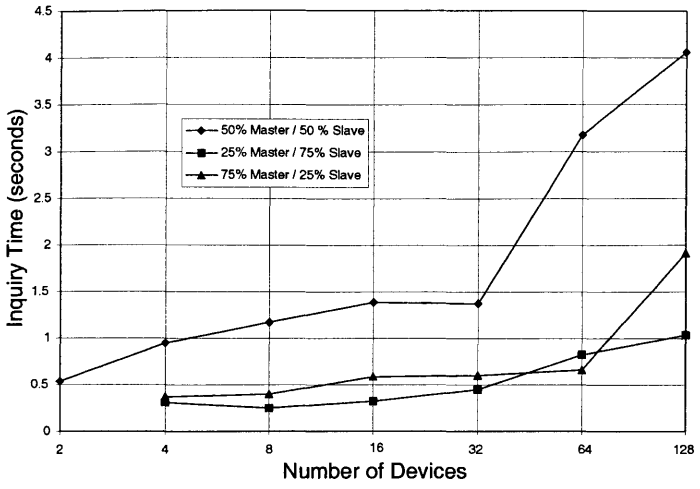

Figure 5.12: Running time of the inquiry phase with three master-to-slave ratios.

distributed according to the binomial distribution. For more than 8 devices, the 25%–75% split and 75%–25% split encompass at least 2 standard deviations around the expectation. We observe that the inquiry time of the 50%–50% split case increases sharply when there are around 64 devices. Since all SEEK devices follow the same inquiry hopping sequence (the phase depends on the device's clock), packet collision is a major problem when there are many devices. From the collision graph (Figure 5.13) on the 50%–50% split case, we can deduce that collisions start to hurt the performance severely when there are around 64 devices.

In Figure 5.14, we observe that, for up to 64 devices, the time consumed by the page phase is below 0.02 seconds, which is insignificant compared to the time required for the inquiry phase. This is because the SEEK devices already know the addresses and clocks of their target SCAN devices, thus they are able to contact the SCAN devices quickly. In addition, since they have different hopping sequences, the amount of collisions is lower in this case.

Figure 5.15 shows the total number of packets sent. Again the number of packets sent rises sharply around the 64-device case, due to collisions. However, we can see that the total number of packets is around $(10000/32)n$ for $n = 32$, 64, and 128. This means that the total number of packets sent increases roughly linearly with the number of devices.

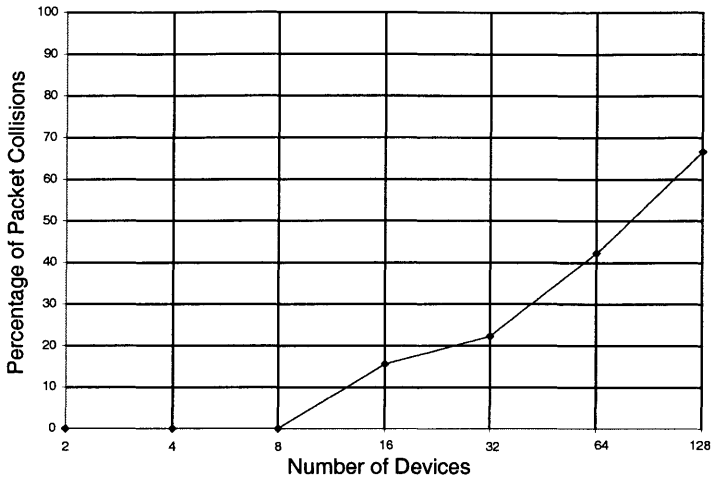

Figure 5.13: Percentage of packet collisions (over all packets sent) when there are 50% masters and 50% slaves.

