Peer-to-Peer Networks based on Random Graphs

Dissertation

by

Peter Mahlmann

Faculty for Electrical Engineering, Computer Science and Mathematics
Department of Computer Science and Heinz Nixdorf Institute
University of Paderborn, Germany

February 2010

Reviewers:

- Prof. Dr. Friedhelm Meyer auf der Heide, University of Paderborn, Germany
- Prof. Dr. Christian Schindelhauer, University of Freiburg, Germany

To Lena and Pia Marie

Acknowledgements

This work would not have been possible without the understanding and encouragement of my family and my parents as well as the support of all the colleagues with whom I worked together. I have to thank them all and I am especially indebted to the following people.

I am deeply grateful to my advisor, Prof. Friedhelm Meyer auf der Heide for his continuous encouragement during all the stages of this thesis and for giving me the opportunity to choose the direction of my research. In particular I would like to thank him for the opportunity to participate in the EU integrated project DELIS and especially for involving me into the management and administration of this project. The latter seemed to be a time intensive and thankless task at first glance, yet turned out to be a great time and an invaluable experience in the end.

I also want to sincerely thank my co-advisor, Prof. Christian Schindelhauer for the cooperation in research, his patience in numerous discussions, and knowing to appreciate a critical point of view.

Last but not least I want to thank Thomas Janson for his excellent work as student assistant and many fruitful discussions about 3nuts.

Contents

Notation

Frequently Used Variables and Notations

d	degree of a graph
n	number of nodes in a graph or peers in a network
P	a path in a graph or network
$E(S,T)$	set of edges between $S \subset V$ and $T \subset V$, i.e. $\{\{u,v\} \in E : u \in S, v \in T\}$
δS	set of undirected edges with exactly one endpoint in S
$\delta^+ S$	set of directed edges starting in S and pointing to nodes not in S
$\delta^- S$	set of directed edges pointing to nodes in S and not starting in S
$\#_E(e)$	multiplicity of edge e in the set of edges E of a multi-graph
$N(v)$	set of nodes neighboring v in an undirected graph
$N^+(v)$	successor nodes of v in a digraph, i.e. $\{u \in V : (v,u) \in E\}$
$N^-(v)$	predecessor nodes of v in a digraph, i.e. $\{u \in V : (u,v) \in E\}$
$N^+(v,i)$	successor node of v in an edge-labeled digraph due to the i-th labeled edge
$A(G)$	adjacency matrix of graph G
λ_i	i-th Eigenvalue of an adjacency matrix
$h(G)$	edge expansion of graph G
$\log n$	the binary logarithm $\log_2(n)$
a.a.s.	asymptotically almost surely, a probability $p \geq 1 - o(1)$
w.h.p.	with high probability, a probability $p \geq 1 - n^{-c}$

Asymptotic Growth of Functions [109]

$\mathcal{O}(n)$	$f(n) = \mathcal{O}(g(n)) :\Leftrightarrow \limsup_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} < \infty$
$\Omega(n)$	$f(n) = \Omega(g(n)) :\Leftrightarrow \liminf_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} > 0$
$\Theta(n)$	$f(n) = \Theta(g(n)) :\Leftrightarrow f(n) = \mathcal{O}(g(n)) \wedge f(n) = \Omega(g(n))$
$o(n)$	$f(n) = o(g(n)) :\Leftrightarrow \lim_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} = 0$
$\omega(n)$	$f(n) = \omega(g(n)) :\Leftrightarrow \lim_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} = \infty$

1

Introduction

Starting with Napster [100] and the Gnutella network [44] a new type of network architectures has emerged and continues pervading the Internet: Peer-to-peer networks. In a recent study by Ipoque[1] 1.3 petabytes of Internet traffic have been monitored and analyzed [105], revealing that peer-to-peer generated most traffic in all regions monitored (e.g. the fraction of peer-to-peer traffic in Germany was 52.79%). "Peer" literally means *equal* and describes the circumstance that the nodes (i.e. peers) of a peer-to-peer network can not be classified as client or server. On the contrary, the nodes (i.e. peers) of a peer-to-peer network have symmetrical functionality and each peer acts as server, client, and router at the same time. The literature provides several descriptions of what exactly characterizes "peer-to-peer" [106, 104, 85, 111, 112, 6]. In [6] peer-to-peer networks are defined as

> "[...] distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority."

This can be considered as a broad definition of peer-to-peer networks because it is not assumed that peers are exactly equal, i.e. according to the definition more powerful peers may take over additional tasks or provide additional resources to the network. In this thesis however, we are interested in networks which are *strictly* peer-to-peer, i.e. all peers of the network are exactly equal and each peer provides the same functionality.

The first application area of peer-to-peer networks was file sharing [74]. A file sharing network allows users to publish own files and make them available for download to other users. In file sharing networks such as Gnutella [44] the peer-to-peer structure is used only for the purpose of publication and searching files. The actual file transfer between users is performed using a direct connection. While file sharing surely still is the most prominent application area of peer-to-peer networks, it often is in conflict with

[1] See http://www.ipoque.com.

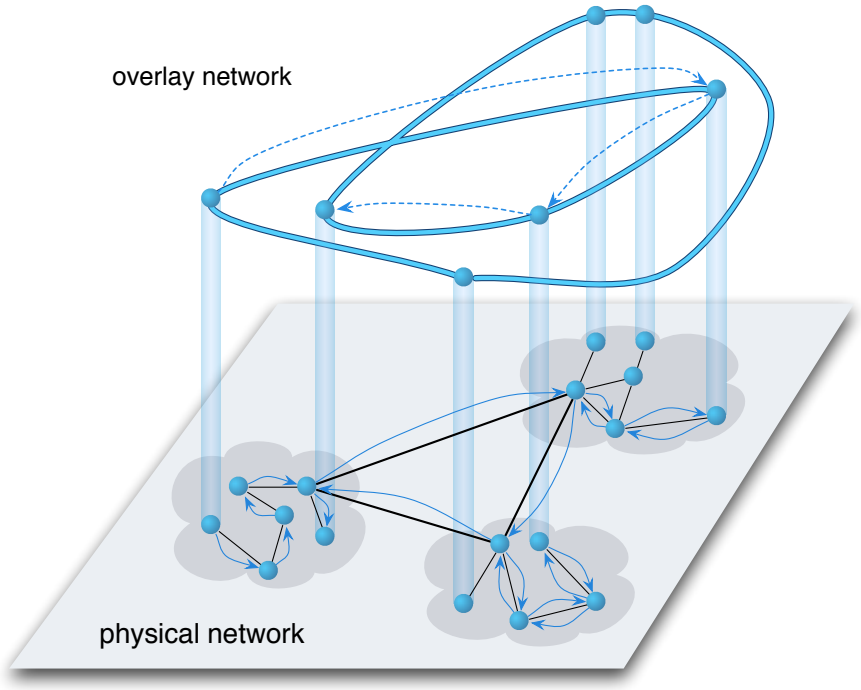


Figure 1.1: *An overlay network with underlying physical network. A single hop in the overlay (dashed lines) typically involves multiple hops in the physical network.*

copyright laws. However, there is also an increasing number of companies [25] developing legal peer-to-peer applications such as the Skype [10] VoIP (Voice over Internet Protocol) software allowing to make phone calls via Internet or the ZATTOO network [21] for live streaming of radio and television programs.[2]

Peer-to-peer networks belong to the class of so called overlay networks. An overlay network is a network which is build on top of a physical network, e.g. the Internet, and uses the physical network for realizing the communication among nodes (peers) that are connected in the overlay network, see Figure 1.1. Thus, links between peers are not physical but logical and therefore peers are able to join or leave the overlay at any time. Indeed, measurement studies of real world peer-to-peer networks reveal that these underly high churn rates [12, 46, 101], i.e. peers frequently join and leave the network without prior notice. Therefore, failure resilience plays an important role in peer-to-peer networks and robustness is a key to realize the actual potential of peer-to-peer in applications other than file sharing [97, 42]. So it is important to fully exploit the potential of the symmetrical functionality among peers since this property bears the potential of excellent failure resilience, i.e. there is no single point of failure and the impact of individual failures may be less than in conventional client server architectures.

[2] See `http://www.skype.com` respectively `http://www.zattoo.com`.

1.1 Unstructured vs. Structured Networks

Peer-to-peer networks may be divided into *unstructured* and *structured* networks. Unstructured networks such as Gnutella [44] are characterized by the lack of constraints on resource distribution and topology. A peer may select arbitrary other peers as neighbors (yet, these may reject connections) resulting in a random graph like topology. To publish resources a peer either maintains its own local index or places references on randomly chosen peers. For resource discovery each local index has to be queried separately so that complex queries — such as range queries or keyword queries with regular expressions — can be implemented easily. However, flooding or random walks have to be used for resource discovery. In consequence unstructured networks only have loose guarantees regarding resource discovery and resources may not be found although they exist in the network.

In structured peer-to-peer networks the placement of resources and the evolution of the topology is strictly controlled and therefore causes extra overhead. Controlling placement of resources obviously has the not inconsiderable advantage that peers searching for resources can determine the location of these resources in the network and so resources are guaranteed to be found as long as they exist. Often hash functions are used for resource placement, whereby many structured peer-to-peer networks are limited to exact match queries. The topology is controlled to allow efficient routing, i.e. the number of hops needed typically is logarithmic in the number of peers, and achieve low congestion. Often the topology is designed in such a way that routing architectures known from static networks — such as the DeBruijn, Butterfly, and Hypercube network — are emulated. Here, it seems that some static network architectures are better suited than others for this purpose, e.g. the Butterfly network seems to be harder to implement [76] than the DeBruijn network [80] in the dynamic setting of a peer-to-peer network. Generally, structured peer-to-peer networks are considered to be harder to maintain under churn and to be less robust than unstructured networks due to the extra overhead [22, 36, 103].

In the bottom line both unstructured and structured peer-to-peer networks have their individual advantages and disadvantages. Unstructured networks stand out with simplicity, robustness and allow complex queries, but can not guarantee to find existing resources and lack efficient query algorithms. On the other hand structured networks provide efficient query algorithms and guarantee to find existing resources but are less robust, harder to maintain under churn, and limited to exact match queries or range queries at the best.

1.2 Our Contribution

As pointed out above robustness is of major importance when designing peer-to-peer networks. It is reasonable to choose a simple network structure that is easy to maintain in case of strong dynamics and therefore keeps the network fully functional under churn or at least allows to recover quickly from worst case scenarios. These criteri-

ons are met by random networks. In this thesis we present methods to construct and maintain random networks using local operations which are applied to the network regularly. We propose to use these operations to improve the robustness of unstructured peer-to-peer networks or use them as building block for a structured peer-to-peer network.

1.2.1 Contributions Concerning Unstructured Networks

In Part I of this thesis we introduce and analyze several local distributed algorithms — we refer to these as local graph transformations — to generate and maintain truly random graphs respectively expander graphs[3]. We also show how these graph transformations can be implemented in a peer-to-peer network and propose to use them as simple and effective maintenance operation. Their practical relevance is underlined by experimental evaluation. Note that the local graph transformations presented here constitute an important improvement since previously known algorithms to generate random regular graphs can not be implemented in a peer-to-peer networking concept [58, 110] and algorithms proposed to maintain random graphs as network topology so far either make use of a central server [86], can not recover from degenerate network states [64], or do not guarantee connectivity [26].

In Chapter 3 our focus is on *undirected* random networks and we introduce a family of local graph transformations for the domain of connected regular undirected graphs, the so called k-Flipper operations. Given a path of $k+2$ edges a k-Flipper operation interchanges the end vertices of the path. We show that for all $k \geq 1$ every d-regular connected graph can be reached by a series of k-Flipper operations and use a randomized version, called Random k-Flipper, in order to create random regular connected undirected graphs. For the Random 1-Flipper we show that in the limit a series of these operations converges against an uniform probability distribution over all connected labeled d-regular graphs, implying that the Random 1-Flipper transforms any given graph into an expander graph asymptotically almost surely. Furthermore, we prove that any connected d-regular graph with n nodes and $d \in \Omega(\log n)$ will be transformed into an expander graph by a series of $\mathcal{O}(dn)$ Random k-Flipper operations when choosing $k \in \Theta\left(d^2 n^2 \log(1/\epsilon)\right)$ with high probability, i.e. $1 - n^{-\Theta(1)}$. The Flipper operations arose strong interest in the research community. At the time of publication [72], no theoretical bounds for the convergence rate of the 1-Flipper operation were known and several researchers tried to find upper bounds. A first bound was given by Feder et al. [35], who showed polynomial convergence. Recently, their result has been improved by Cooper et al. [28]. Despite all the efforts, a tight bound for the convergence rate still seems to be out of reach with current proof techniques. So, we conclude the chapter with experimental results indicating a convergence rate of $\mathcal{O}(dn \log n)$ operations for the 1-Flipper.

[3] An expander graph is a graph in which for every subset S of nodes has a relatively large number of edges with exactly one endpoint S. A formal definition of expander graphs is given by Definition 3.2 on Page 21.

In Chapter 4 we focus on *directed* random networks and introduce the so called Pointer-Push&Pull graph transformation. The aim of Pointer-Push&Pull is to improve the Flipper operations from a practical point of view and minimize the number of messages needed per graph transformation. The Pointer-Push&Pull operation can be used in parallel without central coordination and each operation involves only two peers which have to exchange two messages, each carrying the information of one edge only. We prove that a series of random Pointer-Push&Pull operations eventually leads to a uniform probability distribution over all weakly connected out-regular multi-digraphs[4]. Depending on the probabilities used in the operation this uniform probability distribution either refers to the set of all weakly connected out-regular multi-digraphs or to the set of all weakly connected out-regular edge-labeled multi-digraphs.

1.2.2 Contributions Concerning Structured Networks

In Part II we introduce a structured peer-to-peer architecture called 3nuts. One of the aims of 3nuts is to combine concepts of unstructured and structured peer-to-peer networks to overcome their individual shortcomings. This is achieved by cleverly combining self maintaining random networks for robustness, a search tree to allow range queries, and distributed hash tables (DHT) for load balancing. All network operations in 3nuts are local and distributed, i.e. 3nuts makes extensive use of Pointer-Push&Pull[5] maintained random networks and their excellent communication properties to maintain the network structure, spread information for load balancing, and measure round trip times (RTT) to adapt the overlay to the underlying physical network and thus allow routing with small latency.

Another important aim of 3nuts is to forgo the use of heuristics wherever possible since most other peer-to-peer networks providing similar functionality heavily rely on heuristics [2, 56, 66], this is especially the case for load-balancing mechanisms. So, in Chapter 7 efficiency of load balancing, fast data access, and robustness are proven by rigorous analysis. To the best of our knowledge 3nuts is the first structured peer-to-peer network efficiently supporting range queries, providing load balancing, and adapting the overlay to the physical network at the same time.

1.3 Bibliographical Notes

Many of the results presented in this thesis have been presented and published in a preliminary version in conference proceedings. The Flipper operations and their analysis have been presented in [72]. The corresponding experimental results for the Flipper operations (Section 3.4) appeared in [75]. The Pointer-Push&Pull operation and its

[4] In out-regular multi-digraphs each node has the same number of outgoing edges and multi-edges or self-loops may occur.

[5] Note that Flipper operations could be used as well, yet these would slightly increase the overall network traffic.

analysis has been presented in [73] and the 3nuts peer-to-peer network appeared as technical report [71] and is currently under submission.

Part I

Unstructured Networks

2

Introduction to Random Graphs and Unstructured Networks

In the first part of this theses, we concentrate on two important features necessary for successful peer-to-peer networks: Robust connectivity and randomness. As an example consider the Gnutella network [44]. In its first implementation Gnutella used an insertion procedure aiming at constructing a robust random network structure with small diameter. In Gnutella new peers join the network by connecting to former neighbors. If this fails, e.g. if the Gnutella network is used for the first time, the peer tries out a list of peers which often happen to participate in the network. The history of Napster and other cases have shown that specific hosts may cease to exist. If all former members do so, then a peer is not able to connect to the network. Since the list of former members is extended after each session this risk becomes smaller and smaller. Studies reveal that the network structure of Gnutella is not truly random, but a so-called Pareto (or power law) distributed graph [59, 96]. Compared to a random network, the degree distribution (i.e. the density function of neighbors) is skewed and also the diameter of the network is larger than in random networks with the same average out-degree. The original query algorithm of Gnutella — a broadcast of limited depth (usually referred to as *flooding*) — has severe scalability issues [98]. In [70] the authors evaluate several query strategies for Gnutella on different types of random graphs and the note that among the various network topologies considered, uniform random graphs yield the best performance. So, Gnutella would have been a better network if the peers would have been connected by a truly random network.

The idea of using random networks as topology for peer-to-peer networks also appears in the peer-to-peer network design suite JXTA of Sun Microsystems [90, 118, 84]. JXTA aims at connecting all peers by a random network to provide robustness, i.e. prevent peers or groups of peers from being disconnected. So, JXTA forms a design tool for peer-to-peer network applications like distributed search for web-sites [18]. The robustness and reliability of such random backbones is affirmed in [63], [82], [45], and [11]. The usage of random networks can be further motivated by results of graph theory, which show that random graphs provide high connectivity and expansion even for very small degrees [121]. Informally, the expansion property means that all partitions

of the set of nodes of a graph have a number of edges on the cut which is proportional to the smaller set of the partitions. Such expander graphs have a lot of desirable properties, e.g. logarithmic diameter, large conductance, and short mixing times of random walks [43].

Here, we concentrate on the generation of d-regular connected graphs (Chapter 3) respectively d-out-regular connected digraphs (Chapter 4). There is a number of reasons for this choice. These graphs describe the situation in peer-to-peer networks where the uniform degree induces some fairness to the network since each peer stores the same amount of network information. Furthermore, messages passed over a long random walk will pass all peers with equal probability. For an excellent survey on random regular graphs and their properties we refer to [121].

Little is known so far, whether the processes used in practice to maintain random graphs satisfy even minimal standards. We desire the following properties for random transformations of graphs:

Soundness No transformation maps to graphs which are not in the domain space, e.g. for d-regular undirected connected graphs, this means that each operation preserves degree d at every node and there is not even the slightest (small probability) chance to disconnect parts from the graph.

Generality The random transformation process does not converge to a specific graph. All graphs are *reachable* and in the limit all graphs occur with non-zero probability. This requirement can be tightened to *uniform generality* where in the limit all graphs occur with the same probability.

Feasibility The graph transformation can be described by a simple (distributed) routine changing only a small number of edges of the graph. Its implementation in a distributed network should be straightforward.

Convergence rate Only a small number of transformations is necessary to achieve an approximation of the ultimate distribution of all graphs.

Surprisingly, the process used in JXTA which is widely used, has never been analyzed with respect to these features. Identifying such transformations meeting all these requirements gives an approximate solution to the problem of generating all graphs of a specific domain respectively sampling random graphs — a well studied problem. Algorithms for sampling regular graphs uniformly at random have been presented by Bollobás [15], and McKay and Wormald [77]. For the problem of generating random regular graphs with nearly uniform probability distribution Jerrum and Sinclair [58] give a generator for d-regular graphs on n vertices which approximates the uniform probability distribution by a factor of $1 + \epsilon$ and runs polynomial in n and $\log 1/\epsilon$. Their result was improved by Steger and Wormald [110]. It is well known that almost every d-regular graph is connected and it is easy to use these algorithms to approximate also d-regular connected graphs. However, these methods cannot be applied in a networking concept. Furthermore, the algorithms do not guarantee connectivity of the graph.

Methods to generate random graphs that can be applied in a networking concept also have been proposed. We will briefly discuss some of these in the following. Pandurangan, Rhagavan, and Upfal presented an approach to build unstructured peer-to-peer networks in [86, 87]. In their protocol randomness is introduced to the network via the join operation only. New peers join the network by connecting to a random set of peers which is obtained via central cache server. There are explicit rules for nodes when to enter or leave the cache so that the degree of peers can be controlled. Furthermore, the protocol ensures that the resulting network is connected and has logarithmic diameter with high probability. The crucial weakness of the protocol however is the central cache server which represents a single point of failure and grossly contradicts the peer-to-peer paradigm.

Law and Siu [64] presented a distributed algorithm for constructing random overlay networks that are composed of d Hamilton cycles. A newly arriving peer joins each of the d Hamilton cycles at random positions respectively. The protocol is completely decentralized and the constructed topologies are expanders with diameter $\mathcal{O}(\log_d n)$ with high probability. However, as the authors note the probability space produced by their protocol may deviate far from the uniformly distributed space in a dynamic setting, what may lead to graphs with small or no expansion and their protocol is not able to recover from bad graphs.

A solution that is closely related to the algorithm that we will present in Chapter 3 has been proposed by Cooper, Dyer and Greenhill [26, 27].[1] They use a simple graph transformation called *switch* to maintain the network structure. A switch operation choses two nonadjacent distinct edges $\{u, v\}$, $\{u', v'\}$ uniformly at random (u.a.r.) and chooses a perfect matching M of $\{u, v, u', v'\}$ uniformly at random. If the edges of M do not already exist in the network, $\{u, v\}$, $\{u', v'\}$ are deleted and replaced by the edges of M. The switch operation is applied repeatedly to the network and thus "repairs" the topology in case of node failures or if unfavorable join operations have been used. The authors also present a detailed analysis of the Markov chain described by the switch operation and show that repeatedly applying switch operations to a d-regular graph generates all graphs of this domain with the same probability. Furthermore, they give an upper bound of $d^{15}n^8\left(dn\log(dn) + \log(\epsilon^{-1})\right)$ switch operations to reach an $1+\epsilon$ approximation of the uniform probability distribution. The switch operation is feasible if the graph is given as a data structure on a single machine. However, when the graph constitutes an interconnection network of computers, this operation is no more feasible. As long as all nodes are connected one can choose random edges by performing a random walk with an appropriate length, i.e. the mixing time of the graph. But the switch operation may — and in the long run definitely will — disconnect parts from the network. Then, without extra network connections the network cannot be rejoined anymore. Our point is that feasibility in terms of distributed algorithms implies maintaining connectivity at all stakes.