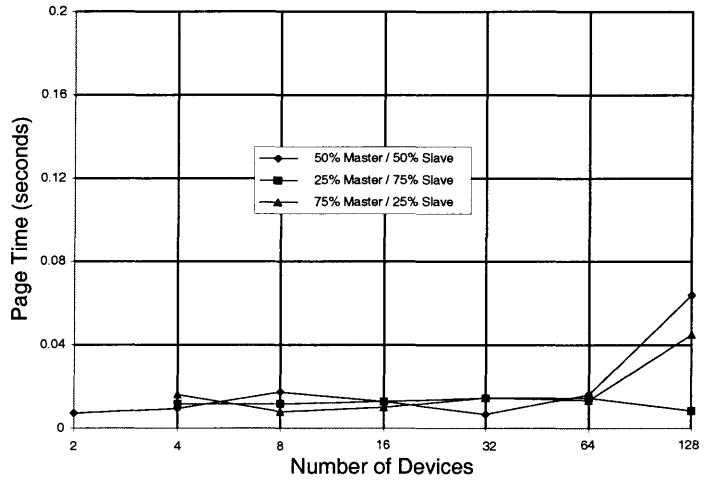

Figure 5.14: Running time of the page phase with three master-to-slave ratios.

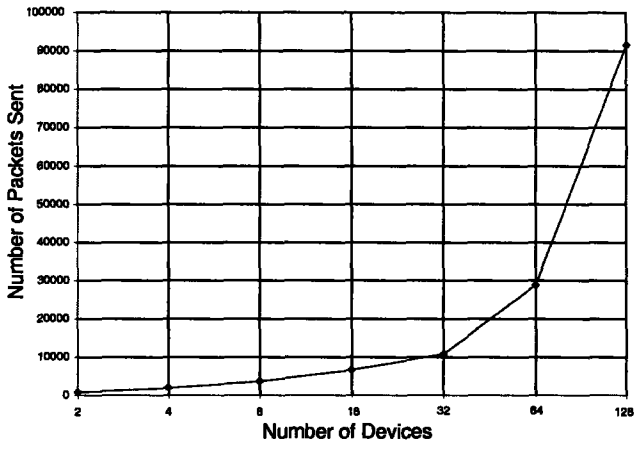

Figure 5.15: Total number of packets sent when there are 50% masters and 50% slaves.

5.7 Overall Performance

We now estimate the overall performance of our protocol, using the results of Sections 5.5.2 and 5.6.2.

In Section 5.5.2, we learned that the number of Inquiries, Pages, and Algorithmic Messages all increase linearly with the number of devices. And in Section 5.6.2, we found that the number of packets sent during a single round of the protocol increases linearly with the number of devices. Therefore, we can conclude that overall message complexity of the protocol is linear. This means that the average power consumed by a device remains constant when the number of devices increases. In Section 5.5.2, we also showed that the number of Inquiries, Pages, and Algorithmic Messages of any single device increases logarithmically. Thus, our protocol does not cause a very high load on any single device.

To estimate the total time taken by the protocol, we can multiply the number of rounds (Figure 5.9) by the time required for each round. The time taken in each round is the sum of the time required for the inquiry phase (Figure 5.12), the page phase (Figure 5.14), and the procedure CONNECTED.

We can estimate the time required for CONNECTED. During each round, each device will perform either PAGE or PAGE SCAN at most once as a result of procedure CONNECTED. Procedure CONNECTED does not cause any INQUIRY because the clocks and addresses of the devices are already known. Therefore, the time required for the PAGEs caused by CONNECTED should be more than the time required in the 50%–50% case of the page phase (Figure 5.14). In addition, $O(k)$ messages need to be exchanged among leader u, slave v, and leader w in procedure CONNECTED. The amount of information to be passed is small,

and thus the time required to pass these messages is insignificant compared to the time required for the INQUIRYs and PAGEs.

For example, according to our simulation results, for up to 32 devices, we expect that $1.39 + 0.02 \times 2 = 1.43$ seconds are required for each round. The protocol takes on the average 7.7 rounds to form the scatternet. Thus, the total time requirement is about $7.7 \times 1.43 < 11.1$ seconds. Similarly, the estimated total time required for 16 devices and 64 devices are at most 10.2 seconds and 30.3 seconds respectively. Section 5.8 discusses how the overall performance can be improved with an asynchronous version of our protocol.

5.8 Variations and Extensions

In the following, we discuss several limitations of our protocol and suggest techniques for overcoming them.

5.8.1 Inquiry Collisions

When there are many devices, packet collisions among INQUIRY devices can adversely affect the performance. In particular, if two INQUIRY devices happen to have their clocks in phase so that their inquiry sequences are synchronized, then their inquiry packets will collide repeatedly. This effect was observed in our simulations in those cases with large numbers of devices. It is conceivable that this problem can be alleviated if the INQUIRY devices back off randomly during a heavy-collision situation. We note that this back-off by the INQUIRY device is not related to the random back-off by an INQUIRY SCAN device after receiving an inquiry packet, as specified in Bluetooth 1.1.

5.8.2 Asynchronous Protocol

The overall time requirement estimated in Section 5.7 is longer than the phase I of BTCP [88]. To improve the performance, we should consider an asynchronous version of our protocol. We believe that the overall time requirement can be reduced because of the following observations:

- The worst-case time required per round happens when there is a perfect split between the masters and slaves. However, if this happens frequently, the total number of rounds required is small. For example, if there is a perfect split every round, the protocol will only need $\log_2 32 = 5$ rounds to form a scatternet of 32 nodes.

- The number of active leaders decreases rapidly. Thus, the device discovery processes in the later rounds can be completed faster.

We can consider an asynchronous version of our protocol with the following change: Once CONNECTED returns, the remaining leader can proceed to MAIN immediately. The synchronized nature of the current protocol is useful for the theoretical analyses of the performance. In practice, it is not necessary for all devices to execute CONNECTED at the same time. The overall performance of the asynchronous version should be better than the synchronous version. Moreover, the analyses on the degrees of devices (Lemma 68) and the number of piconets (Theorem 69) remain valid in the asynchronous version.

We note that it is not necessary for the devices to start at the same time in practice. Even in the synchronous version, a device can join the scatternet in a later round (see Section 5.8.4 for a discussion of dynamic environments).

5.8.3 Out of Range Devices

In some scenarios, some of the Bluetooth devices might be out of range of one another. Given arbitrary device connectivity, it is not possible to maintain the performance and scatternet properties guarantees. Despite such limitations, we can augment the protocol to try to form a scatternet whenever possible. Procedures SEEK and SCAN will not need to be modified because two devices will be connected only if they are in-range. We note that, other than SEEK, the only place that master-slave connections are established (by PAGE and PAGE SCAN) is in procedure MOVE. Therefore, procedure MOVE might fail. Let us consider the places in CONNECTED where MOVE is called:

lines 5–7 (Figure 5.2) If y cannot be connected to v, then we can try to use other unshared slaves of u. If all unshared slaves of u are not able to connect to v, then v should become a retired master and have u as its only slave.

lines 11–12 (Figure 5.3) The MERGE call might fail. In this case, we can let w retire with its smaller piconet.

lines 13–15 (Figure 5.4) If u cannot be connected to w, then we can let u be the slave of v. This will be similar to the original outcome except that v will be the new leader, instead of u.

lines 16–21 (Figure 5.5) If MERGE fails, we will just let w retire.

lines 22–23 (Figure 5.6) The MIGRATE procedure should move as many devices to the retiring master w as allowed by the underlying connectivity.

The above modifications, except the one on lines 5–7, only affect the total number of piconets of the scatternet formed, but not the maximum degree of any device in the scatternet formed.

Each execution of modified lines 5–7 might increase the degree of u by one. Without a distribution assumption of device locations, we cannot bound the probability of such event.

However, we can provide some reasons that such event is unlikely. Given that v was still an isolated device before the connection, we can show that it is unlikely that u has more than one shared slaves. If u has at least two shared slaves, then the component led by u has at least $k + (k-1) + (k-1)$ devices, because each retired piconet has at least $k-1$ slaves. This implies that at least $\lceil \log_2(3k-2) \rceil$ rounds have passed before v is able to make a connection. If $p = 1/2$, v has a probability of at least $1/2$ to make a connection in each round. Thus, if $k = 7$, then the probability that v is not connected for $\lceil \log_2 19 \rceil = 5$ rounds and then connect to u as a slave is at most $(1/32)(1/2) = 1/64$. When u has no more than 1 shared slave, it is unlikely that the $k-1$ or $k-2$ unshared slaves are all out of range with v.

Depending on the underlying connectivity of the devices, the piconets are likely to have smaller sizes, implying a larger number of piconets in the scatternet formed. Unless the situation discussed in the previous paragraph happens, the maximum degree of any device in the scatternet will still be two.