So, several algorithm to build respectively maintain the topology of unstructured

[1] Note that the results of Cooper, Dyer and Greenhill [26, 27] and ours [72] have been developed independently and at the same time.

peer-to-peer networks, i.e. random graphs, have been proposed. However, none of them fulfills the four properties we pointed out as desirable above. Chapter 3 and 4 will introduce maintenance operations that overcome the shortcomings of the switch operation as we will see.

2.1 Notations

In the following we introduce some notations which are used throughout this theses. By $\log n = \log_2 n$ we denote the dual logarithm function. The term "with high probability" (w.h.p.) describes a probability $p \geq 1 - n^{-c}$ and "asymptotically almost surely" (a.a.s.) is a probability $p \geq 1 - o(1)$. Furthermore, we use the following definitions for several classes of directed graphs, which we will refer to as digraphs from now on.

Definition 2.1 (Simple Digraph) *A simple digraph $G = (V, E)$ is defined by a node set $V = \{1, \ldots, n\}$ and a set of directed edges $E = \{(u, v) : u, v \in V, u \neq v\}$.*

A digraph is *strongly connected* if for all pairs of nodes there exists a directed path in E and *weakly connected* if there exists a path neglecting the direction of the edges for each pair of nodes. For $v \in V$ we define $N^+(v) = \{w : (v, w) \in E\}$ and $N^-(v) = \{u : (u, v) \in E\}$ to refer to the successor and predecessor nodes of v in G. A digraph is called d-regular if $\forall v \in V : |N^-(v)| = |N^+(v)| = d$. Furthermore, we say a digraph is d-out-regular if $\forall v \in V : |N^+(v)| = d$.

The usage of a multiset for the set of edges in digraphs will allow us to define a more general model of digraphs. Therefore, we give a formal definition of multisets.

Definition 2.2 (Multiset) *A set E and a function $\#_E : E \to \mathbb{N}_0$, specifying the multiplicity of its elements, define a multiset. For $e \in E$ we write $e \in_k E$ if $\#_E(e) = k$. This implies $e \in_0 E$ for all $e \notin E$. The cardinality of a multiset E is defined as $|E| = \sum_{e \in E} \#_E(e)$.*

For the subtraction on multisets we define $E' = E \setminus e$ such that $\#_{E'}(e) = \#_E(e) - 1$ if $\#_E(e) > 0$ and $\#_{E'}(e) = \#_E(e) = 0$ otherwise. The union of sets is defined analogously, i.e. $\#_{E'}(e) = \#_E(e) + 1$. On basis of simple digraphs and multisets we can now define the more general model of multi-digraphs.

Definition 2.3 (Multi-Digraph) *A multi-digraph $G = (V, E, \#_E)$ is defined by a node set $V = \{1, \ldots, n\}$ and a multiset of directed edges $E = \{(u, v) : u, v \in V\}$ with $\#_E$ specifying the multiplicity of the edges.*

In a multi-digraph self-loops (u, u) and multiple occurrence of edges are explicitly allowed, e.g. an edge (u, v) may occur twice. Analogous to simple digraphs, a multi-digraph is called d-regular if

$$\forall u \in V : \sum_{v \in V, (u,v) \in E} \#_E((u, v)) = \sum_{v \in V, (v,u) \in E} \#_E((v, u)) = d$$

and called d-out-regular if

$$\forall u \in V : \sum_{v \in V, (u,v) \in E} \#_E((u,v)) = d \, .$$

Note, that if no self-loops occur and the multiplicity of all edges is at most one then a multi-digraph describes a simple digraph. So, simple digraphs form a subset of multi-digraphs. As in case of simple digraphs we define the neighborhood of a node $v \in V$ as $N^+(v) = \{w : (v,w) \in E\}$. Note that $N^+(v)$ is not a multi-set and therefore that $|N^+(v)| < d$ is possible in a d-out-regular multi-digraph.

The operations we introduce in the first part of this thesis are so called *graph transformations*, i.e. they transform a graph of a specific domain to another graph of that domain. We now give a formal definition of a graph transformation.

Definition 2.4 (Graph Transformation) *A graph transformation is a random transition between the graphs of a specific domain $\mathcal{G}$, e.g. d-regular graphs or multi-digraphs. A graph $G \in \mathcal{G}$ is mapped to a set of other graphs in $\mathcal{G}$ such that*

$$\sum_{G' \in \mathcal{G}} \Pr[G \to G'] = 1,$$

where $G \to G'$ denotes that G is transformed to G'. A graph transformation describes a Markov chain, where the set of states equals the set of graphs in $\mathcal{G}$, i.e. each $G \in \mathcal{G}$ represents a state of the Markov chain. The transition matrix of the Markov chain is given by the transformation probabilities $P[G \to G']$ for all pairs $G, G' \in \mathcal{G}$.

3

The Undirected Case: Flipper

In this chapter we present sound, general, feasible, quickly converging transformations for d-regular random graphs. We introduce a family of graph transformations, namely the k-Flipper operations, for an integer $k \geq 1$ and their randomized versions. Starting with an arbitrary d-regular connected graph we repeatedly apply these operations. Thereby we can guarantee the resulting graph to stay connected and d-regular. Furthermore, these operations will turn any graph into an expander and introduce fresh randomness to the graph which is especially helpful in dynamic graphs with a changing node set. For the Random 1-Flipper we can prove uniform generality, i.e. all connected d-regular graphs occur with the same probability in the limit and as a consequence, expander graphs occur a.a.s. To the best of our knowledge this gives the first solution to the problem of distributed generation of d-regular connected graphs with labeled nodes with uniform probability.

Up to date no tight bounds for the number of Random 1-Flipper operations needed to reach this uniform probability distribution are known. If we choose $k \in \Theta(d^2 n^2 \log 1/\epsilon)$ we can show the convergence rate of the Random k-Flipper. It turns out that after $\mathcal{O}(dn)$ Random k-Flipper operations an expander graph is established with high probability, i.e. $1 - n^{-\Theta(1)}$.

We use these operations to provide a peer-to-peer network based on random regular connected graphs that provides high robustness and recovers from degenerate network structures — which may be caused join operations or network failures — by continuously applying these random graph transformations. For this, we discuss how the concurrency of Flipper operations can be handled and how operations for joining and leaving the network can be designed.

3.1 Uniform Generation of Regular Connected Graphs

In this section we present an elegant method to generate an uniform probability distribution of regular connected graphs in the limit. For this, we introduce the 1-Flipper and its randomized version.

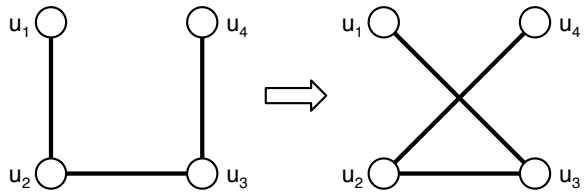


Figure 3.1: *The* 1*-Flipper operation* F_P^1*.*

Definition 3.1 (1-Flipper) *Consider a d-regular undirected graph* $G = (V, E)$ *and four distinct nodes* $u_1, u_2, u_3, u_4 \in V$ *forming a path* $P = (u_1, u_2, u_3, u_4)$ *in* G*. Then, if* $\{u_1, u_3\}$, $\{u_2, u_4\} \notin E$ *the* 1-Flipper *operation* F_P^1 *transforms graph* G *to a graph* $F_P^1(G) = (V, E')$ *with*

$$E' := (E \setminus \{\{u_1, u_2\}, \{u_3, u_4\}\}) \cup \{\{u_1, u_3\}, \{u_2, u_4\}\} .$$

Figure 3.1 illustrates the 1-Flipper operation. We denote $\{u_1, u_2\}, \{u_3, u_4\} \in E$ as *flipping edges* and $\{u_2, u_3\} \in E$ as *hub edge* of the 1-Flipper operation. A randomized version of the 1-Flipper is given by the following algorithm.

Algorithm 3.1 Random 1-Flipper

1: Choose random edge $\{u_2, u_3\} \in E$
2: Choose random node $u_1 \in N(u_2) \setminus \{u_3\}$
3: Choose random node $u_4 \in N(u_3) \setminus \{u_2\}$
4: **if** $\{u_1, u_3\}, \{u_2, u_4\} \notin E$ **then**
5: $\quad E \leftarrow E \setminus \{\{u_1, u_2\}, \{u_3, u_4\}\}$
6: $\quad E \leftarrow E \cup \{\{u_1, u_3\}, \{u_2, u_4\}\}$
7: **end if**

We will now analyze this simple graph transformation. The following lemma shows that the 1-Flipper operation is sound.

Lemma 3.1 *The* 1*-Flipper operation preserves connectivity and d-regularity.*

Proof: Concerning d-regularity note that each node receives one new edge and looses one of its former edges. For connectivity consider two nodes u, v and a path P connecting them. This path can be destroyed, if an edge of P is chosen as flipping edge. However all participating nodes of the 1-Flipper operation remain connected so that another path between u and v can be found. ■

A delimiting factor for applying the 1-Flipper operation is the existence of a triangle such that for a hub edge $\{u_2, u_3\}$ nodes v with $\{u_2, v\}, \{u_3, v\} \in E$ exist. Then neither $\{u_2, v\}$ nor $\{u_3, v\}$ can be chosen as flipping edges. Let $\bigtriangledown_G(u, v)$ be the number of triangles in G with $\{u, v\}$ as an edge. Then the following lemma holds:

Lemma 3.2 *For an arbitrary hub edge* $e \in E$ *there are* $(d - 1 - \bigtriangledown_G(e))^2$ *possibilities to perform a* 1*-Flipper operation that changes the edge set* E*.*

Proof: Assume for a hub edge $e = \{u, v\}$ that edges $\{v, w\}, \{u, w\}$ exist in G, then choosing one of these triangle edges as flipping edges will prevent a change of the edge set. If the flipping edges are not part of such triangles then the 1-Flipper operation will change the edge set. ■

Note that in some graphs for certain hub edges there is no possibility to perform an edge flip at all. In this case all neighbors are connected to both nodes of the hub edge. Then, the 1-Flipper operation has no effect unless another hub edge is chosen. Let $G \xrightarrow{F^1} G'$ denote the predicate that graph G is transformed to G' by a 1-Flipper operation. For the transformation probability between graphs the following lemma holds:

Lemma 3.3

1. *For all $d > 2$ there is a connected d-regular graph G such that* $\Pr[G \xrightarrow{F^1} G] \neq 0$.
2. *For graphs $G' = F_P^1(G)$ with $P = (u_1, u_2, u_3, u_4)$:* $\bigtriangledown_G(\{u_2, u_3\}) = \bigtriangledown_{G'}(\{u_2, u_3\})$.
3. *For graphs $G' = F_P^1(G)$ with $P = (u_1, u_2, u_3, u_4)$:* $\bigtriangledown_G(\{u_1, u_4\}) = \bigtriangledown_{G'}(\{u_1, u_4\})$.
4. *For all undirected regular graphs G, G':* $\Pr[G \xrightarrow{F^1} G'] = \Pr[G' \xrightarrow{F^1} G]$.

Proof:

1. Consider a graph with edges $\{u_1, u_2\}$, $\{u_1, u_i\}$, $\{u_2, u_i\}$ for $i \in \{3, \ldots, d+2\}$ and an arbitrary set of further edges satisfying the d-regularity. If edge $e = \{u_1, u_2\}$ is chosen as hub edge then $\bigtriangledown_G(e) = d - 1$ and from Lemma 3.2 it follows that there is no possibility to perform a 1-Flipper operation that changes E.

2. This property follows by the definition of the 1-Flipper operation. Note that it is not possible to establish or delete triangles containing the hub edge by definition. Of course other triangles can be created or erased. However, they do not count for $\bigtriangledown_{G'}(\{u_2, u_3\})$.

3. The proof is analogous to the proof of Lemma 3.3.2.

4. Note that G and G' differ by exactly four edges connecting four nodes. At least two of these nodes are connected by a hub edge. If exactly two of these nodes are connected by a hub edge e then the probability to perform a successful flip with this hub edge is according to Lemma 3.2 given by $\frac{2}{dn}(1 - \frac{\bigtriangledown_G(e)^2}{(d-1)^2})$ and the probability to transform G directly into G' is given by $\frac{2}{dn}\frac{1}{(d-1)^2}$. From Lemma 3.3.2 the same probabilities follow for the opposite direction $G' \xrightarrow{F^1} G$.

   If there are two hub edges e and e' forming a quadrangle together with the flipping edges the probability to perform a successful flip with these hub edges is $\frac{2}{dn}(2 - \frac{\bigtriangledown_G(e)^2 + \bigtriangledown_G(e')^2}{(d-1)^2})$ and the probability to transform G to G' is given by $\frac{4}{dn}\frac{1}{(d-1)^2}$. Then, the claim follows by applying Lemma 3.3.2 and Lemma 3.3.3.

■

We will now prove that the 1-Flipper operation provides generality.

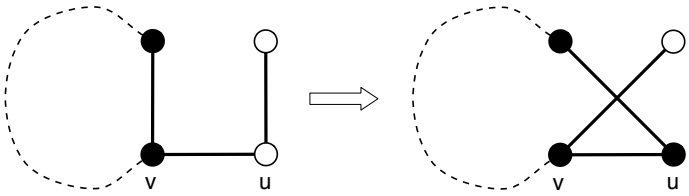


Figure 3.2: *Extending a cycle in G by an* 1*-Flipper operation with* $\{u, v\}$ *as hub edge. Nodes of the cycle are depicted black.*

Lemma 3.4 *For all pairs* G, G' *of connected* d*-regular undirected graphs, with* $d \geq 2$ *and even, there exists a sequence of* 1*-Flipper operations transforming* G *into* G'*.*

Proof: We show that any connected d-regular graph $G = (V, E)$ with $V = \{u_1, \ldots, u_n\}$ can be transformed into a canonical graph $G_C = (V, E_C)$ with edge set defined to be $E_C = \{\{u_i, u_{(i \mod n)+1}\}, \ldots, \{u_i, u_{((i+d/2-1) \mod n)+1}\}\}$ for $1 \leq i \leq n$. From this and the symmetry of the 1-Flipper the lemma follows.

To transform G into G_C we start with making G hamiltonian. Note that every connected regular graph contains a node disjoint cycle, not necessarily containing all n nodes. We extend this cycle to contain all nodes, thus make G hamiltonian. For this we successively add neighboring nodes to the cycle: Let v denote a node of the cycle and let u be a non-cycle node neighboring v. To add u to the cycle we perform a 1-Flipper operation with $\{u, v\}$ as hub edge. The flipping edges are one of v's edges on the cycle and an arbitrary edge incident to u different from $\{u, v\}$. This way u is connected to two neighboring nodes on the cycle and thus the cycle can be extended using u (see Figure 3.2).

It remains to show that no triangles prevent us from applying these 1-Flipper operations. If there exists an edge between u and the end node of the flipping edge lying on the cycle, then u can be incorporated into the cycle without a 1-Flipper operation. For the second flipping edge between u and one of its neighbors we show that u has at least one neighbor which is non-adjacent to v using the d-regularity of G. Node v has already three neighbors: u and two nodes on the cycle. Without v, u has $d - 1$ neighbors which are different from v's neighbors on the cycle (otherwise we do not have to apply the 1-Flipper operation). Furthermore, these $d - 1$ nodes cannot all be adjacent to v since this would imply degree $d + 2$ for v. So there exist flipping edges such that the 1-Flipper operation described above can always be performed. After at most $n - 3$ applications G is hamiltonian.

Having built the Hamilton cycle we bring the nodes of the cycle into the right ordering such that the cycle represents the edges $\{u_i, u_{(i \mod n)+1}\}$ of G_C. Note that applying a 1-Flipper to a path (u_1, u_2, u_3, u_4) interchanges the two inner nodes and results in the path (u_1, u_3, u_2, u_4). If we want to exchange two neighboring nodes u, v of the cycle then we can choose $\{u, v\}$ as hub edge and choose the other cycle edges of u respectively v as flipping edges. This way we can arrange any ordering of the nodes on the cycle.

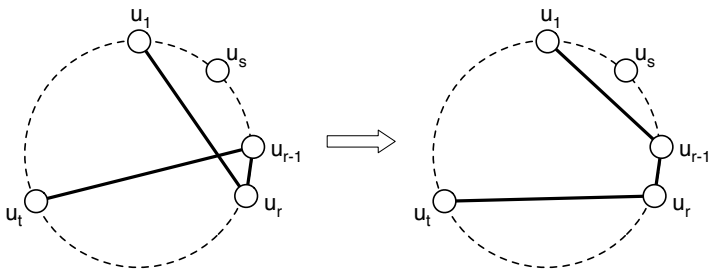


Figure 3.3: *Transforming G into G_C: The endpoint u_r of edge $\{u_1, u_r\}$ is moved in direction of node u_s with a 1-Flipper operation.*

Again we have to show that no triangles prevent these 1-Flipper operations. Therefor consider a 1-Flipper operation applied to four neighboring nodes u_1, u_2, u_3, u_4 of the cycle. This can be done unless edges $\{u_1, u_3\}$ or $\{u_2, u_4\}$ exist. If both of these edges exist we simply redefine the path of the cycle to get the desired ordering. If only one of these edges exists, we assume that $\{u_2, u_4\}$ is the edge preventing the 1-Flipper operation without restriction of generality. In this case we can easily choose another flipping edge instead of $\{u_3, u_4\}$. We need an edge incident to u_3, not incident to u_2 and not part of a triangle over $\{u_2, u_3\}$. The existence of such an edge can be shown using the d-regularity of G as we did above. So, we can always find flipping edges to perform the desired graph transformations.

Having established the edges $\{u_i, u_{(i \bmod n)+1}\}$ we change the remaining edges incrementally according to G_C. To do this, we restrict ourselves to 1-Flipper operations, where the the hub edge is an edge of the cycle and the flipping edges are non-cycle edges. This way the Hamilton cycle remains unchanged. Furthermore, we do not use edges which we have already moved according to G_C as flipping edges, i.e. we do not destroy parts of the graph which we already have adjusted.

We start at node u_1 and establish edges $\{u_1, u_3\}, \ldots, \{u_1, u_{d/2+1}\}$ in this order. Then we do the same for nodes $u_2, \ldots, u_n$. Here we only show how this can be done for node u_1. For the remaining nodes this can be done in the same manner. From the outgoing (non-cycle) edges of u_1 we choose the one whose endpoint is reached first when following the Hamilton cycle in direction of increasing node numbers starting from u_1. Let u_r denote the current and u_s, $s < r$, be the desired endpoint of this edge. We move the endpoint of the edge in decreasing direction of the cycle successively until u_s is reached. The first step is done by applying a 1-Flipper operation to the path (u_1, u_r, u_{r-1}, u_t), with $\{u_{r-1}, u_t\}$ being a non-cycle and not already adjusted edge. This transformation is illustrated in Figure 3.3. The remaining steps towards u_s are done using similar 1-Flipper operations.

Once again these 1-Flipper operations can be blocked by triangles. Note that the first flipping edge $\{u_1, u_r\}$ cannot be part of a triangle over $\{u_r, u_{r-1}\}$ since $\{u_1, u_{r-1}\}$ does not exist in G, otherwise it would have been chosen as flipping edge. However node

u_{r-1} can have no free flipping edge over the hub edge $\{u_r, u_{r-1}\}$. This can only happen if the edge $\{u_{r-1}, u_{r+1}\}$ exists or there is an already adjusted edge pointing to u_{r-1}, since u_{r-1} cannot be a neighbor of u_1. Furthermore, the neighboring cycle edges can also be blocked by triangles. However there will be a non-blocked hub edge in distance $(d-2)/2$, as the following lemma shows.

Lemma 3.5 *In any hamiltonian d-regular graph G with $d > 2$ at most $d-3$ contiguous edges of the Hamilton cycle can be completely blocked by triangles such that no* 1*-Flipper using one of these $d-3$ edges as hub edge and edges lying not on the Hamilton cycle as flipping edges can be applied.*

Proof: Let $\{u_i, u_{(i \bmod n)+1}\}$, $1 \leq i \leq n$, be the edges forming the Hamilton cycle. Without restriction of generality we regard the neighboring nodes $u_1, \ldots, u_{d-2}$ of the Hamilton cycle. To block edge $\{u_1, u_2\}$ u_1 and u_2 have to have $d-2$ common neighbors neglecting their neighbors on the Hamilton cycle. The same argumentation holds for edge $\{u_2, u_3\}$, so that u_3 has also to be connected to these $d-2$ nodes. Continuing to edge $\{u_{d-3}, u_{d-2}\}$ this implies that these $d-2$ neighbors have reached degree d and thus can not have any more neighbors. This implies that edge $\{u_{d-2}, u_{d-1}\}$ can be used as hub edge since there cannot be $d-2$ triangles over $\{u_{d-2}, u_{d-1}\}$. ■

Having such a non-blocked hub edge in distance at most $(d-2)/2$, we can use it to remove a triangle blocking the original considered hub edge as follows. Assume that there are $d-2$ neighboring cycle edges $\{u_k, u_{k+1}\}, \{u_{k+1}, u_{k+2}\}, \ldots, \{u_{k+(d-3)}, u_{k+(d-2)}\}$ blocked by triangles. From Lemma 3.5 we know that $\{u_{k-1}, u_k\}$ has two free flipping edges. Applying a 1-Flipper with hub edge $\{u_{k-1}, u_k\}$ and a triangle edge of u_k and an arbitrary free edge of u_{k-1} as flipping edges will remove one of the triangles over $\{u_k, u_{k+1}\}$, thus make $\{u_k, u_{k+1}\}$ non-blocked. This procedure can be repeated until the desired hub edge is non-blocked.