5.8.4 Joins, Leaves, and Faults

Our protocol can be easily extended to work with dynamic environments (with devices joining and leaving the scatternet) and device failures. Our current protocol already handles the events of devices joining—the new devices can simply start as leaders and thus discover or be discovered by other devices. Additional work is required to deal with the case of devices leaving or failing. We give an outline in the following:

- If a master fails (or leaves the network), then a new master can be elected from the slaves. If the failed master was shared, then the new master should become a leader and merge with the rest of the scatternet by the protocol.

- If a shared slave fails, its older master (the master who connected to this slave first) should become a leader again and then it will be connected to the rest of the scatternet by the protocol.

- Nothing needs to be done when an unshared slave fails, unless it is the only unshared slave of an active leader.

- In general, if we end up with a leader u with no unshared slave, then this leader has to disconnect from its shared slaves. Other masters of those shared slaves should now become leaders again. This will allow the protocol to proceed as usual. Fortunately, this expensive reorganization should occur rarely.

5.9 Concluding Remarks

In this chapter, we introduced a new Bluetooth scatternet formation protocol. We presented both theoretical and simulation results to show that our protocol has $O(\log n)$ time complexity and $O(n)$ message complexity.

We have shown that the algorithm produces scatternet with desirable properties: small number of piconets for minimizing inter-piconet interference, and small degrees for the devices for avoiding network bottlenecks. In addition, according to the simulations, the diameter of the scatternet, which corresponds to the maximum routing distance between nodes, is about $O(\log n)$. At last, we also demonstrated that no single device is particularly exhausted by the protocol.

Chapter 6

Conclusions

This thesis presented three new dynamic topology construction algorithms: a distributed construction of expander graphs, a resource discovery algorithm, and a Bluetooth scatternet formation algorithm.

Traditional distributed algorithms often assume a complete graph topology, a rigid topology such as ring or hypercube, or an arbitrary static topology. In contrast, the algorithms considered in this thesis work on evolving topologies. These algorithms not only adapt to a dynamic topology where nodes join and leave, but also actively establish and remove links between the nodes, to achieve certain global graph properties.

Much of the distributed computing research had focused on fault-tolerant group services. Algorithms such as leader election and consensus assume a set of processes (those non-malicious) working towards a common goal. However, since such service requires participation from all cooperating processes, scalability is limited. On the other hand, algorithms for peer-to-peer networks must consider scalability as the primary requirement. Furthermore, in order to establish a large-scale network in practice, the individual nodes have to be self-motivated. Recent examples are file sharing and distribution systems such as Overnet [76] and BitTorrent [12].

We believe that this new class of scalable algorithms that manipulate topologies will have impacts on practical large-scale distributed systems.

Many algorithms discussed in this thesis can be applied to other problems. For example, we expect the size estimation algorithm we proposed in Chapter 2 would be useful in other distributed systems. One future direction is to utilize the estimates in a load-balancing scheme for layered $\mathbb{H}$-graphs. It is also possible to improve the estimates on layered $\mathbb{H}$-graphs with cooperation across the layers.

In the following, we outline two systems that can be implemented with the $\mathbb{H}$-graphs introduced in this thesis.

We believe that networked virtual environments will be the next momentous application

on the Internet. The Internet has already captured a significant portion of the population. Very soon, our interactions on-line will become as natural as traditional connections through real-life presence. In addition, as advances in virtual reality will bring users immersive experiences, people will spend more time working, learning, and playing in the virtual worlds.

We need a protocol that can scale the size of our on-line societies to those of the off-line societies, and also scale with the bandwidth requirement of future virtual reality technologies. Instead of simply employing more powerful servers, I believe the ultimate solution will depend on direct client-to-client communications. In other words, we should model after the distributed physical world, in which we usually perceive each other directly through light and sound. Our objective is to implement client interactions with an active distributed algorithm — to see as little as possible (to conserve energy), yet still see those need to be seen (to be aware of one's proximity).

This problem can be considered as a distributed version of the dynamic nearest neighbor problem — each point is a process and has to maintain its own set of nearest neighbors. We can set up layers of networks so that each node is connected to all of its nearby neighbors, as well as random samples of those that are further away. The expander construction algorithm developed in this thesis serves as a building block of this novel protocol.

Another applied research direction of $\mathbb{H}$-graph is dynamic addressing. The demand for dynamic addressing arises from several trends:

1. each user will switch between different personal devices frequently;
2. mobile devices will have intermittent connections and dynamic IP addresses;
3. direct connection is preferred for applications such as instant messaging and gaming.

We can study the issues of implementing dynamic addressing with a distributed construction based on layered $\mathbb{H}$-graphs.

Our approaches to these problems carry a common thread — new graph-theoretic algorithms that work under an uncertain, changing topology and make use of local information about each node in the graph and its neighbors to achieve certain global graph properties.

Bibliography

[1] Ittai Abraham and Danny Dolev. Asynchronous resource discovery. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 143–150. ACM Press, 2003.

[2] *Communications of the ACM, special issue on Group Communications Systems*, volume 39. ACM, April 1996.

[3] Alok Aggarwal, Manika Kapoor, Lakshmi Ramachandran, and Abhinanda Sarkar. Clustering algorithms for wireless ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 54–63, Boston, MA, August 2000.

[4] David J. Aldous. On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing. *Probability in the Engineering and Informational Sciences*, 1:33–46, 1987.

[5] David J. Aldous and M. Brown. Inequalities for rare events in time-reversible Markov chains I. In M. Shaked and Y.L. Tong, editors, *Stochastic Inequalities*, volume 22 of *Lecture Notes*, pages 1–16. Institute of Mathematical Statistics, 1992.

[6] David J. Aldous and James A. Fill. Reversible Markov chains and random walks on graphs. Monograph in preparation.

[7] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[8] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–512, March 1992.

[9] Auto-ID Center. `http://autoidcenter.org/`.

[10] Baruch Awerbuch and Yossi Shiloach. New connectivity and msf algorithms for ultracomputer and pram. In *International Conference on Parallel Processing*, pages 157–179, August 1983.

[11] Michael A. Bauer and Tong Wang. Strategies for distributed search. In Jagan P. Agrawal, Vijay Kumar, and Virgil Wallentine, editors, *Proceedings of the Conference on Computer Science*, pages 251–260, New York, NY, USA, March 1992. ACM Press.

[12] Bittorrent. `http://bitconjurer.org/BitTorrent/`.

[13] BlueHoc: Bluetooth performance evaluation tool. `http://oss.software.ibm.com/developerworks/opensource/bluehoc/`.

[14] The Bluetooth Special Interest Group. `http://www.bluetooth.com/`.

[15] Specification of the Bluetooth system, version 1.1, February 2001.

[16] Béla Bollobás. *Random Graphs.* Cambridge University Press, second edition, September 2001.

[17] Jennifer Bray and Charles F. Sturman. *Bluetooth: Connect Without Cables.* Prentice Hall, 2001.

[18] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Mobile Computing and Networking*, pages 85–97, 1998.

[19] Fan R. K. Chung. *Spectral Graph Theory.* Number 92 in Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[20] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[21] Samir R. Das, Robert Castañeda, and Jiangtao Yan. Simulation-based performance evaluation of routing protocols for mobile ad hoc networks. *Mobile Networks and Applications*, 5:179–189, 2000.

[22] Eleni Drinea, Alan Frieze, and Michael Mitzenmacher. Balls and bins models with feedback. In *Proceedings of SODA 2002*, 2002.

[23] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960.

[24] Patrick Eugster, Sidath Handurukande, Rachid Guerraoui, Anne-Marie Kermarrec, and Petr Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of The International Conference on Dependable Systems and Networks*, July 2001.

[25] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, May 1996.

[26] Andras Farago, Imrich Chlamtac, and Stefano Basagni. Virtual path network topology optimization using random graphs. In *Proceedings of the 1999 IEEE Computer and Communications Societies Conference on Computer Communications*, 1999.

[27] Uriel Feige. A fast randomized LOGSPACE algorithm for graph connectivity. *Theoretical Computer Science*, 169(2):147–160, December 1996.

[28] James Allen Fill. Eigenvalue bounds on convergence to stationarity for nonreversible markov chains. *The Annals of Applied Probability*, 1(1):62–87, 1991.

[29] David A. Freedman. Bernard Friedman's urn. *Annals of Mathematical Statistics*, 36:956–970, 1965.

[30] Freenet. `http://freenet.sourceforge.net/`.

[31] Bernard Friedman. A simple urn model. *Communications on Pure and Applied Mathematics*, pages 59–70, 1949.

[32] Joel Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11:331–362, 1991.

[33] Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. accepted to the Memoirs of the American Mathematical Society, 2004.