The way described above we can make any desired hub edge non-blocked and thus apply the desired 1-Flipper operations. The situation is slightly different when we already have adjusted the edges of $n-(d-1)$ nodes. Then there are $d-2$ hub edges left which we will use (this is due to our construction scheme). According to Lemma 3.5 alone it is possible that these $d-2$ edges are blocked. However in our particular case this cannot happen since the endpoints of the blocking triangles would have to lie in parts of G which we have already processed, and this is not possible by definition of G_C. This concludes the proof of Lemma 3.4. ■

Let $G \xrightarrow{i} G'$ denote the predicate that G' is derived from G by applying i Random 1-Flipper operations. Furthermore, let $\mathcal{C}_{n,d}$ denote the set of all connected d-regular graphs with n nodes. The following theorem shows that the Random 1-Flipper operation provides uniform generality.

Theorem 3.1 *Let G_0 be a d-regular connected graph with n nodes and $d > 2$. Then in the limit the Random 1-Flipper operation constructs all connected d-regular labeled graphs with the same probability, i.e.*

$$\lim_{t \to \infty} \Pr[G_0 \xrightarrow{t} G] = \frac{1}{|\mathcal{C}_{n,d}|}.$$

Proof: Consider a Markov process over the set of all connected d-regular graphs described by the Random 1-Flipper. From Lemma 3.3.1 we know that some diagonal entries of the Markov transition matrix are non-zero. From Lemma 3.3.4 we know that the process is symmetric and therefore doubly stochastic. Lemma 3.4 shows that every state of the Markov process can be reached. From this the claim follows by applying essential results of Markov theory. ■

We now give a definition for expander graphs. Expander graphs have a number of advantageous properties, e.g. logarithmic diameter, high vertex connectivity and a small mixing time of random walks (for an excellent survey about expander graphs and their applications see [53]).

Definition 3.2 (Expander graph)

1. *For a graph $G = (V, E)$ and $S, T \subset V$ denote the set of all edges between S and T by*
$$E(S, T) = \{\{u, v\} | u \in S, v \in T, \{u, v\} \in E\}.$$
2. *The* edge boundary *of a set $S \subset V$, denoted by δS, is $\delta S = E(S, \bar{S})$ with $\bar{S} = V \setminus S$.*
3. *A graph $G = (V, E)$ provides* expansion $\beta > 0$*, or is a* β-expander*, if for all node sets S with $|S| \leq |V|/2$ it holds that*
$$|\delta S| \geq \beta |S| .$$

Theorem 3.2 *For $d \in \omega(1)$ a random connected d-regular graph is a $\Theta(d)$-expander graph a.a.s.*

Proof: Note that for fixed $d \geq 3$ any random d-regular graph is connected a.a.s. This follows by independent proofs of Bollobás [16] and Wormald [120] who even prove d-connectivity. Furthermore in [17] it is proved that the isoperimetric number (which is the expansion) of a random regular graph is a.a.s. between $\frac{1}{2}d - \epsilon(d)$ and $\frac{1}{2}d + \epsilon(d)$ where $\epsilon(d) \to 0$ as $d \to \infty$.

Since, nearly all random regular graphs are connected and nearly all of them have an expansion of $\Theta(d)$ it follows that nearly all random regular connected graphs have an expansion of $\Theta(d)$. ■

Corollary 3.1 *For $d > 2$ consider any d-regular connected graph G_0 with n nodes. Then in the limit the Random 1-Flipper operation establishes an expander graph after a sufficiently large number of applications a.a.s.*

As we have seen in this section the Random 1-Flipper constructs expander graphs in the limit. A first bound for the number of operations needed to construct an expander graph, i.e. an upper bound for the mixing time of the Markov chain defined by the 1-Flipper operation, has been given by Feder et al. in [35]. The authors were able to bound the mixing time in terms of the mixing of the switch operation (see Page 2) with help of a Markov chain comparison argument [31, 94, 34] and show that the mixing

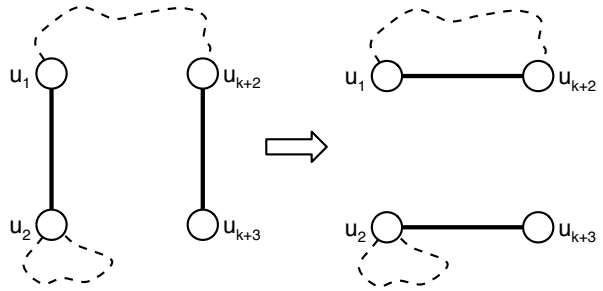


Figure 3.4: *A k-Flipper can disconnect a graph.*

time of the 1-Flipper is bounded by a high degree polynomial. Recently, this result has been improved by Cooper, Dyer, and Handley [28], who show that the mixing time is bounded by $\mathcal{O}\left(d^{20}n^{14}(dn\log dn + \log\epsilon^{-1})\right)$. We conjecture that the mixing time actually is bounded by $\mathcal{O}(dn\log n)$, however this seems to be beyond the reach of current proof techniques. Before we give some experimental evidence in favor of this conjecture in Section 3.4, we will analyze a generalized version of the 1-Flipper.

3.2 Fast Construction of Expander Graphs

In this section we present a generalization of the 1-Flipper operation for which we can show a polynomial bound on the convergence speed towards an expander graph. For this, we extend the hub edge of the 1-Flipper to a path of k edges leading to following definition.

Definition 3.3 (k-Flipper) *Consider a d-regular undirected graph $G = (V, E)$ and $k+3$, $k \in \mathbb{N}$, nodes $u_1, \ldots, u_{k+3} \in V$ forming a path $P = (u_1, \ldots, u_{k+3})$ in G. We call $\{u_1, u_2\}$, $\{u_{k+2}, u_{k+3}\} \in E$ flipping edges and the path $(u_2, \ldots, u_{k+2})$ in G the hub path. If the edges $\{u_1, u_{k+2}\}$ and $\{u_2, u_{k+3}\}$ do not exist in E then the k-*Flipper *operation F_P^k transforms G to $F_P^k(G) = (V, E')$ with*

$$\begin{aligned} E' := \quad & (E \setminus \{\{u_1, u_2\}, \{u_{k+2}, u_{k+3}\}\}) \\ \cup \quad & \{\{u_1, u_{k+2}\}, \{u_2, u_{k+3}\}\}\,. \end{aligned}$$

In contrast to the 1-Flipper, the k-Flipper can disconnect a graph. Figure 3.4 shows a k-Flipper operation which uses the flipping edge $\{u_1, u_2\}$ twice such that the resulting graph is possibly partitioned into disconnected components. In order to preserve connectivity we have to ensure that there is a hub path between nodes u_2 and u_{k+2} of a k-Flipper operation without using the flipping edges. On the other hand we do not want to bias the random walk by forbidding to use the first edge or the a priori unknown last edge.

Fortunately, this problem is easy to handle. A simple solution is to truncate the hub path $P = (u_1, \ldots, u_{k+3})$ to a path $P' = (u_\ell, \ldots, u_{r+1})$ with $1 \leq \ell < r < k+3$ such that

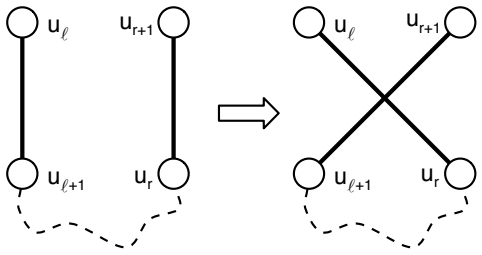


Figure 3.5: *The k-Flipper with truncated hub path.*

$\{u_\ell, u_{\ell+1}\} = \{u_1, u_2\}$, $\{u_r, u_{r+1}\} = \{u_{k+2}, u_{k+3}\}$ and $\{u_1, u_2\}, \{u_{k+2}, u_{k+3}\}$ do occur only once in P' (see Figure 3.5). This observation leads to the following algorithm.

Algorithm 3.2 Random k-Flipper

```
1: Choose random node u_1 ∈ V
2: for i ← 1 to k + 2 do
3:     Choose random node u_{i+1} ∈ N(u_i)
4: end for
5: for i ← k + 2 downto 2 do
6:     if {u_i, u_{i+1}} = {u_{k+2}, u_{k+3}} then r ← i
7:     end if
8: end for
9: for i ← 1 to r do
10:    if {u_i, u_{i+1}} = {u_1, u_2} then ℓ ← i
11:    end if
12: end for
13: if r ≥ ℓ + 2 and {u_ℓ, u_r}, {u_{ℓ+1}, u_{r+1}} ∉ E then
14:    E ← E \ {{u_ℓ, u_{ℓ+1}}, {u_r, u_{r+1}}}
15:    E ← E ∪ {{u_ℓ, u_r}, {u_{ℓ+1}, u_{r+1}}}
16: end if
```

We continue with the analysis of the Random k-Flipper.

Lemma 3.6 *The Random k-Flipper operation preserves connectivity and d-regularity.*

Proof: For d-regularity the same arguments as in Lemma 3.1 hold. Applying the edge flip to the truncated path $P' = (u_\ell, \ldots, u_{r+1})$ it is ensured that the graph will stay connected since no edges of the truncated hub path will be removed. ■

Similar to the 1-Flipper operation, the Random k-Flipper operation provides generality.

Lemma 3.7 *For all pairs G, G' of connected d-regular undirected graphs there exists a sequence of Random k-Flipper operations transforming G into G'.*

Proof: Note that a Random k-Flipper operation can be reduced to a 1-Flipper operation if the path at the beginning uses the start edge $k-1$ times. In Lemma 3.4 we have proved that this property holds for the 1-Flipper. ■

As opposed to the Random 1-Flipper, the Random k-Flipper is not a symmetric graph transformation. Therefore, it is not clear if the Random k-Flipper provides uniform generality. Nevertheless, it establishes an expander graph in a polynomial number of rounds which we prove now. For this, we start with a bisection of the node set of a graph $G = (V, E)$ into $S \subset V$ and $\bar{S} = V \setminus S$ with $|S| \leq |V|/2$. Let $|V| = n$, $|S| = m$ and $|E(S, \bar{S})| = q$ be the number of edges of the cut. We are interested in the number q' of edges between S and $\bar{S}$ after applying a Random k-Flipper operation. Now assume that each edge is chosen with uniform probability $\frac{2}{dn}$ as a flipping edge. This assumption is motivated by the fact that a long random walk in the graph will choose flipping edges with uniform probability. Then there are the following cases:

1. Both flipping edges are chosen from $E(S, S)$. Then the Random k-Flipper operation will not increase the number of edges of the cut, i.e. $q' = q$.
2. Both flipping edges are chosen form $E(\bar{S}, \bar{S})$. Again $q' = q$.
3. One flipping edge is in $E(S, S)$ and one is in $E(\bar{S}, \bar{S})$. This will occur with probability $\frac{2(dm-q)(dn-dm-q)}{d^2n^2}$ and two edges are added to the cut, i.e. $q' = q + 2$.
4. One flipping edge is in $E(S, \bar{S})$ and one is in $E(S, S)$. Then $q' = q$.
5. One flipping edge is in $E(S, \bar{S})$ and one is in $E(\bar{S}, \bar{S})$. Then $q' = q$.
6. Both flipping edges are in $E(S, \bar{S})$. This happens with probability $\left(\frac{2q}{dn}\right)^2$. In this case the number of edges on the cut can be decreased by two or stay the same, i.e. $q' \in \{q-2, q\}$. However, it is guaranteed from the connectivity property that $q' \geq 1$.

This proves the following lemma.

Lemma 3.8 *Consider a bisection of a graph $G = (V, E)$ into $S \subset V, \bar{S} = V \setminus S$, $|S| \leq |V|/2$ and let $|V| = n, |S| = m$ and $|E(S, \bar{S})| = q$. If the flipping edges are chosen with uniform probability then a Random k-Flipper operation transforms q to q' as follows:*

$$\begin{aligned} \Pr[q' = q-2] &\leq \left(\frac{2q}{dn}\right)^2 \\ \Pr[q' = q+2] &= 2\left(\frac{m}{n} - \frac{q}{dn}\right)\left(1 - \frac{m}{n} - \frac{q}{dn}\right) \end{aligned}$$

So, for a given partition the Random k-Flipper operations describe a random drift towards a state that satisfies an expansion. However, one cannot guarantee a truly uniform choice of the flipping edges. Yet, if the random walk is long enough an approximation can be provided.

Lemma 3.9 *For $k \in \Theta\left(d^2n^2 \log 1/\epsilon\right)$ the Random k-Flipper chooses the first flipping edge with uniform probability and the second edge with probability $(1 \pm \epsilon)\frac{2}{dn}$ for any $\epsilon > 0$.*

Proof: Consider the conductance Φ of a d-regular graph defined by

$$\Phi := \min_{S \subset V, S \neq \emptyset} \frac{n|\delta S|}{d|S| \cdot |\bar{S}|} .$$

As a lower bound for the conductance of connected graphs we have

$$\Phi \geq \min_{m \in \{1,\ldots,n-1\}} \frac{n}{dm(n-m)} = \frac{4}{dn} .$$

According to Lovász [69] the second eigenvalue λ_2 of the Markov process is bounded by

$$\frac{\Phi^2}{8} \leq 1 - \lambda_2 \leq \Phi ,$$

which implies a bound of $\lambda_2 \leq 1 - \frac{2}{d^2n^2}$. Let $P^t(v)$ denote the probability that a random walk ends at node v after t rounds. Then, this implies

$$\left| P^t(v) - \frac{1}{n} \right| \leq \left(1 - \frac{\Phi^2}{8}\right)^t \leq \left(1 - \frac{2}{d^2n^2}\right)^t .$$

So, after $d^2n^2 \log 1/\epsilon$ rounds in all graphs this term is smaller than ϵ. This proof follows the ideas presented in [69]. ∎

Combining Lemma 3.8 and 3.9 we get the following lemma.

Lemma 3.10 *Consider a bisection of a graph $G = (V, E)$ into the sets $S \subset V$ and $\bar{S} = V \setminus S$, $|S| \leq |V|/2$, with $|V| = n, |S| = m$ and $|E(S, \bar{S})| = q$. Applying a Random k-Flipper operation with $k \in \Theta\left(d^2n^2 \log 1/\epsilon\right)$ and $\epsilon \in (0, \frac{1}{8}]$ the number of edges of the cut is changed from q to q' such that*

$$\begin{aligned} \Pr[q' = q - 2] &\leq 4(1+\epsilon)\alpha^2 \\ \Pr[q' = q + 2] &\geq 2(1-\epsilon)(\beta - \alpha)(1 - \beta - \alpha) \end{aligned}$$

with $\alpha = \frac{q}{dn}$ and $\beta = \frac{m}{n}$.

Proof: The proof follows by adapting the approximation bound $(1 \pm \epsilon)$ of Lemma 3.9 to Lemma 3.8. ∎

Note that for $\alpha \ll \beta$ we observe a random walk with a strong drift over the number of edges. The probability that two edges are added is larger than the probability that two edges are removed. So, we can reduce the analysis to a random walk with a drift and use the following lemma.

Lemma 3.11 *Consider a random walk on the set of numbers $\{1, \ldots, B\}$ with transition probability $2p$ from i to $i+1$ for $i < B$ and probability p from i to $i-1$ for $i > 1$. The probability to remain in state i is $1 - 3p$ for $i \in \{2, \ldots, B-1\}$, $1 - 2p$ for state 1 and $1 - p$ for state B.*

For any $c > 0$ there exists c' such that after $c'B/p$ rounds the probability that the random walk ends within the set $\{1, \ldots, B - t\}$ is at most $2^{-cB} + 2^{-t+1}$.

Proof: In the worst case the random walk starts at position 1. The expected distance the random walk upwards within k rounds is pk. If we choose $k = c'B/p$ we can apply Chernoff bounds to prove that with probability $1 - 2^{-cB}$ the position B is reached at least once.

Now consider the stationary distribution, when the Markov process has converged. Let P_t be the probability for state $B - t$. For $t > 1$ we have the recursion

$$P_t = 2pP_{t+1} + (1 - 3p)P_t + pP_{t-1}.$$

This recursion is satisfied for $P_t = \gamma 2^{-t-1}$. Furthermore, this implies: $P_0 = 2P_1 = \gamma/2$. Summing up over all positions and using $P_{B-2} = 2P_{B-1}$ we have $\gamma = \frac{1}{1-2^{-B-1}}$ and the probability that the random walk is in the interval $\{1, \ldots, B - t\}$ after reaching B is at most 2^{-t+1}, which implies the claim. ∎

Now we are able to prove the fast convergence of the Random k-Flipper.

Theorem 3.3 *If we choose $d \in \Omega(\log n)$ applying $\mathcal{O}(dn)$ Random k-Flipper operations with $k \in \Theta\left(d^2n^2 \log 1/\epsilon\right)$ transforms any given d-regular connected graph into a connected d-regular graph with expansion $\Theta(d)$ with high probability.*

Proof: We will prove an expansion of at least $d/16$. For this we consider all sub-sets $S \subset V, S \neq \emptyset$ with $|S| = m \leq n/2$. Now we apply Lemma 3.10 with a constant choice for ϵ for a bisection where the edge set is at most twice than the expansion, i.e. $q \leq md/8$. Then $\alpha = \frac{q}{dn} \leq \frac{m}{8n} = \frac{1}{8}\beta$ and $\beta \leq \frac{1}{2}$. This implies for $\epsilon = \frac{1}{8}$:

$$\begin{aligned} 2(1-\epsilon)(\beta-\alpha)(1-\beta-\alpha) &\geq 2(1-\epsilon)\frac{7}{8}\beta\left(1-\frac{9}{8}\beta\right) \\ &\geq \frac{63}{64}(1-\epsilon)\beta \\ &\geq \frac{63}{32}(1-\epsilon)\beta^2 \\ &\geq 126(1-\epsilon)\alpha^2\,. \end{aligned}$$

Again let q' denote the number of edges of the cut after applying one Random k-Flipper operation. Then, we have

$$\Pr[q' = q+2] \;\geq\; 94\Pr[q' = q-2]$$

and

$$\Pr[q' = q+2] \;\geq\; \frac{1}{2}\frac{m}{n}\,.$$

So, for the number of boundary edges we observe a random walk with a drift. The process that we have studied in Lemma 3.11 gives an upper bound on the number of Random k-Flipper operations for $p = \frac{m}{4n}$ and $B = \frac{md}{4}$ (we spare a factor of two since we walk two steps in a round). Hence, with probability $1 - (2^{-cB} + 2^{-\frac{1}{2}B})$ the number

of edges is at least $\frac{md}{16}$ within this cut after $c'B/p \leq c'\frac{md}{4}\frac{4n}{m} \leq c'dn$ rounds of Random k-Flipper operations.

Let $c \geq \frac{1}{2}$. It remains to sum up all error probabilities $2^{-cB} + 2^{-\frac{1}{2}B} \leq 2^{-\frac{1}{2}B} + 2^{-\frac{1}{2}B} = 2 \cdot 2^{-\frac{md}{4}}$ of all sub-sets $S \subset V$ of size $m \leq n/2$ for some $d = k \log n$ with $k > 8$.

$$\begin{aligned}
2\sum_{m=1}^{n/2} \binom{n}{m} 2^{-\frac{1}{4}km\log n} &\leq 2\sum_{m=1}^{n/2} n^m 2^{-\frac{1}{4}km\log n} \\
&\leq 2\sum_{m=1}^{n/2} 2^{(1-\frac{1}{4}k)m\log n} \\
&\leq 2\sum_{m=1}^{n/2} n^{(1-\frac{1}{4}k)m} \\
&\leq 2\frac{n}{2} n^{1-\frac{1}{4}k} \\
&\leq n^{-k'}.
\end{aligned}$$

■

Possible Improvements

As noted above the Random k-Flipper is not symmetric, i.e. the transition probability from graph G to G' may differ from the transition probability from G' to G. Therefore, it is an open problem if the Random k-Flipper with $k > 1$ provides uniform generality. Nevertheless, slightly modified versions provide symmetry. For this, the random walk needs to avoid to traverse the flipping edges more than once. The Random k-Flipper is allowed to traverse these edges and a sub-path of the random walk will be chosen. This choice causes the break of symmetry. If we avoid visiting nodes more than once we have the *Node Disjoint Random k-Flipper* and if we avoid visiting edges more than once we get the *Edge Disjoint Random k-Flipper*. Both operations have symmetric transition probabilities. However, when using these operations long random walks are not possible, especially not paths of length $\Theta\left(d^2n^2 \log 1/\epsilon\right)$. Furthermore, the proof of Theorem 3.3 cannot be applied since the random walk used in these modified versions is rather biased. So, the practical use of these symmetric versions is doubtable.

As soon as the expansion property is established one can reduce the length of the random walk of the Random k-Flipper to a polylogarithmic term. Furthermore, it maybe that such a random walk suffices for the whole procedure.

3.3 Peer-to-Peer Networks based on Random Regular Graphs

The operations introduced in the previous sections are particularly suitable for building and maintaining large distributed random networks. In this section we will discuss

the networking aspects like dynamics (joining and leaving peers) and problems arising with the concurrent use of Random k-Flipper operations in a distributed network. In particular we show how to maintain dynamic connected d-regular peer-to-peer networks based on random graphs with expansion property.

For this the Random k-Flipper operations are started distributedly by every peer from time to time and control messages are sent over random paths and neighbors are exchanged. This way continuously fresh randomness is introduced to the network and with help of the parallel operations the network quickly converges to an expander graph. Furthermore, the network connections are validated automatically by the random walks of the k-Flipper operations.

Unlike as in the graph approach, in distributed dynamic networks it is difficult to guarantee the d-regularity, e.g. if only one edge is missing, then there are only two nodes with degree $d-1$ while the other nodes have degree d. Finding the partner would involve a search (or a data structure) for the whole network. Yet, the benefit of having exactly d neighbors is rather small if d is at least logarithmic, what is characteristic for peer-to-peer networks. Therefore, we allow nodes to have either d or $d-1$ neighbors and call such networks $\{d, d-1\}$-regular.

3.3.1 Joining Peers

A joining peer has to connect to d other peers to get a valid neighborhood. The easiest way for a peer to do this without violating the $\{d, d-1\}$-regularity of the network is to randomly choose $d/2$ connections defined by d distinct nodes, erase these connections and connect to each of the d nodes: "The peer places itself in the middle of these connections". The distinctness of the d peers of these connections is crucial, since otherwise the new peer would either create multiple connections to some peers or reduce the degree of some peers violating the $\{d, d-1\}$-regularity. The 'randomness' of the chosen connections on the other hand is of subordinate importance since even the worst choice of connections will be corrected by the periodic Flipper operations.

So, we have to find enough random connections in reasonable time, since no peer has a global view of the network we propose to do this in a local way with help of a single or up to $d/2$ concurrent random walks. We illustrate the local join operation with a single random walk in the following and assume a peer u willing to join the network knows at least one peer v within the network. Starting from v, u sends a control message on a random walk, see Figure 3.6. Whenever a suitable connection e is found a lock is created on e to prevent other peers from selecting this connection. Furthermore, these locks prevent Flipper operations from choosing these connections as flipping edges. According to [9] the length of the random walk can be chosen as $\mathcal{O}\left(d^2 \log d\right)$. In the expectation this number of hops is sufficient to detect d distinct nodes in any graph by a random walk. If after $\mathcal{O}\left(d^2 \log d\right)$ hops not enough random connections could be found the random walk is canceled. This process can fail if either the network is too small or if the network structure is bad (which is unlikely, yet possible). A new peer which could not find enough neighbors can retry finding new neighbors by the same

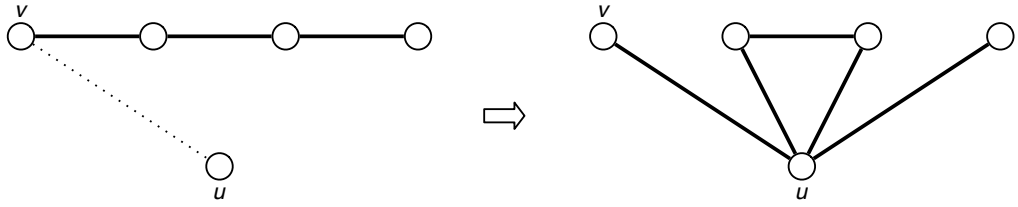


Figure 3.6: *The local join operation when $d = 4$.*

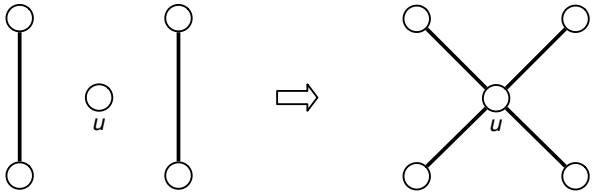


Figure 3.7: *The pegging operation when $d = 4$.*

procedure after some time. If the control message has successfully finished the random walk reporting enough connections, u is contacted by enough peers of the network and can place itself in the middle of each of the connections.

As this scheme is somehow obvious to join a d-regular network, almost equal schemes have been proposed by several researchers, e.g. [19, 26] and implicitly in [64]. Bourassa and Holt [19] proposed a join operation called *pegging* to construct the d-regular unstructured network SWAN for example. The pegging operation is illustrated in Figure 3.7 and the only difference to the join operation described above is that pegging assumes the $d/2$ connections to be chosen uniformly at random. In the SWAN network it is crucial that the chosen edges are indeed (almost) truly random since there is no maintenance operation, i.e. the topology of the SWAN network is changed by joining and leaving peers only.

It is easy to see that a network topology created by the pegging operation is not a truly random connected d-regular graph, for example it is impossible to create cliques of size greater than $d/2 + 1$ with pegging operations. Nevertheless they share some desirable characteristics with random regular graphs [41, 40]. So, it was recently shown by Gao [40] that the evolving graphs are a.a.s. d-connected for any even constant degree $d \geq 4$. Furthermore, the following theorem does hold.

Theorem 3.4 *A d-regular graph G with $d \in \Omega(\log n)$ constructed using the pegging operation is a $\theta(d)$ expander with high probability.*

We will not prove this theorem here since we give an identical result for directed networks in Chapter 4 with Theorem 4.5 and the proof is almost identical.[1] As mentioned

[1] The proof of Theorem 3.4 can be found in [51]. Note however that the proof in [51] has been derived from the proof of Theorem 4.5 presented in this thesis.

in context of choosing flipping edges in the previous chapter already, it is difficult to choose edges uniformly at random in a peer-to-peer network. Therefore, we propose to not rely on the insertion scheme alone to form respectively maintain the topology as it is done in SWAN, but to use random Flipper operations to correct "bad choices" that may have been made by the insertion scheme.

3.3.2 Leaving Peers

The case of leaving peers can be divided in peers leaving intentionally and peers leaving by some kind of local or network failure. We will consider the case of nodes leaving intentionally first. It turns out that this case is easy to handle. A leaving peer u successively selects two random peers v and v' of its neighborhood. Then v and v' are informed that u is about to leave. The peers v and v' then connect to each other and remove u from their neighborhood. Next, u removes v and v' from its neighborhood and continues with the next pair of neighbors until all neighbors are processed. This procedure ensures that the network still has degree d or $d-1$ (provided that the network consists of enough peers).

The case of peers failing unexpectedly is more problematic. First of all, failing peers bear the problem of disconnecting the network. Classical failure analysis in peer-to-peer networks is focused on analyzing the probability that a given peer becomes disconnected [114, 68], so that we will restrict to this case, too.

Fact 3.1 *Let G be a random d-regular network of size n with $d \in \Omega(\log n)$. When each peer fails with probability $1/2$, then a single peer will stay connected to the rest of the network with high probability.*

Proof: To separate a peer from the network all of his $d \in \Omega(\log n)$ neighbors have to fail. This will happen with probability $2^{-d} \leq 2^{-c \log n} = n^{-c}$. ∎

Another issue is that a failing peer reduces its neighbors degree by one. As noted above we do not want to fix this degree by choosing pairs since this causes too much communication overhead. Our solution is that a neighbor detects an edge failure while sending control information for the periodically occurring Random k-Flipper operations. If the degree is smaller than $d-1$ then a peer uses the node joining algorithm to increase its degree by the missing number of edges, i.e. it chooses random connections of the network and places itself in the middle of these connections, thus increasing its own degree by two without changing the degree of other peers.

3.3.3 Concurrency

When multiple Random k-Flipper operations are applied to a graph concurrently (which is likely when the graph represents a huge network) some additional effort is necessary to guarantee connectivity. In case of the Random 1-Flipper operation there is only a small constant number of nodes involved in intersecting operations. So, these can coordinate their graph transformations in a way that no two intersecting transformations take place at the same time.

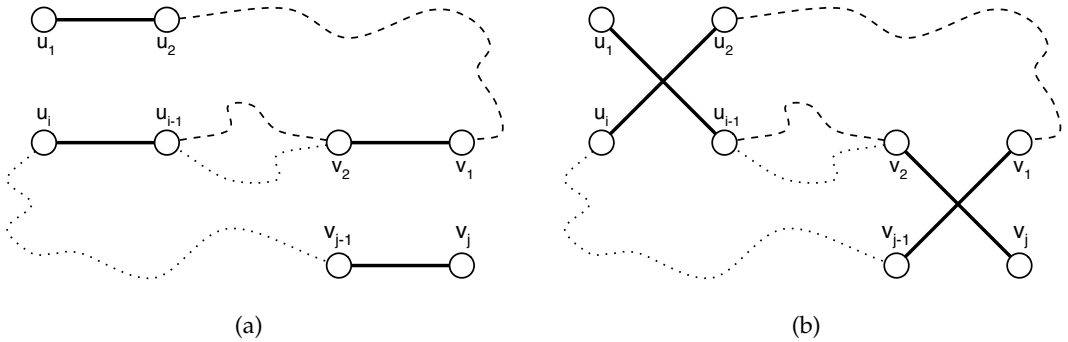


Figure 3.8: *Concurrent Random k-Flipper operations can disconnect a graph. The graph at the left shows random walks performed by two Random k-Flipper operations U (dashed) and V (dotted), which result in a disconnected graph shown at the right.*

However some additional effort is necessary to guarantee connectivity in case of concurrent Random k-Flipper operations without central coordination mechanisms. We start our analysis with the following fact.

Fact 3.2 *The only way to divide a graph into components with Random k-Flipper operations is to destroy the hub path of a Random k-Flipper operation. Furthermore, the only way to remove edges with Random k-Flipper operations is to choose these edges as flipping edges.*

In the following we will consider two concurrent Random k-Flipper operations U and V. Let $u_1, \ldots, u_i$ respectively $v_1, \ldots, v_j$ denote their truncated random walks according to the Random k-Flipper algorithm (see Algorithm 3.2).

Lemma 3.12 *The hub path of a Random k-Flipper operation U cannot be destroyed by another Random k-Flipper operation V if V does not use the flipping edges $\{u_1, u_2\}, \{u_{i-1}, u_i\}$ of U.*

Proof: When V chooses one or both of its flipping edges to lie on the path $(u_2, \ldots, u_{i-1})$ this will only substitute the edges to be deleted by V by a path. Thus the network will stay connected. ■

Things get more complicated when U and V interfere each other, i.e. U has an flipping edge on the path $(v_2, \ldots, v_j)$ and V has a flipping edge on the path $(u_2, \ldots, u_i)$.[2] This may lead to a disconnected graph as shown in Figure 3.8.

However the partition of the graph does not necessarily take place since there may be other independent paths between u_1 and u_2. While the probability of the disconnecting the graph is rather small, we aim at guaranteed connectivity. In our view this is important because on the long run also small probability events will surely occur – regardless of their small probability.

[2] Note that we can guarantee $u_1 \neq v_1$ since no peer will start another Random k-Flipper operation before the previous one is finished.

So, we use the following locking mechanism. Each Random k-Flipper operation U leaves a *stamp* — a representation of its starting edge — at each edge it passes. In addition U keeps track of the edges it has visited during the random walk in a list L. When the random walk is finished, it is checked whether one of the flipping edges $\{u_1, u_2\}, \{u_{i-1}, u_i\}$ has a stamp $s \in L$ on it. In this case another Random k-Flipper with starting edge on U's random walk has already passed one of U's flipping edges and U is risking connectivity if the edges are flipped. The solution to this is rather simple. If the last edge $\{u_{i-1}, u_i\}$ has critical stamps on it U can do another random step or choose another edge of the last but one node and check again for stamps in L. Critical stamps on the starting edge $\{u_1, u_2\}$ can be handled similar. If no suitable edges can be found the Random k-Flipper operation is canceled. To prevent the whole graph from getting locked, the stamps should be soft state and be deleted after some constant time t. This implies that also all Random k-Flipper operations have to be finished in time t or they will have to be canceled. Given this stamp mechanism the following lemma holds.

Lemma 3.13 *Using random walks with stamps as described above, no two concurrent Random k-Flipper operations can interfere with each other.*

Proof: Again consider two Random k-Flipper operations U and V. For the proof we will regard the path $(u_1, \ldots, u_i)$ of U as fixed. Furthermore, V has one flipping edge on the path $(u_1, \ldots, u_i)$ and thus interferes U. We assume this to be the starting edge $\{v_1, v_2\}$ of V. Note that we can restrict to this case because of symmetry. In order to make U and V interfere with each other, V has to pass at least one flipping edge of U. When V passes one of these edges it will leave a stamp on it. This will make U to find a stamp representing an edge of the path $(u_1, \ldots, u_i)$ on it when checking the flipping edges $\{u_1, u_2\}, \{u_{i-1}, u_i\}$. Thus U will not perform the Random k-Flipper operation. ■

Combining our results the following theorem holds.

Theorem 3.5 *Using random walks with stamps the network is guaranteed to stay connected while applying concurrent Random k-Flipper operations.*

Proof: This theorem follows directly from the combination of Lemma 3.12 and Lemma 3.13. ■

For peer-to-peer networks it turns out that the Random 1-Flipper is easier to handle than the Random k-Flipper with larger k. Especially if $k \in \Theta\left(d^2 n^2 \log 1/\epsilon\right)$ is chosen as in Theorem 3.3 then the whole network is swamped with stamps of one type. Parallel Random k-Flipper operations may block each other or in the best case are performed sequentially. For the networking point a small choice for k is highly desirable. However, little is known about the convergence rate except for the case of expander graphs. In an expander graph $k \in \mathcal{O}(\log n)$ can be used.

3.4 Experimental Evaluation

As mentioned above, the convergence rate of the 1-Flipper can be bounded by a high degree polynomial, i.e. $\mathcal{O}\left(d^{20} n^{14} (dn \log dn + \log \epsilon^{-1})\right)$ [28]. In this section we present

experimental results for the 1-Flipper operation which indicate that this bound is far apart from the true bound. Furthermore, we will experimentally compare the performance of 1-Flipper and k-Flipper.

There is no method known which can test the true randomness of a graph. Therefore, we concentrate on the expansion $h(G)$ of the graph. The exact determination of $h(G)$ for a given graph G is known to be co-*NP*-hard [14]. Fortunately it is possible to bound the expansion with help of eigenvalues. Therefore, consider a d-regular graph G with n nodes and its $n \times n$ adjacency matrix $A(G)$. The entries a_{ij} of $A(G)$ are 1 if node i and j are adjacent and 0 otherwise. Since $A(G)$ is symmetric and real there are n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, also denoted as *spectrum* of the graph G. The following theorem gives an upper and a lower bound for $h(G)$ using the eigenvalue λ_2.

Theorem 3.6 *Let G be a d- regular graph with spectrum $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then*

$$\frac{d-\lambda_2}{2} \;\leq\; h(G) \;\leq\; \sqrt{2d(d-\lambda_2)}.$$

Theorem 3.6 has been proven by several researchers, i.e. Dodziuk [32], Alon-Milman [5], and Alon [4] in the discrete case and Cheeger [23] and Buser [20] in the continuous case. The term $d - \lambda_2$ is also known as *spectral gap* and according to Theorem 3.6 a large spectral gap implies a high expansion.

Knowing that we can bound the expansion of a graph G by calculating its spectral gap we still need to define when we should consider a graph as an expander based on its spectral gap only. The following theorem (cf. [83, 53, 38]) will allow us to give an upper bound on the spectral gap.

Theorem 3.7 (Alon-Boppana) *For every d-regular graph it holds that*

$$\lambda \geq 2\sqrt{d-1} - o_n(1)$$

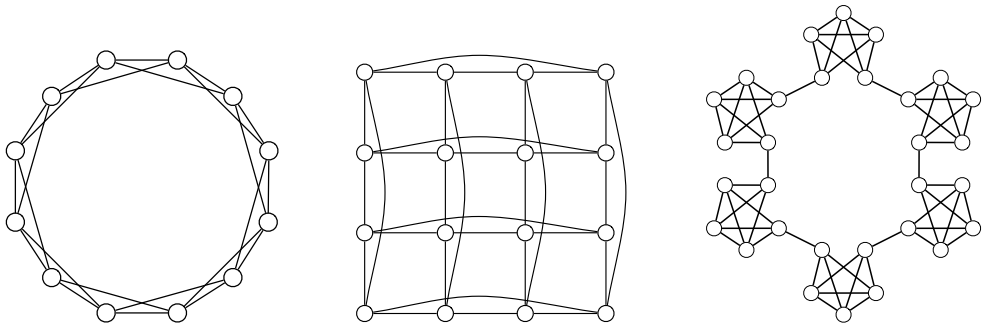
where $\lambda = \max(|\lambda_2|, |\lambda_n|)$ and $o_n(1)$ tends to zero for fixed d as $n \to \infty$.

It follows that $d - \lambda_2 \leq d - 2\sqrt{d-1}$ which reduces the interval for the spectral gap from $[0, 2d]$ to $[0, d - 2\sqrt{d-1}]$.[3] Combining the results of Theorem 3.6 and 3.7 we may use the function $\rho(d) = \frac{d-2\sqrt{d-1}}{2}$ to approximate the lower bound for the expansion ratio of a graph from above. So, we use the following definition to assess a graph G as an expander based on the eigenvalues of $A(G)$ only.

Definition 3.4 *We assess a d-regular graph G as an expander graph when the lower-bound $\frac{d-\lambda_2}{2}$ of $h(G)$ satisfies*

$$\frac{d-\lambda_2}{2} \;\geq\; 0.98\,\rho(d).$$

[3] We neglect the term $o_n(1)$ here since we are interested in large graphs.



(a) ring (b) torus (c) ring of cliques

Figure 3.9: *Starting graphs used in the experiments.*

To measure the number of Flipper operations needed to transform a graph G into an expander graph we repeatedly calculate the spectral gap of G and hence the second largest eigenvalue λ_2 of $A(G)$ while applying Flipper operations. To do this, we used the simulation environment developed in [51], which uses the PRIMME (PReconditioned Iterative MultiMethod Eigensolver) [107, 108] by McCombs and Stathopoulos for eigenvalue calculations.

Clearly, the time to transform a graph G into an expander depends on the structure of G. For our experiments we selected three different starting graphs:

Ring The ring graph consists of a cycle containing all n nodes and for $d > 2$ of additional edges connecting nodes in distance $2, 3, \ldots, d/2$ on the ring to reach the desired degree d.

Torus A two dimensional mesh with wrap around edges (we restrict ourselves to the two dimensional case here).

Ring of Cliques The ring of cliques graph of degree d consists of a number of $(d+1)$-cliques. These cliques are then connected to form a ring by removing one edge from every clique and connecting the endpoints of the removed edges to form a ring.

The three graph types are illustrated in Figure 3.9. These graph types were selected for two reasons: they have a relatively large diameter and, in case of the ring and ring of cliques, they contain a lot of triangles which prevent 1-Flipper operations from being applied successfully and therefore will delay the convergence process towards an expander graph (recall that a 1-Flipper operation does not change the graph if one or both flipping edges are part of triangles over the hub-edge).

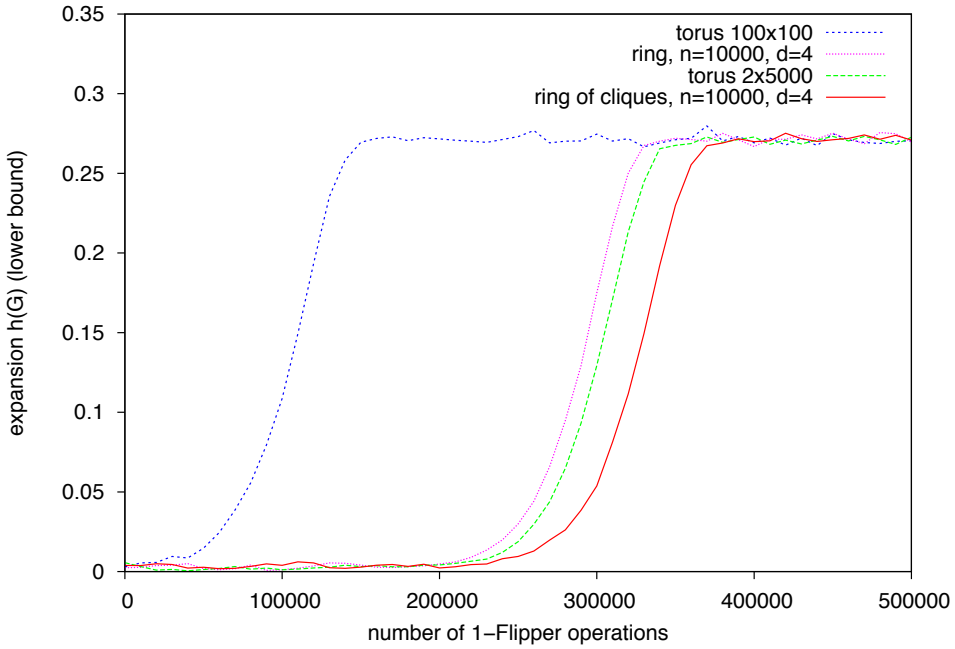


Figure 3.10: *The start graph influences the convergence speed of the* 1*-Flipper operation.*

3.4.1 Results for the 1-Flipper

If not mentioned otherwise experiments have been repeated ten times respectively and the experimental results presented in the remainder of this section show the average of these test runs.

Figure 3.10 shows the evolution of the lower bound of the graph expansion $h(G)$ for the 1-Flipper on the described graph types. All graphs have degree $d = 4$ and $n = 10,000$ nodes. The shapes of the four curves are very similar, yet shifted on the x-axis. In our experiments the ring of cliques needed the largest number of 1-Flipper operations to be transformed into an expander graph. Figure 3.11 exemplifies the convergence for a single test instance starting with the $10,000$ nodes 4-regular ring of cliques. We observe the following three phases:

Initial Phase Steady small expansion with only minor changes in the lower bound of the expansion (first $250,000$ Flipper operations).

Expansion Phase A short period in which the expansion of the graph grows dramatically and reaches $0.98\,\rho(d)$ (Flipper operations $250,000 - 360,000$).