[34] Alan Frieze, Mark Jerrum, Michael Molloy, Robert Robinson, and Nicholas Wormald. Generating and counting Hamilton cycles in random regular graphs. *Journal of Algorithms*, 21(1):176–198, July 1996.

[35] Alan Frieze and Colin McDiarmid. Algorithmic theory of random graphs. *Random Structures & Algorithms*, 10:5–42, 1997.

[36] Alan Frieze and Lei Zhao. Optimal construction of edge-disjoint paths in random regular graphs. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 346–355, January 17–19 1999.

[37] Hans Garmo. The asymptotic distribution of long cycles in random regular graphs. *Random Structures and Algorithms*, 15, 1999.

[38] David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, August 1998.

[39] Gnutella. `http://gnutella.wego.com/`.

[40] Francisco M. Gomes and Danny C. Sorensen. ARPACK++: A C++ implementation of ARPACK eigenvalue package. Technical Report TR97729, CRPC, Rice University, Houston, TX, 1997.

[41] John Greiner. A comparison of parallel algorithms for connected components. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 16–25. ACM Press, 1994.

[42] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 658–667. IEEE Computer Society, October 14–17 2001.

[43] Jaap Haartsen. Bluetooth - the universal radio interface for ad hoc, wireless connectivity. *Ericsson Review*, (3):110–117, 1998.

[44] Shay Halperin and Uri Zwick. An optimal randomized logarithmic time connectivity algorithm for the erew pram (extended abstract). In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM Press, 1994.

[45] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, May 1999.

[46] Fred Howell and Ross McNab. simjava: A discrete event simulation library for Java. In *Proceedings of International Conference on Web-Based Modeling and Simulation.* Society for Computer Simulation International, January 1998.

[47] Joseph Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, 1992.

[48] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 482–520. PWS Publishing, 1996.

[49] P. Johansson, N. Johansson, U. Korner, J. Elg, and G. Svennarp. Short range radio based ad-hoc networking: performance and properties. In *Proceedings of the IEEE International Conference on Communications 1999*, volume 3, pages 1414–1420, 1999.

[50] M. F. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, pages 323–336, Berkeley, CA, February 2003.

[51] Nabil Kahale. Large deviation bounds for markov chains. Technical Report 94-39, DIMACS, 1994.

[52] Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, September 1995.

[53] Nabil Kahale. Isoperimetric inequalities and eigenvalues. *SIAM Journal on Discrete Mathematics*, 10(1):30–40, 1997.

[54] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 654–663, May 1997.

[55] David R. Karger, Noam Nisan, and Michal Parnas. Fast connected components algorithms for the EREW PRAM. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 373–381. ACM Press, 1992.

[56] Gyula O.H. Katona and Akos Seress. Greedy construction of nearly regular graphs. *European Journal of Combinatorics*, 14, 1993.

[57] A.-M. Kermarrec, L. Massoulié, and A.J. Ganesh. Reliable probabilistic communication in large-scale information dissemination systems. Technical Report 2000-105, Microsoft Research Cambridge, October 2000.

[58] Michael Krivelevich, Benny Sudakov, Van H. Vu, and Nicholas C. Wormald. Random regular graphs of high degree. *Random Structures and Algorithms*, 18, 2001.

[59] Shay Kutten and David Peleg. Asynchronous resource discovery in peer to peer networks. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, page 224. IEEE Computer Society, 2002.

[60] Shay Kutten, David Peleg, and Uzi Vishkin. Deterministic resource discovery in distributed networks. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 77–83. ACM Press, 2001.

[61] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. Performance of a new Bluetooth scatternet formation protocol. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing 2001*, Long Beach, California, USA, October 2001.

[62] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. A New Bluetooth Scatternet Formation Protocol. *ACM Mobile Networks and Applications Journal*, 8(5), October 2003.

[63] Ching Law and Kai-Yeung Siu. An $O(\log n)$ randomized resource discovery algorithm. In *Brief Announcements of the 14th International Symposium on Distributed Computing, Technical Report, Technical University of Madrid*, number FIM/110.1/DLSIIS/2000, pages 5–8, October 2000.

[64] Ching Law and Kai-Yeung Siu. A Bluetooth scatternet formation algorithm. In *Proceedings of the IEEE Symposium on Ad Hoc Wireless Networks 2001*, San Antonio, Texas, USA, November 2001.