Stable Phase The maximal expansion has been reached (after $360,000$ Flipper operations). Once this phase has been reached, the expansion is stable and stays on this high level.

Figure 3.12 shows that during the initial phase the graph is modified already: The

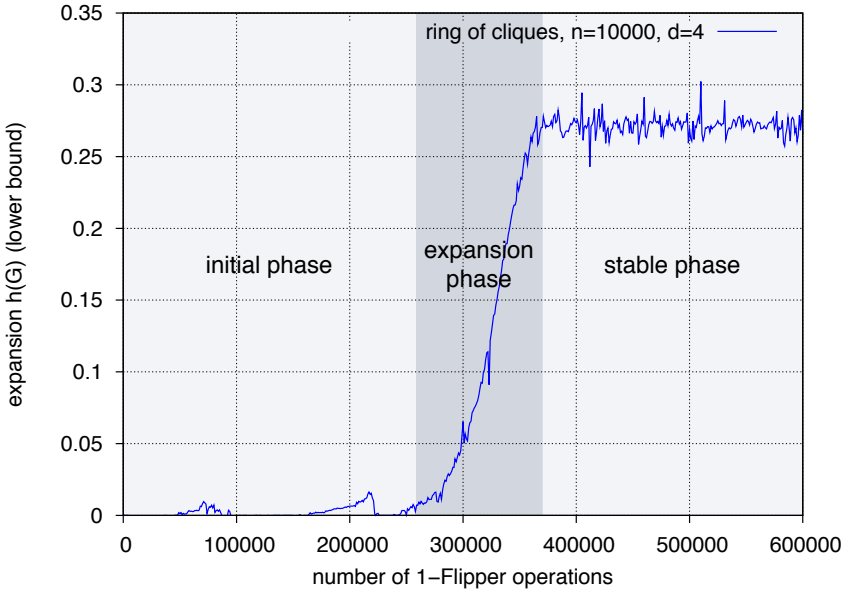


Figure 3.11: *The three convergence phases of the* 1*-Flipper operation.*

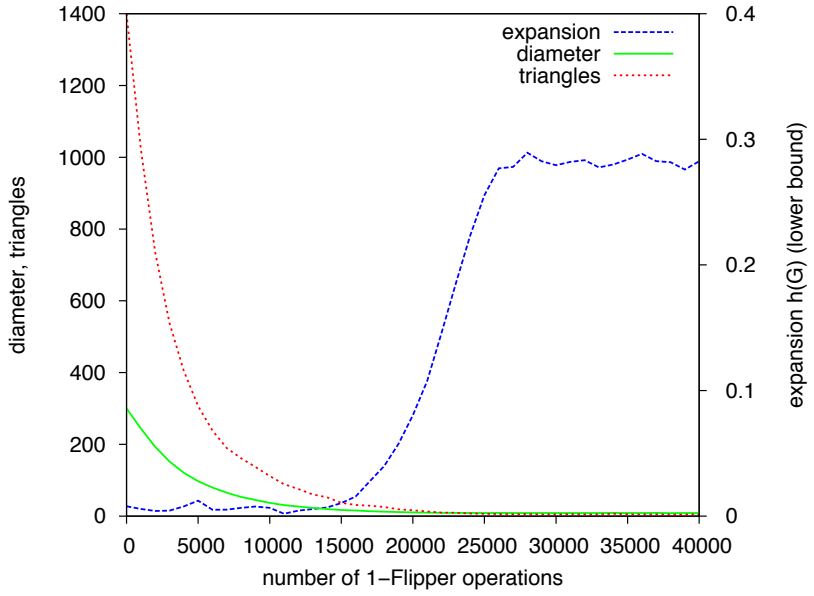


Figure 3.12: *Evolution of expansion (lower bound), diameter, and triangles for* 1*-Flipper operation started on a* 1,000 *node* 4*-regular ring of cliques.*

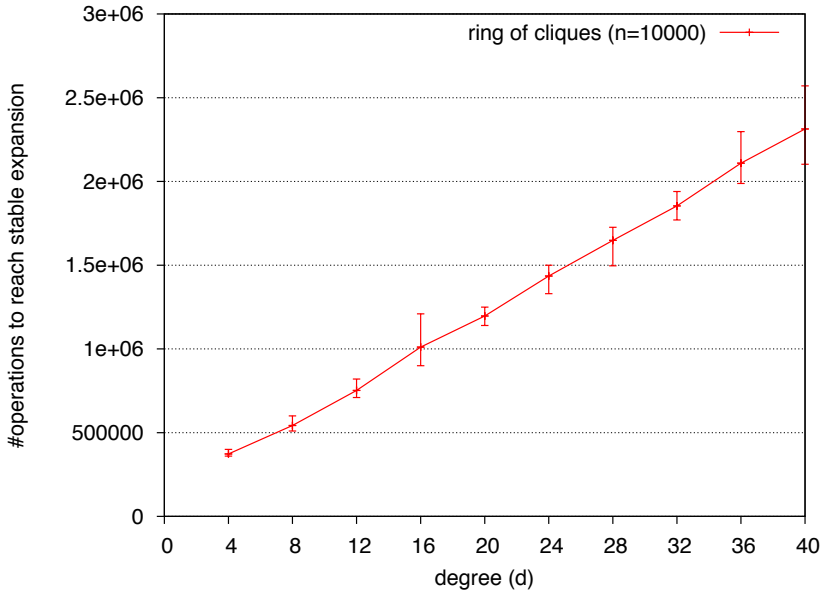


Figure 3.13: *Number of* 1*-Flipper operations to reach stable expansion for variable degree* d*, starting from a* $10,000$ *nodes ring of cliques.*

number of triangles and the diameter of the graph are reduced dramatically. This seems to be a pre-requisite for the expansion phase. The test has been performed for a $1,000$ node 4-regular ring of cliques.

The convergence rate also depends on the degree of the graph. Since $dn/2$ edges need to be changed one expects at least a linear growth of the convergence speed depending on the degree. In fact the growth seems to be linear as Figure 3.13 indicates. Starting from a d-regular ring of cliques with $10,000$ nodes the number of operations to reach the stable phase is shown.

The graph in Figure 3.14 shows the number of operations necessary to reach the stable phase for the ring of cliques with degree 4, 8, and 12 and growing number of nodes. As a comparison the graphs of the functions $4n \log n$ and $8n \log n$ are added to the diagram. We conjecture that the mixing time of the Markov chain described by the 1-Flipper operation is bounded by $\mathcal{O}(dn \log n)$. According to the measurements in Figure 3.14 this conjecture might be correct since the curve representing the 4-regular ring of cliques is bounded from above by the $4n \log(n)$ curve. However, there are two important things to note. Most importantly it is not clear if the ring of cliques really represents the worst case graph concerning the convergence rate of the 1-Flipper operation. Secondly, we measured the number of operations needed to reach stable expansion only. However, once an expander graph is reached a truly random graph should be reached quickly.

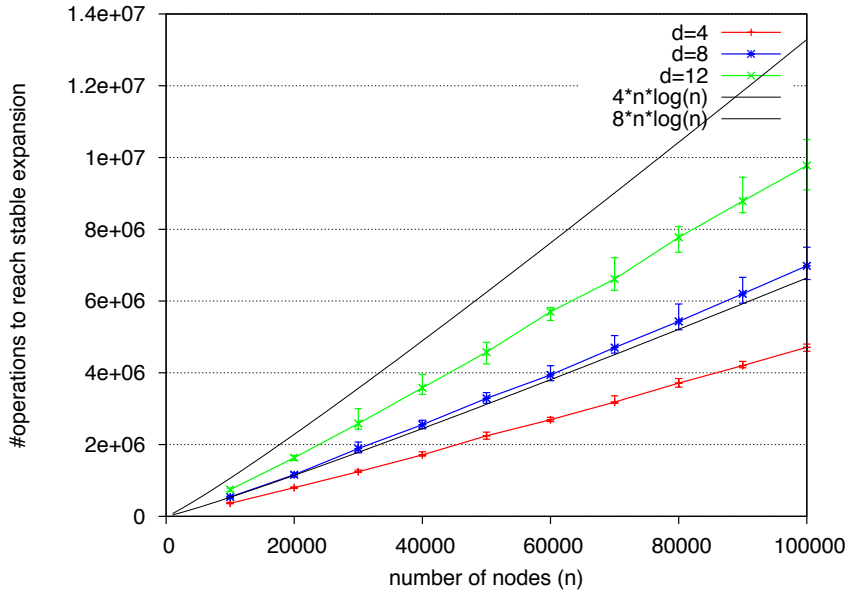


Figure 3.14: *Number of 1-Flipper operations to reach stable expansion for variable degree d and variable number of nodes starting from node ring of cliques.*

3.4.2 Results for the *k*-Flipper

In Section 3.2 we introduced the k-Flipper operation as a generalization of the 1-Flipper operation. In contrast to the 1-Flipper, it is known that the k-Flipper operation needs $\mathcal{O}(dn)$ operations to transform any connected d-regular graph into an expander graph if k, i.e. the length of the hub-path, is chosen large enough and $d \in \Omega(\log n)$. However, in practice k should be chosen to be a small constant. Therefore, we also did simulations to analyze the behavior of the k-Flipper operation with small k. As starting graph we have chosen the ring of cliques graph with $n = 10,000$ and $d = 4$, since this graph turned out to behave worst for the 1-Flipper.

Figure 3.15 shows the evolution of the lower bound of the graph expansion for different values of k. Please note that the curve for $k = 1$ actually represents the 1-Flipper operation (there is a small difference between the 1-Flipper and the k-Flipper when it comes to the success rate of an operation, so that the k-Flipper with $k = 1$ would actually perform slightly worse than the 1-Flipper.). For the k-Flipper we observe the same shape of the curve describing the lower bound of the graph expansion as for the 1-Flipper. As expected, the number of operations to convergence to an expander graph is drastically decreased with increasing k. Figure 3.16 shows the number of operations necessary to reach the stable phase for different values of k.

When comparing the performance of 1-Flipper and k-Flipper one should keep in mind that increasing the hub-path length k also means to increase the number of messages per operation: a 1-Flipper operation requires 4 messages to be exchanged among the participating nodes while a k-Flipper requires $k + 5$ messages ($k + 2$ messages to

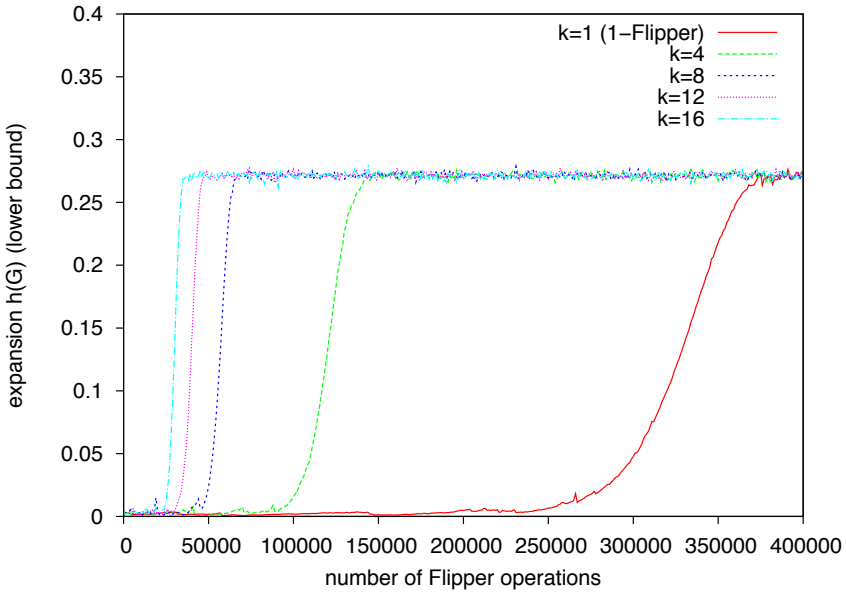


Figure 3.15: *k-Flipper operations applied to a* 10,000 *node* 4*-regular ring of cliques: evolution of the lower bound of the graph expansion for different values of k.*

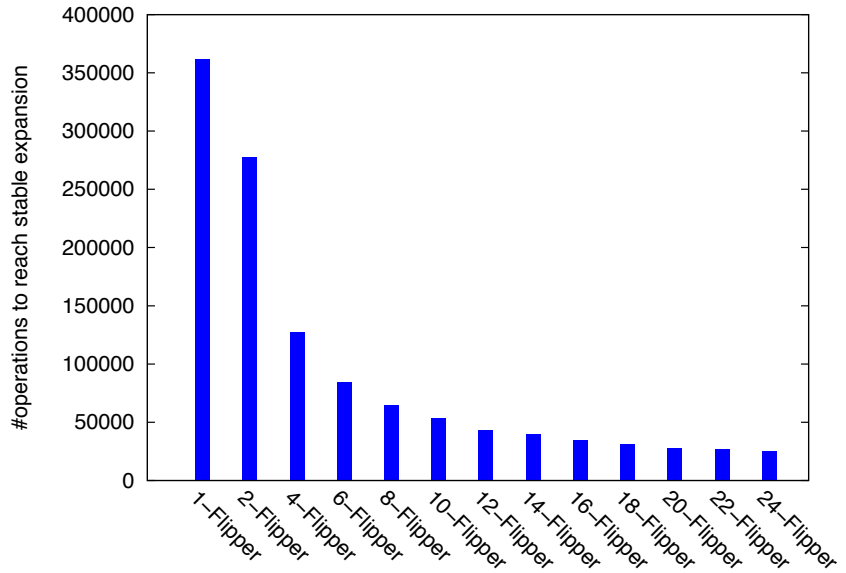


Figure 3.16: *Number of k-Flipper operations to reach stable expansion for* 10,000 *nodes and* 4*-regular ring of cliques.*

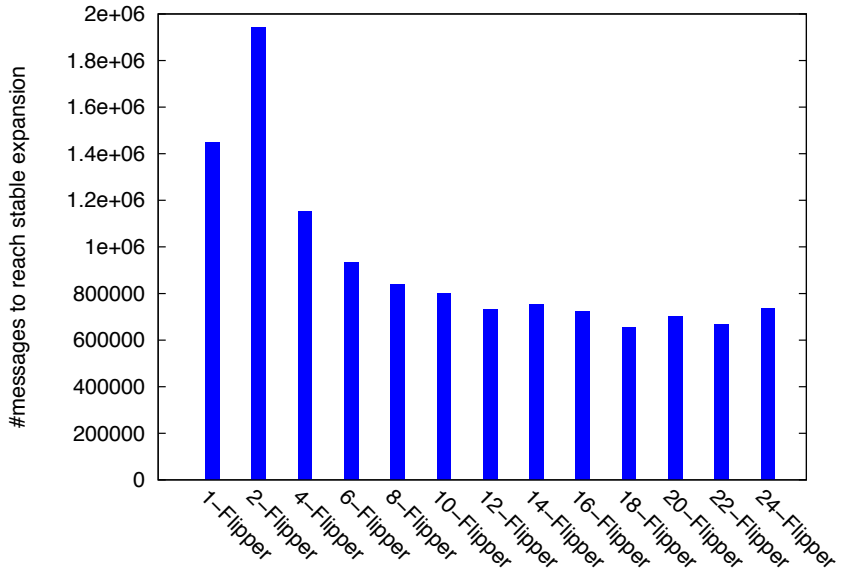


Figure 3.17: *Number of messages to reach stable expansion for* 10,000 *nodes and* 4*-regular ring of cliques.*

perform the random walk and 3 further messages to communicate the truncated random walk and perform the edge flip). The bar chart in Figure 3.17 shows the number of messages needed to reach the phase of stable expansion. Notably the 1-Flipper needs fewer messages than the 2-Flipper. This is because of the slightly higher success probability in case of the 1-Flipper and the longer hub path (which may have to be truncated) in case of the 2-Flipper. For $3 \leq k \leq 24$ the k-Flipper needs fewer messages than the 1-Flipper. Yet, the simulation was done applying Flipper operation sequentially. When concurrent Flipper operations are applied without central coordination the operations may block each other and thus decrease the rate of successful operations. So, for the networking point, a small choice for k is highly desirable.

3.4.3 1-Flipper in a Real World Network

In the masters thesis of Nicolas Heine [51] we evaluated if the 1-Flipper operation can be used to improve real world peer-to-peer networks such as the unstructured Gnutella network. As mentioned in Chapter 2 peers join the Gnutella network by connecting to neighbors of former sessions or by initiating a broadcast of limited depth to find new neighbors to reach their specified degree. Since every peer may freely choose its degree the resulting network is not regular. Therefore, the analysis of the 1-Flipper has to be generalized to non-regular graphs. The generalization is rather straight forward and has been shown by Feder et al. in [35]. The only stumbling block is the proof for generality, i.e. to show that every connected graph with the same degree sequence can be reached by a series of 1-Flipper operations. In a nutshell, generality is provided by

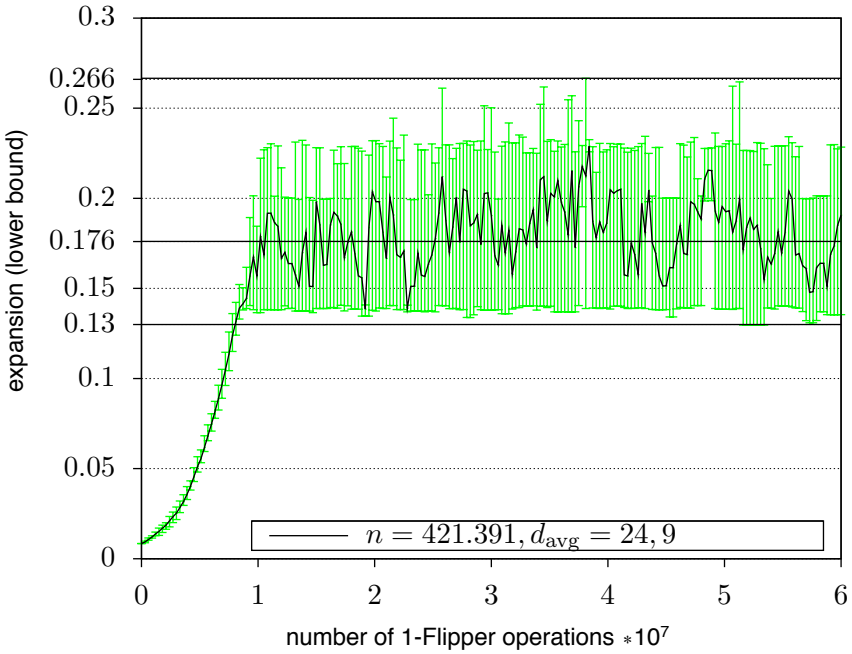


Figure 3.18: *Evolution of expansion when applying* 1*-Flipper operations to a Gnutella snapshot [51].*

the 1-Flipper operation if the degree sequence of the graph implies diameter larger than three for all possible graphs.[4]

We used a snapshot of the actual Gnutella network which was provided by Stutzbach and Rejaie [115] and taken using Cruiser [116] on May 24th, 2006, 00:25am. The graph generated from the snapshot consists of $n = 421,391$ nodes and $|E| = 5,264,433$ edges. For the node degrees

$$d_{min} = 1, \quad d_{avg} = 24.9, \quad \text{and} \quad d_{max} = 672$$

were observed. Figure 3.18 shows the evolution of the expansion's lower bound when applying 1-Flipper operations to the Gnutella snapshot.[5] First of all, we note that there is no initial phase as it was observed for the three starting graphs above. So, the Gnutella snapshot seems to provide minor expansion already. Yet, the 1-Flipper operations are able to drastically increase the expansion. The stable phase, reached after $0.9 * 10^7$ operations, differs from the stable phase observed for regular graphs: the lower bound of the graph expansion is more unsteady and oscillates in the interval $[0.13, 0.266]$. This effect seems to be typical for irregular graphs [51]. The conclusion is

[4] Otherwise an additional graph transformation, the so called bow-tie switch, has to be used to ensure that every graph can be reached.

[5] Concerning the scale of the y-axis note that a different definition of expansion defined over the volume of subsets $S \subset V$, i.e. the sum of the degrees of all $v \in S$, instead of the number of nodes in S has been used (see [24]).

that an incorporation of the 1-Flipper operation into the Gnutella protocol would drastically improve the network structure of Gnutella, resulting in improved robustness and lookup performance.

4

The Directed Case: Pointer-Push&Pull

For the use in peer-to-peer networks, the 1-Flipper introduced in the previous chapter can be improved. First, the 1-Flipper involves the active participation of four peers. Second, the maintenance of undirected, or to be more precise bi-directed, graphs is costly and not necessary at all costs. In practice the use of directed graphs (digraphs) is often completely sufficient since peers do not necessarily need to know who is pointing towards them as long as the network is connected and robust. Note, that there are several peer-to-peer networks which use digraphs, see [114, 60, 80, 52] for some prominent examples.

In this chapter we concentrate on random graph transformations for weakly connected multi-digraphs and present the Pointer-Push&Pull operation. Its main advantage is simplicity: For a local update operation only two peers need to exchange two messages. Using Markov theory, we show that Pointer-Push&Pull operations will eventually generate all weakly connected out-regular multi-digraphs with the same probability. Interestingly this is not the case if Pointer-Push&Pull is applied to out-regular simple digraphs. Then we show that a slight change of the probability distribution for the choice of edges gives another terminal distribution over multi-digraphs with simple digraphs occurring with higher probability. At last we discuss how to implement Pointer-Push&Pull in a distributed network and solve the problems arising by concurrent operations.

4.1 The Pointer-Push&Pull Graph Transformation

As in the previous chapter, our goal is to design a graph transformation for connected digraphs which is sound, general, feasible, and converges quickly (cf. Chapter 2). Since the model of multi-digraphs is less restrictive than the model of simple digraphs, we will first consider multi-digraphs and then show that our results can not be transferred to simple digraphs.

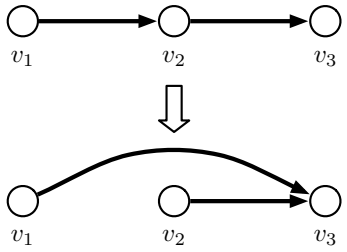


Figure 4.1: *The Pointer-Push operation.*

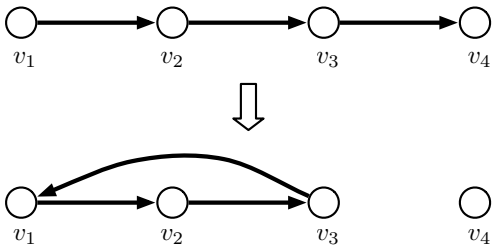


Figure 4.2: *The Pointer-Pull operation.*

4.1.1 Multi-Digraphs

We start with some fundamental considerations about transformations of multi-digraphs. Considering a node v_1 of a multi-digraph $G = (V, E)$, there are basically two possibilities to change the set of edges E:

Pointer-Push Change the outgoing edges of the node, i.e. do a random walk v_1, v_2, v_3 in G and replace (v_1, v_2) with (v_1, v_3) (see Figure 4.1).

Pointer-Pull Change the ingoing edges of the node, i.e. do a random walk v_1, v_2, v_3, v_4 in G and replace (v_3, v_4) with (v_3, v_1) (see Figure 4.2).

However, neither the Pointer-Push nor the Pointer-Pull operation meet our requirements for random graph transformations. For the Pointer-Push operation it turns out that it is sound and feasible but not general, i.e. the digraph will converge to a set of connected stars in the limit. This is because there is a non zero probability to generate a sink in the digraph, i.e. a node with all edges pointing to itself. Once a node points to such a sink there is no possibility to remove this edge with a Pointer-Push operation anymore.

The situation of the Pointer-Pull operation is similar. The Pointer-Pull operation is not able to preserve connectivity in a digraph and therefore is not sound. To see this, note that there is a non-zero probability for a node to create self-loops. Once all edges of a node are pointing to itself this node is disconnected from the rest of the digraph. Without global knowledge such a situation is irrevocable and therefore the digraph

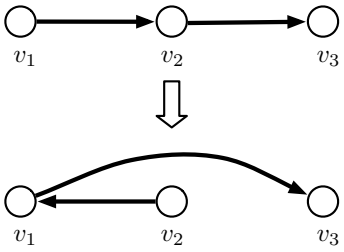


Figure 4.3: *The Pointer-Push&Pull operation.*

will consist only of disconnected nodes in the long run. A more detailed analysis of the Pointer-Push and the Pointer-Pull operation is presented in Appendix A.

We now show that a combination of these basic graph transformations, called Pointer-Push&Pull operation, overcomes the shortcomings of the Pointer-Push respectively Pointer-Pull operation. It is defined as follows.

Definition 4.1 (Pointer-Push&Pull) *Let $G = (V, E, \#_E)$ be a d-out-regular multi-digraph and let nodes $v_1, v_2, v_3 \in V$ form a directed path $P = (v_1, v_2, v_3)$ in G. Then, the Pointer-Push&Pull operation $\mathcal{PP}_P$ transforms graph G to graph $\mathcal{PP}_P(G) = (V, E', \#_{E'})$ with*

$$E' = (E \setminus \{(v_1, v_2), (v_2, v_3)\}) \cup \{(v_1, v_3), (v_2, v_1)\} .$$

The Pointer-Push&Pull operation is illustrated in Figure 4.3. Note, that a Pointer-Push&Pull operation can be divided to a Pointer-Push operation (between nodes v_1 and v_3) and a Pointer-Pull operation (between nodes v_1 and v_2). We start our analysis of this graph transformation with the following lemma.

Lemma 4.1 *The Pointer-Push&Pull operation is sound for the domain of weakly connected out-regular multi-digraphs.*

Proof: We have to show that the Pointer-Push&Pull operation preserves connectivity and outdegree of all nodes. For connectivity note that all participating nodes stay connected, what implies at least weak connectivity for the graph. Concerning the outdegree node v_1 as well as v_2 just replace one of their outgoing edges and therefore their outdegrees remain unchanged. ■

Algorithm 4.1 shows a randomized variant of the Pointer-Push&Pull operation. The random Pointer-Push&Pull operation chooses its starting node uniformly at random and then performs a random walk of length two with some probability. Recall that due to multi-edges $|N^+(v)|$ may be less than d in d-out-regular multi-digraphs. Therefore, a random Pointer-Push&Pull operation may cancel with a probability proportional to the number of multi-edges of v_1 and v_2. This specific definition of the random Pointer-Push&Pull operation is motivated by the following lemma.

Algorithm 4.1 Random Pointer-Push&Pull

1: $v_1 \leftarrow$ random node $\in V$
2: **if** random event with $p = \frac{|N^+(v_1)|}{d}$ occurs **then**
3: $\quad v_2 \leftarrow$ random node $\in N^+(v_1)$
4: $\quad$ **if** random event with $p = \frac{|N^+(v_2)|}{d}$ occurs **then**
5: $\quad\quad v_3 \leftarrow$ random node $\in N^+(v_2)$
6: $\quad\quad E \leftarrow (E \setminus \{(v_1, v_2), (v_2, v_3)\}) \cup \{(v_1, v_3), (v_2, v_1)\}$
7: $\quad$ **end if**
8: **end if**

Lemma 4.2 *The random Pointer-Push&Pull operation is symmetric for out-regular multi-digraphs. Thus, for two out-regular multi-digraphs G, G' the probability to transform G to G' by a random Pointer-Push&Pull operation is the same as to transform G' to G by a random Pointer-Push&Pull operation, i.e.*

$$\Pr\left[G \overset{\mathcal{PP}}{\to} G'\right] = \Pr\left[G' \overset{\mathcal{PP}}{\to} G\right],$$

where $G \overset{\mathcal{PP}}{\to} G'$ denotes that G can be transformed to G' with a single Pointer-Push&Pull operation.

Proof: Let $P = (u, v, w)$ be the path of the Pointer-Push&Pull operation transforming G to G'. To transform G' back to G we need to apply a Pointer-Push&Pull operation $\mathcal{PP}_{P'}$ with $P' = (v, u, w)$. Note, that $\mathcal{PP}_{P'}$ is the only possibility to transform G' back to G with a single operation. As noted above, the random Pointer-Push&Pull operation chooses its starting node uniformly at random and then chooses a neighboring node for two times with probability $p = 1/d$ each. This implies

$$\Pr\left[G \overset{\mathcal{PP}}{\to} G'\right] = \Pr\left[G' \overset{\mathcal{PP}}{\to} G\right] = \frac{1}{nd^2}.$$

■

The following lemma names the set of graphs which can be reached by applying random Pointer-Push&Pull operations to an arbitrary starting graph repeatedly.

Lemma 4.3 *A series of random Pointer-Push&Pull operations can transform a graph to any other graph within the domain of out-regular weakly connected multi-digraphs.*

Proof: We show that any weakly connected d-out-regular multi-digraph $G = (V, E, \#_E)$ with $V = \{v_1, \ldots, v_n\}$ can be transformed to a canonical graph $G_C = (V, E_C, \#_{E_C})$ with

$$E_C = \{(v_1, v_1), (v_2, v_1), \ldots, (v_n, v_1)\},$$

and $\forall e \in E_C : \#_{E_C}(e) = d$, i.e. all edges are pointing to v_1. Then, the theorem follows since each Pointer-Push&Pull operation is reversible so that two arbitrary graphs can be transformed to each other using G_C as an intermediate state.

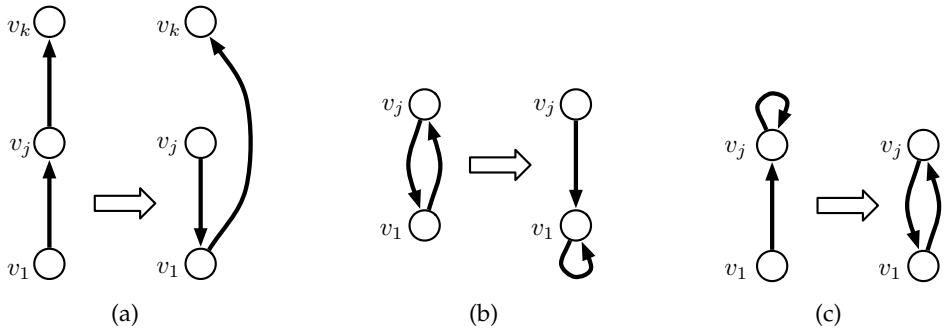


Figure 4.4: *Cases 1.1 (a), 1.2 (b), and 1.3 (c) of Lemma 4.3 (from left to right).*

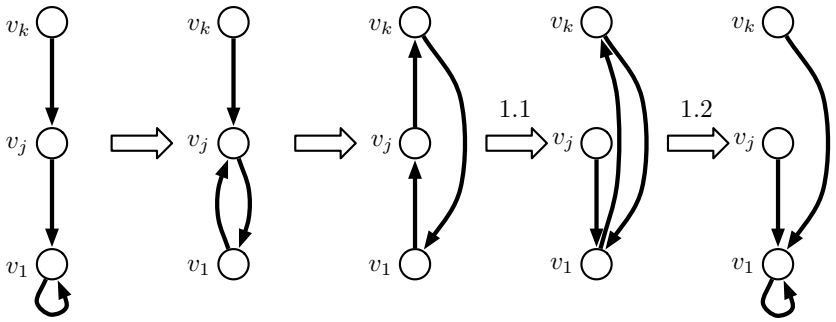


Figure 4.5: *Case 2.1 of Lemma 4.3.*

We start with an arbitrary weakly connected d-out-regular multi-digraph G and increase the indegree of v_1 successively. This can be done by repeatedly applying the at each time first applicable of the following six transformations:

Case 1: v_1 has at least one edge (v_1, v_j) with $j \neq 1$

Case 1.1: v_j has an edge (v_j, v_k) with $k \neq 1$ and $k \neq j$. Apply a Pointer-Push&Pull operation to the path $P = (v_1, v_j, v_k)$ (see Figure 4.4(a)).

Case 1.2: v_j has an edge (v_j, v_1). Apply a Pointer-Push&Pull operation with $P = (v_1, v_j, v_1)$ (see Figure 4.4(b)).

Case 1.3: v_j has an edge (v_j, v_j). Apply a Pointer-Push&Pull operation with $P = (v_1, v_j, v_j)$ (see Figure 4.4(c)).

Case 2: v_1 has no edge (v_1, v_j) with $j \neq 1$ and therefore has an edge (v_1, v_1)

Case 2.1: There is a node v_j pointing to v_1 and a node v_k, with $k \neq j$, pointing to v_j. Apply Pointer-Push&Pull operations to the paths $P_1 = (v_j, v_1, v_1)$, $P_2 = (v_k, v_j, v_1)$, $P_3 = (v_1, v_j, v_k)$, and $P_4 = (v_1, v_k, v_1)$ (see Figure 4.5).

Case 2.2: There is a node v_j pointing both to v_1 and v_k, $k \neq 1$. Furthermore, v_k points to a node v_l with $l \neq 1$. If we apply a Pointer-Push&Pull operation

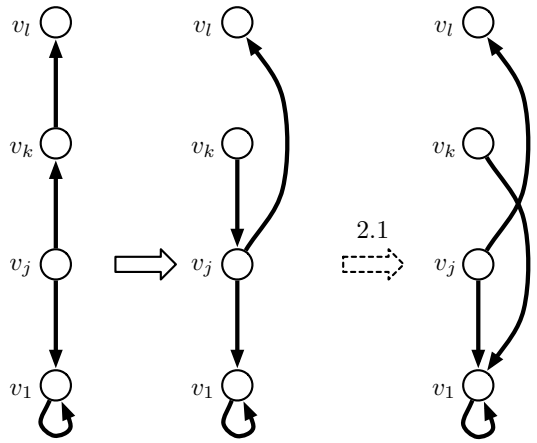


Figure 4.6: *Case 2.2 of Lemma 4.3.*

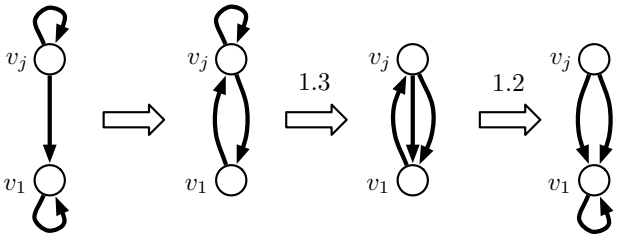


Figure 4.7: *Case 2.3 of Lemma 4.3.*

to the path $P_1 = (v_j, v_k, v_l)$, then we reach the start configuration of case 2.1 and can continue with the operations described there (see Figure 4.6).

Case 2.3: There is a node v_j pointing to v_1 and itself. Apply Pointer-Push&Pull operations to the paths $P_1 = (v_j, v_1, v_1)$, $P_2 = (v_1, v_j, v_j)$, and $P_3 = (v_1, v_j, v_1)$ (see Figure 4.7).

Following this scheme, the starting graph G will be transformed to G_C in the limit. To see this, note that G_C is the only possible graph with indegree dn for node v_1. Furthermore, the six cases cover all possible arrangements of edges, so that always one of the cases can be applied unless G_C is reached already. ■

As a side effect, the proof of Lemma 4.3 delivers an upper bound for the number of Pointer-Push&Pull operations necessary to transform two graphs to each other: Starting with an arbitrary d-out-regular weakly connected multi-digraph G, at most $10nd$ Pointer-Push&Pull operations are needed to reach any other d-out-regular weakly connected multi-digraph G'.

Combining our results, we are now able to show that the Pointer-Push&Pull operation provides uniform generality within the domain of d-out-regular weakly connected multi-digraphs.

Theorem 4.1 *Let G' be an arbitrary d-out-regular weakly connected multi-digraph with n nodes. Then, applying random Pointer-Push&Pull operations repeatedly to G' will construct every d-out-regular weakly connected multi-digraph with the same probability in the limit, i.e.*

$$\lim_{t\to\infty} \Pr\left[G' \xrightarrow{t} G\right] = \frac{1}{|\mathcal{MDG}_{n,d}|},$$

where $\mathcal{MDG}_{n,d}$ denotes the set of all d-out-regular weakly connected multi-digraphs with n nodes.

Proof: Consider the Markov chain over the set of d-out-regular weakly connected multi-digraphs described by the random Pointer-Push&Pull operation. Lemma 4.2 implies that the transition matrix of the Markov chain is symmetric and therefore doubly stochastic. Lemma 4.3 shows that every state of the Markov chain is reachable. Furthermore, there is a non-zero probability to transform a graph to itself (this happens when the first edge chosen during the random walk is a self-loop). This implies that some diagonal entries of the transition matrix are non-zero. Using these three properties of the transition matrix the theorem follows by applying essential results of Markov theory. ■

4.1.2 Edge Labeled Multi-Digraphs

We have seen that the Pointer-Push&Pull operation generates out-regular weakly connected multi-digraphs with uniform probability. We now show that a slight change of the probability distribution for the choice of edges gives another terminal distribution over multi-digraphs with simple digraphs occurring with higher probability. To achieve this, we modify the Pointer-Push&Pull operation to work with out-regular edge labeled multi-digraphs, which we formally define as follows.

Definition 4.2 (Edge Labeled Multi-Digraph) *An edge labeled d-out-regular multi-digraph $G^* = (V, E^*)$ is defined by a node set $V = \{1, \ldots, n\}$ and an edge set $E^* = \{(u, v, i) : u, v \in V, i \in \{1, \ldots, d\}\}$, where i specifies the label of an edge. Here, we restrict to the labels $1, \ldots, d$ and unique labels for each outgoing edge of a node, i.e. $\forall u \in V, \forall i \in \{1, \ldots, d\} : \exists v \in V : (u, v, i) \in E^*$.*

For the domain of d-out-regular edge labeled multi-digraphs we use the notation $N^+(v, i)$ with $i \in \{1, \ldots, d\}$ and $v \in V$ to refer to v's neighbor due to the i-th labeled edge. In addition we use $E^-(v) = \{(w, v, i) \in E^*\}$ to refer to the set of v's ingoing edges. Recall that $N^+(v, i) = N^+(v, j)$, $i \neq j$ is possible in a multi-digraph. Algorithm 4.2 shows the random Pointer-Push&Pull operation, modified for edge labeled multi-digraphs. For the sake of clarity and to simplify notation, we will refer to this algorithm as *labeled*-Pointer-Push&Pull and use *unlabeled*-Pointer-Push&Pull to refer to the previous section Pointer-Push&Pull algorithm.

The proofs for preserving connectivity and out-degrees of the unlabeled-Pointer-Push&Pull can be transferred directly to the labeled-Pointer-Push&Pull operation. We now show that the labeled-Pointer-Push&Pull operation is symmetric within the domain of out-regular weakly connected edge labeled multi-digraphs.

Algorithm 4.2 Random labeled-Pointer-Push&Pull

1: $v_1 \leftarrow$ random node $\in V$
2: $i \leftarrow$ random number $\in \{1, ..., d\}$
3: $v_2 \leftarrow N^+(v_1, i)$
4: $j \leftarrow$ random number $\in \{1, ..., d\}$
5: $v_3 \leftarrow N^+(v_2, j)$
6: $E^* \leftarrow (E^* \setminus \{(v_1, v_2, i), (v_2, v_3, j)\}) \cup \{(v_1, v_3, i), (v_2, v_1, j)\}$

Lemma 4.4 *The labeled-Pointer-Push&Pull operation $\mathcal{PP}^*$ is symmetric within the domain of out-regular weakly connected edge labeled multi-digraphs. That is, for two arbitrary graphs G_1^*, G_2^* of this domain*

$$\Pr\left[G_1^* \overset{\mathcal{PP}^*}{\to} G_2^*\right] = \Pr\left[G_2^* \overset{\mathcal{PP}^*}{\to} G_1^*\right].$$

Proof: According to Algorithm 4.2 G_1^* and G_2^* differ in exactly two edges. More precisely G_1^* has edges (v_1, v_2, i), (v_2, v_3, j) and G_2^* has edges $(v_1, v_3, i), (v_2, v_1, j)$. The only way to transform G_2^* back to G_1^* with a single operation is a labeled-Pointer-Push&Pull operation using the edges (v_2, v_1, j) and (v_1, v_3, i). Now observe that the starting node of the labeled-Pointer-Push&Pull operation is chosen uniformly at random, i.e. with $p = 1/n$. So, the algorithm does a random walk by choosing two edges uniformly at random, implying $\Pr[G_1^* \overset{\mathcal{PP}^*}{\to} G_2^*] = \Pr[G_2^* \overset{\mathcal{PP}^*}{\to} G_1^*] = 1/(nd^2)$. ∎

Similar as in case of the unlabeled-Pointer-Push&Pull operation, every out-regular weakly connected edge labeled multi-digraph can be reached by a series of labeled-Pointer-Push&Pull operations. This can be shown by transferring the proof of Lemma 4.3 to edge labeled multi-digraphs. To see this, note that using the proof of Lemma 4.3 any starting graph will be transformed to the canonical graph G_C — regardless of its labeling. Furthermore, the labeling in G_C can be neglected, since all edges point to the same node. Therefore, the proof also holds for the labeled-Pointer-Push&Pull operation.

Finally, these results lead to the following theorem showing that the labeled-Pointer-Push&Pull operation provides uniform generality within the domain of out-regular weakly connected edge labeled multi-digraph.

Theorem 4.2 *Let G_0^* be an d-out-regular weakly connected edge labeled multi-digraph with n nodes. Then, applying random labeled-Pointer-Push&Pull operations repeatedly to the graph will construct every d-out-regular weakly connected edge labeled multi-digraph with the same probability in the limit, i.e.*

$$\lim_{t\to\infty} \Pr\left[G_0^* \overset{t}{\to} G^*\right] = \frac{1}{|\mathcal{MDG}_{n,d}^*|},$$

where $\mathcal{MDG}_{n,d}^*$ denotes the set of all d-out-regular weakly connected edge labeled multi-digraphs.

Proof: The proof is essentially the same as the proof of Theorem 4.1. The transition matrix of the Markov chain described by the labeled-Pointer-Push&Pull operation over all $G^* \in \mathcal{MDG}^*_{n,d}$ is doubly stochastic and every state is reachable. Furthermore the labeled-Pointer-Push&Pull operation does not change the edge set if the operation chooses a self-loop for two times, implying non-zero diagonal entries. Therefore, the theorem follows by essential results of Markov theory. ■

At first glance this result does not seem to be more powerful than the uniform generation of d-out-regular weakly connected multi-digraphs, shown by Theorem 4.1. Note however, that each d-out-regular edge labeled multi-digraph can be transformed to a d-out-regular (unlabeled) multi-digraph G. For this, the set of triples in E^* is transformed to a multi-set by omitting the labels, i.e. the third element of each triple. This leads to the following definition of an equivalence class.

Definition 4.3 (Equivalence Class) *The set of d-out-regular edge labeled multi-digraphs describing a d-out-regular (unlabeled) multi-digraph G, when omitting the edge labels, is called equivalence class of G:* $[G]$.

Now, we are able to transfer Theorem 4.2 to the domain of unlabeled multi-digraphs using equivalence classes.

Theorem 4.3 *Let G be a d-out-regular weakly connected multi-digraph with n nodes. Then, applying random labeled-Pointer-Push&Pull operations to the graph repeatedly will construct every d-out-regular weakly connected multi-digraph G with probability* $\sim |[G]|$.