[65] Ching Law and Kai-Yeung Siu. Distributed Construction of Random Expander Networks. In *Proceedings of The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, California, USA, April 2003.

[66] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 395–406. ACM Press, 2003.

[67] László Lovász. Random walks on graphs: a survey. *Combinatorics, Paul Erdős is Eighty (vol. 2)*, pages 353–398, 1996.

[68] László Lovász and Peter Winkler. Exact mixing in an unknown Markov chain. *Electronic Journal of Combinatorics*, 2, 1995. Paper #R15.

[69] Brendan D. McKay and Nicholas C. Wormald. Uniform generation of random graphs of moderate degree. *Journal of Algorithms*, 11:52–67, 1990.

[70] Amar K. Mehta. Ad-hoc network formation using Bluetooth scatternets. Master's thesis, Massachusetts Institute of Technology, June 2001.

[71] Gy. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson. Performance aspects of Bluetooth scatternet formation. In *Proceedings of The First Annual Workshop on Mobile Ad Hoc Networking and Computing*, 2000.

[72] Brent A. Miller and Chatschik Bisdikian. *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications.* Prentice Hall, 2000.

[73] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[74] Napster. `http://www.napster.com/`.

[75] Sotiris E. Nikoletseas, Krishna V. Palem, Paul G. Spirakis, and Moti Yung. Connectivity properties in random regular graphs with edge faults. *International Journal of Foundations of Computer Science*, 11, 2000.

[76] Overnet. `http://www.overnet.com/`.

[77] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, October 2001.

[78] Charles E. Perkins. *Ad Hoc Networking.* Addison-Wesley, 2000.

[79] C. A. Philips. Parallel graph contraction. In *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pages 148–157. ACM Press, 1989.

[80] James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9:223–252, 1996.

[81] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *J. Algorithms*, 27(2):170–217, May 1998.

[82] Bhaskaran Raman, Pravin Bhagwat, and Srinivasan Seshan. Arguments for cross-layer optimizations in Bluetooth scatternets. In *Proceedings of Symposium on Applications and the Internet, 2001*, pages 176–184, 2001.

[83] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of SIGCOMM 2001*, pages 161–172, August 2001.

[84] John H. Reif. Optimal parallel algorithms for integer sorting and graph connectivity. Technical Report TR-08-85, March 1985.

[85] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[86] Jared Saia, Amos Fiat, Steve Gribble, Anna R. Karlin, and Stefan Saroiu. Dynamically fault-tolerant content addressable networks. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[87] Theodoros Salonidis, Pravin Bhagwat, and Leandros Tassiulas. Proximity awareness and fast connection establishment in Bluetooth. In *First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, pages 141–142, 2000.

[88] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire. Distributed topology construction of Bluetooth personal area networks. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, 2001.

[89] Yossi Shiloach and Uzi Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:55–67, 1982.

[90] Aravind Srinivasan, K. G. Ramakrishnan, Krishnan Kumaran, Murali Aravamudan, and Shamim Naqvi. Optimal design of signaling networks for Internet telephony. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications*, pages 688–716, March 2000.

[91] Angelika Steger and Nicholas C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8:377–396, 1999.

[92] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001*, pages 149–160, August 2001.

[93] Tomasz Luczak Svante Janson and Andrzej Rucinski. *Random Graphs.* Wiley, 2000.

[94] Godfrey Tan. Self-organizing Bluetooth scatternets. Master's thesis, Massachusetts Institute of Technology, January 2002.

[95] C. K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems.* Prentice Hall, 2001.

[96] Zhifang Wang, Robert J. Thomas, and Zygmunt Haas. Bluenet – a new scatternet formation scheme. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, January 2002.

[97] N. C. Wormald. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory (B)*, 31:156–167, 1981.

[98] Gergely V. Záruba, Stefano Basagni, and Imrich Chlamtac. Bluetrees—scatternet formation to enable Bluetooth-based ad hoc networks. In *Proceedings of IEEE International Conference on Communications*, pages 273–277, June 2001.

[99] Ben Y. Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, University of California, Berkeley, 2001.

[100] Stefan Zurbes. Considerations on link and system throughput of Bluetooth networks. In *Proceedings of the 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 2, pages 1315–1319, 2000.