It remains to analyze the size of the equivalence classes, i.e. how many ways are there to label the edges of a d-out-regular multi-digraph with labels chosen according to Definition 4.2. Straight forward combinatorics leads to the following sizes for equivalence classes in dependency of the number of multi-edges in a graph.

Lemma 4.5 *Let G be a d-out-regular multi-digraph. Then, the size of the equivalence class of G is given by*

$$|[G]| = \prod_{u \in V(G)} \frac{d!}{M_0(u)! \prod_{i=1}^{d} (i!)^{M_i(u)}} ,$$

with $M_0(u)$ *denoting the number of slopes,* $M_1(u)$ *denoting the number of single edges,* $M_2(u)$ *denoting the number of double edges, etc.*

In other words, Lemma 4.5 shows: The lower the number of multi-edges in G is, the larger is the cardinality of its equivalence class $[G]$. This again implies that the labeled-Pointer-Push&Pull operation will generate a particular simple digraph with higher probability than a particular multi-digraph. Therefore, and because there is no additional overhead compared to the unlabeled-Pointer-Push&Pull operation, the labeled-Pointer-Push&Pull operation will be preferable in the majority of cases.

By definition the Pointer-Push&Pull operations guarantee out-regularity only. In many applications it is desirable to have regular digraphs. Therefore we now analyze

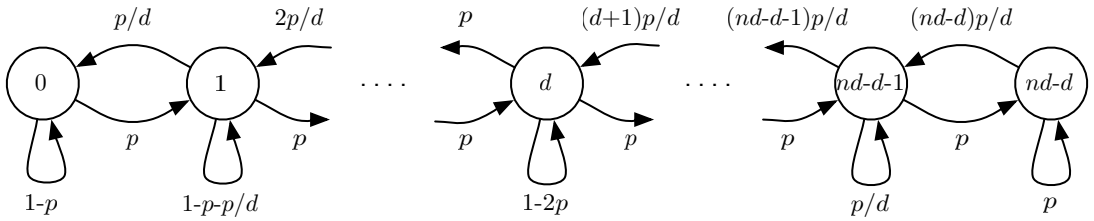


Figure 4.8: *A Markov chain modeling the change of a node's indegree.*

the indegree distribution of the evolving graphs. On the one hand a node v increases its indegree by one if chosen as starting node.[1] This occurs with probability $p = 1/n$. On the other hand v's indegree is decreased only if v is chosen as second node of a Pointer-Push&Pull operation. This happens with probability proportional to v's indegree, i.e. $|E^-(v)|p/d$. Knowing this, the change of v's indegree can be modeled by a Markov chain with states $0, \ldots, nd - d$ representing the current indegree and transition probabilities as shown in Figure 4.8. Analyzing the stationary probability distribution of this Markov chain leads to an almost Poisson distribution with expectation d.

Lemma 4.6 *Starting with an arbitrary d-out-regular weakly connected edge labeled multi-digraph with n nodes and applying labeled-Pointer-Push&Pull operations repeatedly the indegrees of the nodes will be almost Poisson distributed, i.e. a node will have indegree k with probability*

$$P_k = \frac{d^k}{k!} \left(\sum_{i=0}^{nd-d} \frac{d^i}{i!} \right)^{-1}.$$

Proof: Consider the stationary distribution when the Markov chain shown in Figure 4.8 has converged. Let P_k, $0 \leq k \leq nd - d$, be the probability for state k. For state $1 \leq k \leq nd - d - 1$ we observe the recurrence

$$P_k = pP_{k-1} + \left(1 - p - k\frac{p}{d}\right) P_k + \left((k+1)\frac{p}{d}\right) P_{k+1}.$$

Furthermore we observe

$$P_0 = (1-p)P_0 + \frac{p}{d}P_1 \tag{4.1}$$

for state $k = 0$. Rearranging Equation 4.1 we have $\frac{P_1}{P_0} = d$ and using induction it can be shown that

$$\frac{P_k}{P_{k-1}} = \frac{d}{k}$$

[1] Actually v's indegree will not increase if the first edge chosen is a self-loop. However the expected overall number of self-loops in a d-out-regular multi-digraph is constant, or to be more specific d. So, we neglect the constant number of affected nodes in our analysis and assume that the node has no self-loops during the graph transformations.

holds for $k \geq 1$. Thus we have $P_k = \frac{d}{k} P_{k-1}$ and rewriting this equation we get

$$P_k = \frac{d}{k} \cdot \frac{d}{k-1} \cdots \frac{d}{1} \cdot P_0. \tag{4.2}$$

Having Equation 4.2 it remains to find an adequate term for P_o. Solving $\sum_{k=0}^{nd-d} P_k = 1$ for P_0 we have

$$P_0 = \left(\sum_{k=0}^{nd-d} \frac{d^k}{k!} \right)^{-1}. \tag{4.3}$$

Combining Equation 4.2 and Equation 4.3 we receive

$$P_k = \frac{d^k}{k!} \left(\sum_{i=0}^{nd-d} \frac{d^i}{i!} \right)^{-1}.$$

This concludes the proof of Lemma 4.6. ∎

4.1.3 Simple Digraphs

We have seen that the labeled-Pointer-Push&Pull operation generates simple digraphs with higher probability than the unlabeled-Pointer-Push&Pull operation. Though, the following theorem shows that simple digraphs are still outnumbered.

Theorem 4.4 *The fraction p^* of d-out-regular edge labeled simple digraphs with n nodes in all d-out-regular edge labeled multi-digraphs with n nodes is bounded by*

$$e^{-\frac{d^2}{1-\frac{d}{n}}} \leq p^* \leq e^{-d}.$$

Proof: In a d-out-regular edge labeled simple digraph neither self-loops nor multi-edges occur. This implies that there are $(n-1)!/(n-d-1)!$ possibilities for a nodes neighborhood while there are n^d possibilities if self-loops and multi-edges are allowed. This gives an upper bound of

$$\begin{aligned} \left(\frac{(n-1)!}{(n-d-1)! n^d} \right)^n &\leq \frac{(n-1)^{dn}}{n^{dn}} \\ &= \left(1 - \frac{1}{n} \right)^{dn} \\ &\leq e^{-d}. \end{aligned}$$

For the lower bound of d-out-regular edge-labeled simple multi-digraphs, observe that $\frac{(n-1)!}{(n-d-1)!} \geq (n-d)^d$. This implies

$$\begin{aligned}\left(\frac{(n-1)!}{(n-d-1)!n^d}\right)^n &\geq \frac{(n-d)^{dn}}{n^{dn}} \\ &= \left(1-\frac{d}{n}\right)^{dn} \\ &= \left(1-\frac{d}{n}\right)^{(\frac{n-d}{d})\frac{d^2n}{n-d}} \\ &\geq e^{-\frac{d^2n}{n-d}},\end{aligned}$$

and thus proves the theorem. ■

Given a node, the probability of being the source of only simple edges (not multiple edges or self-loops) is quite high, i.e. at least $1 - \frac{d}{n-d-1}$ (which suffices practical needs). Whereas, if we consider the complete graph, the fraction of simple digraphs created by the labeled-Pointer-Push&Pull operation is rather small, i.e. decreasing exponentially with the degree. Since multiple edges are an unnecessary waste of resources one might want to generate simple digraphs only. A straight forward solution is to modify the labeled- or unlabeled-Pointer-Push&Pull operation such that it is only applied if the resulting digraph is simple. We call this modified graph transformation *simple*-Pointer-Push&Pull. Unfortunately, the simple-Pointer-Push&Pull operation is not general for the domain of simple digraphs. To see this consider any symmetric digraph, i.e. a digraph where for each edge (u, v) also the edge (v, u) is in the edge set. In such digraphs no simple-Pointer-Push&Pull operation can be applied, since each operation would either create a multi-edge or a self-loop and therefore leave the domain of simple digraphs. The only way to reach all simple digraphs using Pointer-Push&Pull operations is to allow the multiple occurrence of edges, i.e. the use of multi-digraphs.

4.2 Pointer-Push&Pull in Peer-to-Peer Networks

The labeled- and unlabeled-Pointer-Push&Pull operations are in particular useful for the maintenance of distributed random networks, as they are used in unstructured peer-to-peer networks for example. Applying Pointer-Push&Pull operations repeatedly ensures that the network stays truly random, even if peers join, leave, or fail in non-random fashion. In this section we discuss how to implement Pointer-Push&Pull operations in a distributed network without central coordination.

A Pointer-Push&Pull operation on the path $P = (v_1, v_2, v_3)$ consists of the following three sequential steps:

Step 1 v_1 requests a random neighbor from v_2

Step 2 v_2 replaces v_3 by v_1 in its neighborhood list and sends the ID of v_3 to v_1

Step 3 v_1 receives the ID of v_3 from v_2 and replaces v_2 by v_3 in its neighborhood list

These three steps involve only two messages between v_1 and v_2, carrying the information of one edge respectively. This shows, that a Pointer-Push&Pull operation does not introduce additional overhead compared to so called *ping* messages for the periodical verification of the neighborhood, which is mandatory in dynamic networks. Since Pointer-Push&Pull operations are initiated randomly by every peer they may be used to replace periodical verification messages and allow to maintain the network structure at the same time.

4.2.1 Concurrency

In the previous section we have shown that a single Pointer-Push&Pull operation never disconnects a network. However, things are different when there are concurrent Pointer-Push&Pull operations with intersecting paths. They bear the risk of disconnecting a network. To see this, consider a directed path (s, t, u, v) in the network and two concurrent Pointer-Push&Pull operations $\mathcal{PP}_P$ and $\mathcal{PP}_{P'}$ with $P = (s, t, u)$ and $P' = (t, u, v)$. Now consider the following situation. $\mathcal{PP}_{P'}$ has removed edge (u, v) and created edge (u, t). At this point of time $\mathcal{PP}_P$ starts and finishes before $\mathcal{PP}_{P'}$ continues, i.e. $\mathcal{PP}_P$ creates edges (s, u), (t, s) and removes edges (s, t), (t, u). Then, $\mathcal{PP}_{P'}$ can not be finished since $\mathcal{PP}_{P'}$ will try to remove edge (t, u) which no longer exists. Even worse, the resulting network is possibly disconnected since there is no path between u and v.

Fortunately, there is a simple solution to this problem. To prevent the interference of Pointer-Push&Pull operations the first edge of each operation is locked for the use by further operations. This implies that this edge can not be used by multiple Pointer-Push&Pull operations at the same time. The second edge of a Pointer-Push&Pull operation does not need to be locked, since it will be replaced immediately without any delay due to network communication.

If a Pointer-Push&Pull operation choses an edge, which is currently locked, then there are two possible solutions. First, the Pointer-Push&Pull operation can of course be canceled. This is unproblematic since no changes have been made to the network graph, yet. Another option is to wait for the lock to disappear. This is reasonable since an operation will usually need few milliseconds. Furthermore, the number of intersecting operations will be low if the interval in which a node starts Pointer-Push&Pull operations is chosen reasonably.

4.2.2 Joining Peers

In this section we describe several operations for a peer to join a Pointer-Push&Pull maintained network. Here, we distinguish between *local* and *global* join operations. Local join operations do only rely on the local information each peer has of the network and therefore can be easily implemented in the dynamic setting of a peer-to-peer network. Global join operations require the whole topology of the network to be known.

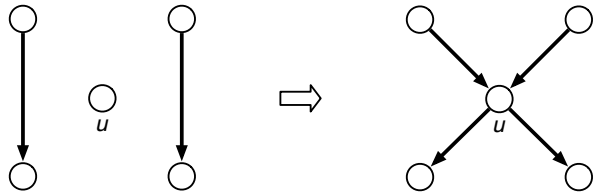


Figure 4.9: *The directed-pegging operation when $d = 2$.*

Thus global join algorithms cannot be implemented one-to-one in a peer-to-peer network but it is conceivable that local variations giving an approximate solution can be implemented. Furthermore, we will prove that the evolving graphs of the global join algorithm described below are expander graphs.

Local Join Operations

Since Pointer-Push&Pull operations allow to recover from any degenerate network structure and the domain of connected out-regular multi-digraphs is very unrestrictive joining peers may make use of minimalistic join operations. So, it would be sufficient if a peer u willing to join the network via a previously known peer v creates d connections to v or even worse: creates one connection to v and $d-1$ self-loops. Of course, more elaborate join algorithms will speed up the convergence process to a truly random topology. So, depending on the network dynamics it may be reasonable to join a network by performing a short random walk in the network and make u's outgoing edges point to some of the nodes encountered during the random walk.

A Global Join Operation Generating Expander Digraphs

In Chapter 3 we described the pegging operation to join d-regular undirected graphs respectively networks and have seen that the graphs created with this operation are d-connected a.a.s. for example. Algorithm 4.3 describes a variation of the pegging operation for the domain of d-regular digraphs to which we will refer to as *directed-pegging* operation from now on.

Algorithm 4.3 directed-pegging (node u)

1: $V \leftarrow V \cup u$
2: **repeat**
3: choose edge $e = (v, w) \in E$ with $v \notin N^-(u)$ and $w \notin N^+(u)$ u.a.r.
4: $E \leftarrow (E \setminus (v, w)) \cup \{(v, u), (u, w)\}$
5: **until** $|N^+(u)| = d$

The directed-pegging operation is illustrated in Figure 4.9. As the pegging operation for undirected graphs, the directed-pegging operation chooses edges of the digraph uniformly at random and puts a joining node (or peer) "in the middle" of these edges.

The smallest digraph to which directed-pegging operations can be applied successfully is a clique of $d + 1$ nodes. We will neglect smaller graphs here and assume that there is a bootstrapping process to construct such a clique.

Before we analyze the digraphs evolving by the use of the directed-pegging operation we define some additional notations. For a digraph $G = (V, E)$ and a subset $S \subset V$ we denote the set of all directed edges leaving S as $\delta^+ S = \{(u, v) \in E | u \in S, v \notin S\}$ and the set of all directed edges entering S as $\delta^- S = \{(u, v) \in E | u \notin S, v \in S\}$. The following lemma states some fundamental properties of the digraphs evolving by the use of the directed-pegging operation.

Lemma 4.7 *A digraph $G = (V, E)$ generated using the directed-pegging operation has the following properties:*

1. *G is regular, i.e. out-regular and in-regular.*

2. *For every subset $S \subset V$ it holds that $|\delta^+ S| = |\delta^- S|$.*

Proof: Note that the new node u creates d outgoing edges and receives d ingoing edges. Furthermore, the out- and indegrees of existing nodes remain unchanged since an edge (v, w) chosen by the directed-pegging operation is replaced by a directed path (v, u, w), i.e. the outgoing edge of v respectively the ingoing edge of w are replaced by new edges.

For the proof of property 2 we consider an arbitrary subset $S \subset V$ and examine how $\delta^+ S$ and $\delta^- S$ change when a new node u arrives and the set of edges is changed. We assume $u \notin S$ and the edge chosen by the insertion scheme is (v, w). First of all note that the edge boundary of S, i.e. $\delta^+ S$ and $\delta^- S$, will not change if $v, w \notin S$. So, the edge boundary will change in the following cases only:

- $v \in S$ and $w \in S$: Since $(v, w) \notin \delta^+ S$ and $(v, w) \notin \delta^- S$ no edges will be removed from $\delta^+ S$ respectively $\delta^- S$. But (v, u) will be added to $\delta^+ S$ and (u, w) will be added to $\delta^- S$ and thus $|\delta^+ S|$ and $|\delta^- S|$ grow by one respectively.

- $v \in S$ and $w \notin S$: (v, w) will be removed from $\delta^+ S$ and (v, u) will be added to $\delta^+ S$. No changes are made to $\delta^- S$. Thus $|\delta^+ S|$ and $|\delta^- S|$ remain unchanged.

- $v \notin S$ and $w \in S$: This case is symmetric to the previous case.

The same line of arguments holds when $u \notin S$ so we do not explicitly state these additional cases here. Note that $|\delta^+ S| = |\delta^- S|$ holds in a clique and that $\delta^+ S$ and $\delta^- S$ always grow by the same amount of edges when a new node is inserted. Thus, $|\delta^+ S| = |\delta^- S|$ will still hold after the insertion. ■

In the following we prove that the evolving digraphs are expanders with high probability. Before we do this we give a formal definition of directed expander graphs by transferring Definition 3.2 to the domain of digraphs.

Definition 4.4 (Expander digraph) *A digraph $G = (V, E)$ provides* expansion $\beta > 0$*, or is a* β-expander*, if for all node sets S with $|S| \leq |V|/2$ it holds that*

$$|\delta^+ S| \geq \beta |S| \text{ and } |\delta^- S| \geq \beta |S|.$$

Theorem 4.5 *A d-regular digraph $G = (V, E)$ with $d \in \Omega(\log n)$ constructed using the directed-pegging operation is a $\Theta(d)$-expander w.h.p.*

Proof: Let $V = \{v_1, \ldots, v_n\}$ be the nodes of G and let the indices $1, \ldots, n$ denote the order of their insertion into G. Without loss of generality we assume n to be even. Furthermore, we define $V_1 = \{v_1, \ldots, v_{n/2}\}$ and $V_2 = \{v_{n/2+1}, \ldots, v_n\}$. To proof the theorem we will analyze the development of the size of the edge boundaries $\delta^+ S$ and $\delta^- S$ for all $S \subset V$ with $|S| \leq n/2$ during the insertion process. We start our analysis at the point in time when all nodes in V_1 have already been inserted into G and assume that $|\delta^+ S| = 1$ at this point in time (this is the lowest possible value for $|\delta^+ S|$ since G is connected). Note that all bounds on probabilities presented in the remainder of this proof refer to points in time after the insertion of all nodes in V_1. From Lemma 4.7 we know that $|\delta^+ S| = |\delta^- S|$ does hold so that we can restrict ourselves to the analysis of $\delta^+ S$ here. In the analysis we distinguish between subsets S with $|S \cap V_1| \geq |S|/2$ and $|S \cap V_2| \geq |S|/2$, starting with the former.

$|S \cap V_1| \geq |S|/2$ means that the majority of nodes in S belongs to the first half of nodes inserted into G. Now consider the nodes in $V_2 \setminus S$. Since $|S \cap V_2| \leq |S|/2$ and $|S| \leq n/2$ it follows that there are at least $n/4$ nodes in $V_2 \setminus S$. When these nodes choose edges lying in S during their insertion $\delta^+ S$ will grow. For the choice of a single edge we observe the following probability:

$$\Pr[v \in V_2 \setminus S \text{ chooses edge in } S] \geq \frac{|S|}{2n} - \frac{|\delta^+ S|}{dn}.$$

We now show that after inserting all the nodes in $V_2 \setminus S$ the edge boundary for an arbitrary subset $S \subset V$, with $|S \cap V_1| \geq |S|/2$ and $|S| \leq n/2$, satisfies $|\delta^+ S| > \frac{d|S|}{16}$ with high probability. First of all note that $\delta^+ S$ can not shrink and that the probability to add edges to $\delta^+ S$ does shrink with growing $\delta^+ S$. As long as $|\delta^+ S| \leq \frac{d|S|}{8}$ we observe the following lower bound for the probability to add edges to $\delta^+ S$

$$\begin{aligned} \Pr\left[\delta^+ S \text{ will grow}\right] &\geq \frac{|S|}{2n} - \frac{|\delta^+ S|}{dn} && (4.4) \\ &\geq \frac{|S|}{2n} - \frac{d|S|}{8dn} && (4.5) \\ &= \frac{3|S|}{8n}. && (4.6) \end{aligned}$$

This lower bound allows us to model the growth of the edge boundary as a Bernoulli trial. The success probability is given by Inequality 4.6 and since each node in $V_2 \setminus S$ will choose d edges we have a total number of $d|V_2 \setminus S| \geq dn/4$ trials. Assuming the

minimum number of $dn/4$ trials, it follows that $E\left[|\delta^+ S|\right] \geq \frac{3}{32} d|S|$. Applying Chernoff bounds we can bound the probability of being successful less than $\frac{d|S|}{16}$ times during these trials as follows

$$\begin{aligned}
\Pr\left[|\delta^+ S| \leq \frac{d|S|}{16}\right] &= \Pr\left[|\delta^+ S| \leq \left(1 - \frac{1}{3}\right)\frac{3}{32} d|S|\right] \\
&\leq e^{-\frac{1}{2}\left(\frac{1}{3}\right)^2 \frac{3}{32} d|S|} \\
&= e^{-\frac{1}{192} d|S|}\,.
\end{aligned}$$

Next, we consider the subsets S with $|S \cap V_2| \geq |S|/2$ and show that these have a large edge boundary after inserting all n nodes. Now, the majority of the nodes in S belongs to V_2, i.e. the last half nodes to be inserted. Once again we assume that the nodes of V_1 have already been inserted and that $|\delta^+ S| = 1$ at this point in time. Note that the edges chosen by nodes in $S \cap V_2$ are likely to lie in $\bar{S} := V \setminus S$ and therefore have the potential to add edges to $\delta^+ S$. When all nodes in V_1 have been inserted into G it holds that $|\bar{S}| \geq \frac{n}{4}$ and thus the probability for nodes in $S \cap V_2$ to choose an edge not lying in S after this point in time is bounded by

$$\begin{aligned}
\Pr\left[v \in S \cap V_2 \text{ chooses edge in } \bar{S}\right] &\geq \frac{d|\bar{S}| - |\delta^+ \bar{S}|}{dn} \\
&\geq \frac{1}{4} - \frac{|\delta^+ \bar{S}|}{dn}\,.
\end{aligned}$$

We will now make use of the fact that $|\delta^+ \bar{S}| = |\delta^- \bar{S}| = |\delta^+ S|$, see Lemma 4.7. As long as $|\delta^+ S| \leq \frac{d|S|}{8}$ does hold we observe the following lower bound for the probability to add edges to $\delta^+ S$:

$$\begin{aligned}
\mathit{Prob}\left[\delta^+ S \text{ will grow}\right] &\geq \frac{1}{4} - \frac{|\delta^+ \bar{S}|}{dn} && (4.7)\\
&\geq \frac{1}{4} - \frac{|S|}{8n} && (4.8)\\
&\geq \frac{1}{4} - \frac{\frac{n}{2}}{8n} && (4.9)\\
&= \frac{3}{16}\,. && (4.10)
\end{aligned}$$

Again we can model the growth of the edge boundary as a Bernoulli trial with success probability given by Inequality 4.10. Since $|S \cap V_2| \geq |S|/2$ we have at least $d|S|/2$ trials and $E\left[|\delta^+ S|\right] \geq \frac{3}{32} d|S|$. Applying Chernoff bounds we have

$$\begin{aligned}
\Pr\left[|\delta^+ S| \leq \frac{d|S|}{16}\right] &\leq e^{-\frac{1}{2}\left(\frac{1}{3}\right)^2 \frac{3}{32} d|S|} \\
&= e^{-\frac{1}{192} d|S|}\,.
\end{aligned}$$

It remains to sum up the error probabilities for all subsets $S \subset V$ with $|S| = m \leq n/2$. Assuming $d \geq c \log n$ for some constant c chosen large enough we have

$$\begin{aligned}
\sum_{m=1}^{n/2} \binom{n}{m} e^{-\frac{1}{192}dm} &\leq \sum_{m=1}^{n/2} n^m e^{-\frac{1}{192}dm} \\
&\leq \sum_{m=1}^{n/2} n^{\left(1-\frac{c}{192}\right)m} \\
&\leq \frac{n}{2} n^{1-\frac{c}{192}} \\
&\leq n^{-c'},
\end{aligned}$$

so the probability that there is a subset S with $|\delta^+ S| = |\delta^- S| < \frac{d|S|}{16}$ is polynomially small in n. ∎

The constant hidden in the degree $\Omega(\log n)$ used in Theorem 4.5 is quite large. Note that we made no effort to minimize this constant — this could be achieved by a more exact distinction of the subsets S according to the points in time when their nodes are inserted into the graph, for example.

The domain of digraphs generated by the directed-pegging operation is a subdomain of connected regular digraphs since the operation does not allow to construct large cliques as subgraphs. The argumentation to prove this is the same as in case of the (undirected) pegging operation, see [51]. A local version of the directed-pegging operation could be derived by using d random walks to choose the random edges, with the length of the random walks chosen according to the mixing time of the digraph. However, the mixing time is unknown and there is no simple way for the peers to calculate it in a distributed way. Furthermore, the mixing time may change with joining and leaving peers. It is conceivable that random walks of length logarithmic in the number of nodes are sufficient to construct expander digraphs, experimental results in [51] give evidence to this.

5

Conclusion and Open Problems

In the first part of this thesis we have presented several random graph transformations for maintaining a stable random backbone for peer-to-peer networks: The family of Flipper operations for undirected connected graphs in Chapter 3 and the Pointer-Push&Pull operations for connected out-regular multi-digraphs in Chapter 4. In Chapter 2 we defined four properties desirable for random transformations of graphs: soundness, generality, feasibility, and fast convergence. Figure 5.1 presents a summary of the graph transformations presented in this thesis and the switch operation proposed by Cooper et al. [26, 27] with respect to these properties. The Archilles heel of the switch operation is that it cannot guarantee connectivity and thus will disconnect a network in the long run unless additional mechanism to guarantee connectivity are used. So, the switch operation is neither feasible nor does it provide generality when used in a networking concept since not every graph of the domain is reachable in the limit. Flipper and Pointer-Push&Pull operations do not have this problem since they were designed to guarantee connectivity — we even presented simple mechanisms to guarantee connectivity in case of concurrent operations. Furthermore, all findings are rigorously proved using Markov theory. In our view this is the only reasonable way as graph transformations like the Pointer-Pull operation perform very well in simulations of large networks, yet eventually disconnect the network as we have proven here.

Despite of their simplicity these operations are able to establish a stable network and we have shown that in the limit truly random graphs respectively expander graphs

	switch [26, 27]	1-Flipper	k-Flipper	Pointer-Push&Pull
domain	regular graphs	connected regular graphs	connected regular graphs	connected regular multi-digraphs
soundness	yes	yes	yes	yes
generality	no	yes	yes	yes
feasibility	no	yes	yes*	yes
convergence	yes	yes	yes	yes

Figure 5.1: *Comparing the properties of graph transformations (*k-Flipper is only feasible for small values of k)*

evolve. Therefore, small diameter and high connectivity is guaranteed. An interesting feature — making these graph transformations an ideal maintenance operation for unstructured peer-to-peer networks — is that they allow to recover from any worst case situation. For the 1-Flipper operation we have shown that it would indeed drastically improve the network structure of Gnutella. Furthermore, the 1-Flipper operation could be used to improve the unstructured overlay network proposed by Law and Siu [64] which is composed of d Hamilton cycles and is not able to recover from degenerate states (see Page 11).

From a practical point of view the Pointer-Push&Pull operation directly competes with the 1-Flipper operation. Both graph transformations are able to maintain truly random networks, yet in different domains: The 1-Flipper operation maintains undirected connected regular graphs, whereas the Pointer-Push&Pull operation maintains out-regular weakly connected multi-digraphs. While the domain of the 1-Flipper operation may be the more common one for random networks, there is a strong argument in favor of the Pointer-Push&Pull operation and the — at first glance incongruous — use of multi-digraphs: a Pointer-Push&Pull operation needs only two nodes to exchange two messages per operation, while the 1-Flipper operation needs four nodes to communicate with each other. This implies lower network overhead and improved locality for each operation. So, the Pointer-Push&Pull will be the operation of choice unless there are strong arguments in favor of using undirected networks in a particular application scenario.

In Sections 3.3.1 and 4.2.2 we have seen that there exist join operations constructing expander (di)graphs right away. However, the join operations for which we were able to prove that they construct expander graphs required global knowledge of the network structure. Furthermore, one should not forget that — even if a join operation constructs expanders right away — it is still mandatory for a peer to periodically verify its neighborhood in a dynamic network. So, at the latest with the introduction of the Pointer-Push&Pull operation it is not that essential that join operations construct a decent topology, since Pointer-Push&Pull operations can replace periodic verification messages and maintain a random network at the same time without causing communication overhead. Last but not least a continuously changing and random topology offers further advantages. The change of a peers neighborhood via random graph transformations will make it harder for a group of malicious peers to take control over parts of the network. Furthermore, a peer will be connect to every other peer over time so random graph transformations are a tool to exchange or spread information in the network.

Open Problems

The research in the field of local random graph transformations is still in its infancy. Bounding the convergence rate of local graph transformations turned out to be hard and the experimental results presented in Section 3.4 suggest that the best bound known for the 1-Flipper operation [28] is far from the true behavior of this graph transforma-

tion. For the k-Flipper we were only able to bound the number of operations needed to reach an expander graph for extra-ordinary large k, i.e. k chosen such that both flipping edges are chosen uniformly at random. So, it remains open to show how many operations are needed for reasonable values of k, i.e. $k \in \mathcal{O}(\log n)$. Furthermore, the convergence rate has been analyzed in static graphs without nodes joining and leaving so far.

For the Pointer-Push&Pull operation no bounds on the convergence rate have been proven so far. The hope that bounding the convergence rate of the Pointer-Push&Pull operation would be easier than in case of the 1-Flipper since the domain of out-regular multi-digraphs is less restrictive than the domain of regular undirected graphs has not proven to be true so far. In case of undirected graphs we were able to examine the convergence behavior of the 1-Flipper by simulations. Unfortunately we are not able to bound the expansion of a digraph with help of eigenvalues. We certainly could interpret the evolving digraphs as undirected by neglecting the direction of edges. This however implies that the degree sequence of the graph would change constantly so that it is rather doubtful to obtain useful results this way.

Another point is that we have only discussed undirected graphs and multi-digraphs since it turns out that the Pointer-Push&Pull operation is unable to maintain simple digraphs. Yet, it is not clear if there exists a similar operation for simple digraphs suitable for peer-to-peer networks.

Part II

Structured Networks

6

Introduction to Structured Networks

The local graph transformations introduced in Part I of this thesis improve the robustness of unstructured peer-to-peer networks under churn. However, they do not help to overcome the major drawback of unstructured networks: The lack of efficient query algorithms. Queries still have to be performed by broadcasts of limited depth, imposing a lot of traffic or by random walks, which reduce network traffic but massively increase search latency [70].

Structured peer-to-peer networks overcome the shortcomings of unstructured peer-to-peer networks concerning efficient queries. Starting with CAN [95], numerous network designs based on distributed hash tables (DHT) [61] have been proposed. In DHT based networks, peers and resources (e.g. data) are mapped into a virtual space by hash functions to assign data to peers. Then, data is assigned to the peer which is closest in the virtual space, for example. This way peers are able to calculate the position of a data element in the virtual space and then route to the peer hosting that data element. Additionally, the use of pseudo random hash functions provides simple and efficient load balancing since data elements and peers will be evenly distributed in the virtual space. The simplicity and elegance of the DHT scheme let to a large number of DHT based peer-to-peer networks proposed by the scientific community, e.g. Chord [114, 113], Viceroy [76], Pastry [99], Tapestry [52], Koorde [60], D2B [37], and the Distance Halving network [81, 80, 119]. These networks typically provide logarithmic hop distance for lookups and mostly differ in the topology chosen to connect the peers (see Figure 6.1). So, the design of static DHT based networks is well understood [65].

However, the use of DHTs also involves a few drawbacks compared to unstructured networks. First of all, DHT based networks are harder to maintain under churn than unstructured peer-to-peer networks in general [22]. Second, the use of hash functions destroys any semantic interdependencies between data since data elements are mapped to (pseudo) random positions in the virtual space. Thus, *locality* is neglected during data placement and data items stored at the same peer are usually completely unrelated. Consequently, DHTs are considered to be inherently ill-suited to complex queries such as range queries and in this regard constitute a step backwards compared to unstructured networks [13].

6.1 Locality in Peer-to-Peer Networks

DHTs certainly constitute an important step towards scalable and robust peer-to-peer networks. However, as we have seen the efficiency is bought dearly by neglecting locality during data placement, resulting in the restriction to exact match queries. This clearly shows that locality is of major importance in peer-to-peer networks. Here, we distinguish three types of locality: Network locality, information locality, and interest locality. In the following we will describe these types of locality and briefly discuss the relevant literature for each type of locality.

6.1.1 Network Locality

While the typical measure to evaluate routing algorithms is the hop number, this alone is not a good measure if the goal is to provide short response times. The reason for this is that a hop connecting peers in Greece and Australia has higher latency than a hop connecting peers in the same building for example. This observation leads to the following definition of network locality.

Definition 6.1 (Network locality) *A peer-to-peer network provides network locality if lookup operations can be performed with small latency.*

Network locality has been addressed early by the research community and several peer-to-peer networks providing network locality have been proposed. Pastry [99] and Tapestry [52], based on the seminal work of Plaxton et al. [89], were the first DHT based peer-to-peer architectures providing network locality innately.[1] Other architectures have been extended to support network locality, e.g. Dabek et al. [30] as well as Montresor et al. [79] extend the Chord architecture [114] in this regard. The crux in providing network locality is to find latency wise close neighbors without generating too much additional network traffic.

6.1.2 Information Locality

As we have seen during our discussion of DHTs locality and data placement plays a key role when it comes to supporting complex queries such as range queries.

Definition 6.2 (Information locality) *A peer-to-peer network provides information locality if closely related data elements are stored on network-wise close peers.*

The problem of supporting range queries in peer-to-peer overlays has been identified early by several researchers, e.g. [49, 54]. Ratnasamy et al. proposed the trie based Prefix Hash Tree (PHT) [92, 93], where prefixes of a trie are hashed onto an arbitrary DHT

[1] We are aware that network locality has also been addressed in CAN [95] and Chord [114]. However, the solution presented in [95] comes with the drawback of loosing the load balancing feature and the solution presented in [114] is sketched very briefly and is not very effective. So, we do not consider them here.

network. An advantage of their approach is that this way the load balancing functionality of DHTs can still be used. However, as mentioned above DHTs are inherently ill-suited to range queries and thus it is hardly surprising that the lookup in PHT is not as efficient as in DHTs, i.e. a lookup requires $\mathcal{O}(\log^2 n)$ hops, with n denoting the number of peers.

The skip list [91] based Skip Graphs by Aspnes and Shah [7] belongs to the most prominent and earliest peer-to-peer networks supporting range queries efficiently. Yet, range query support in Skip Graphs is bought dearly by the loss of load balancing: Resources, e.g. data files, are managed by the peer hosting that resource respectively. Consequently, a peer hosting k resources has to maintain k nodes in the Skip Graph overlay. So, in a Skip Graph with n peers and m resources ($m \gg n$ is a typical assumption) a peer has to maintain $\mathcal{O}(k \log m)$ links, whereas the number of links to be maintained in DHT based networks typically is $\mathcal{O}(\log n)$ [114] or constant [80, 76, 60] per peer, and thus independent of the number of resources.

Several tree based peer-to-peer architectures supporting (multi-dimensional) range queries and providing load balancing at the same time have been proposed, e.g. [66, 56, 2, 57, 29, 78, 67, 33, 123]. In the following we briefly discuss P-Grid [2], Baton [56], and DPTree [66].

P-Grid [2, 3, 1] abstracts a binary trie structure defined by the data available in the network. Each peer is responsible for a particular prefix of the trie and maintains links to random peers of every subtree neighboring its own prefix. However, it is not clear if the links to subtrees selected in P-Grid are truly random. The load balancing mechanism used in P-Grid is based on heuristics. Without a central load balancing instance assigning prefixes to the peers, peers have to determine their prefix to manage in a distributed manner. This can result in complex dependencies between all peers. So, the load balancing mechanism in P-Grid is based on random processes and the fact that P-Grid needs an extra bootstrapping mechanism just for an initial network state underlines its complexity. Furthermore, it has been critiqued by Ganesan et al. that there is no formal characterization of imbalance ratios and balancing costs in P-Grid [39].

Baton (BAlanced Tree Overlay Network) [56] is based on a binary balanced tree structure. Each peer is responsible for a particular node of the tree. Besides the obligatory parent/child links of a tree, the peers of a layer of the tree are connected by a Chord like ring with pointers to peers in distance 2^i on the ring. Load balancing is achieved by shedding load to an adjacent lightly loaded peer or by a leave/re-join mechanism. In the latter case, the tree may get unbalanced so that it has to be rebalanced.

DPTree (Distributed Peer Tree) [66] is a peer-to-peer architecture inspired by balanced tree indexes (R-Tree [47]). The authors propose to decouple the tree structure from the actual structure of the overlay. This is achieved by using a Skip Graph as overlay structure and choosing peer identifiers such that these represent paths from the root to leaves of the tree structure. Balancing of access load is done with the help of a wavelet based mechanism that is used to choose peer identifiers. Peers noticing to be overloaded may shed part of their load to neighboring peers. Unfortunately, a cost analysis of the load balancing mechanism is not given in [66] and it may be critiqued that the authors do

not verify the robustness of DPTree under network dynamics. Furthermore, as mentioned by Tran and Nguyen [117], the costs of rebuilding the balanced index tree upon structural changes remain unclear.

So, there exists large number of peer-to-peer network architectures that overcome the crucial limitation of DHTs to exact match queries. However, all the network architectures mentioned above either do not provide load balancing at all (Skip Graphs) or make use of complex heuristics (e.g. P-Grid, Baton, DPTree), which are in stark contrast simple and efficient load balancing provided by the DHT scheme and often make a formal analysis impossible. The important thing to note here is that these networks — although they allow to process range queries efficiently — are not superior to DHTs in every respect.

6.1.3 Interest Locality

In the Web certain data is intrinsically local, e.g. most of all greek web-sites are created in Greece and accessed from computers in Greece. Hence, it makes sense to store such data on peers located in Greece.

Definition 6.3 (Interest locality) *A peer-to-peer network provides interest locality if peers can choose on providing lookup service and data storage for certain data. If peers choose to provide certain data, then the network structure allows efficient lookup to data relevant to a peer.*

Interest locality is rarely addressed in peer-to-peer networks. An exception is the SkipNet [50] architecture introduced by Harvey et al. SkipNet is based on skip lists and therefore closely related to the Skip Graphs [7, 8] architecture mentioned above.[2] So, it is hardly surprising that SkipNet shares some shortcomings with Skip Graphs, i.e. the lack of load balancing. In fact the authors present a way to provide a *constrained* form of load balancing in SkipNet, but we will not go into detail here and focus on interest locality instead. In SkipNet peers may choose arbitrary name id's. If all peers of a domain (e.g. '.de' or '.it') choose their name id to begin with their domain, then the peers of the same domain will be neighboring in the id space of SkipNet. Using the domain as prefix for data elements as well, allows to control data placement. Since the routing algorithm ensures that a query, which has reached the target domain will never leave it again, SkipNet provides a form of interest locality. This, however, comes at the price of diminishing information locality: To retrieve all documents relevant to a query each domain has to be queried separately then. So, there is a trade-off between information locality and interest locality in SkipNet.

The table shown in Figure 6.1 gives an overview of peer-to-peer networks and the types of locality they provide. Network designs based on plain DHTs such as Chord [114], Viceroy [76], and Koorde [60], to name just a few, do not support any type of

[2] Note however that SkipNet and Skip Graphs have been developed independently and with different focus.

Network	Topology	Network locality	Information locality	Interest locality
Gnutella [44]	random graph	no	yes	no
Chord [114]	hypercube	no	no	no
CAN [95]	torus	no	no	no
Viceroy [76]	butterfly	no	no	no
Koorde [60]	de Bruijn	no	no	no
Distance Halving [80]	de Bruijn	no	no	no
Pastry [99]/Tapestry [52]	mesh of trees	yes	no	no
Skip Graphs [7]	skip list/rings	no	yes	no
Prefix Hash Tree [92]	DHT/trie	no	yes	no
DPTree [66]	Skip Graph	no	yes	no
Baton [56]	B^+-tree/rings	no	yes	no
P-Grid [2]	mesh of trees	no	yes	no
SkipNet [50]	skip list/rings	no	yes*	yes*
3nuts [71]	tree/random graphs	yes	yes	yes

Figure 6.1: *Overview of peer-to-peer networks and provided types of locality (*support of interest locality in SkipNet will diminish information locality and vice versa).*

locality, tree and skip list based networks provide information locality, etc. In the bottom line many peer-to-peer architectures supporting either network, information, or interest locality have been proposed in the past years. To the best of our knowledge the 3nuts architecture introduced in the following chapter is the first peer-to-peer network providing all three types of locality at the same time.

7

3nuts: Combining Random Networks, Search Trees, and DHTs

In this chapter we present the 3nuts peer-to-peer network, which provides network locality, information locality, and interest locality at the same time. 3nuts combines concepts of unstructured and structured peer-to-peer networks to overcome their particular shortcomings while keeping their individual strength. So, one of the most robust backbone structures is combined with one of the most efficient lookup methods: random networks and search trees. These two structures complement each other excellently. On the one hand, random networks are provably robust, but there are no efficient lookup algorithms. On the other hand, search trees allow efficient and non-trivial lookups like range queries, but are not robust against node failures and do not distribute access load evenly among peers when the tree structure is directly translated into a network, i.e. nodes of the tree are replaced by peers.

In a nutshell the 3nuts network can be described as follows. In order to preserve semantic relationship of data, peers resemble the prefix search tree (*data tree*) defined by all data available in the network and build the so called *network tree*. In the network tree each node of the data tree is represented by a directed, connected random network of constant degree. Starting with the root of the tree, that is represented by a random network containing all peers and thus forms a reliable backbone, peers are recursively assigned to subtrees using a variation of DHTs. This recursive assignment is continued until there is only a single peer left in a subtree of the data tree. So, a peer actually chooses a path from the root to a leaf of the network tree and participates in each of the random networks representing the corresponding nodes. To allow efficient routing, each peer maintains so called *branch links* to a random peer of each subtree neighboring its own path.

Random networks represent a core component of 3nuts and we make extensive use of their excellent communication properties to maintain the network structure under churn. 3nuts rigorously forgoes any form of central structures and coordination mechanisms, i.e. the network structure is maintained using local handshake operations applied in the random networks only. Furthermore, one of the major design goals of 3nuts was to forgo the use of heuristics wherever possible. So, 3nuts makes use of the

simple and efficient load balancing provided by DHTs — proven successful in many peer-to-peer networks [114, 76, 99, 52, 60, 37, 80] — and thus differs greatly from the majority of tree based peer-to-peer networks that rely on heuristics for load balancing (e.g. [2, 56], see Section 6.1.2). Furthermore, the use of the Pointer-Push&Pull operation to maintain the random networks guarantees that these are truly random and provide excellent communication properties and robustness. Last but not least we will prove in this chapter that the number of hops needed by the lookup operation is bounded by $\mathcal{O}(\log n)$ w.h.p. regardless of the structure of the data tree.

7.1 The 3nuts Architecture

In this section we describe the overall structure of the 3nuts network, the interplay of tree structure and random networks, a peers local view of the network, and how the network structure is maintained solely by local periodic handshake operations.

In contrast to the standard DHT approach where data is assigned to peers, peers are assigned to data in 3nuts. Thus, an existent ordering (e.g. lexicographical) of data is preserved, which is essential to be able to process range queries efficiently. As a consequence, the actual network structure depends on the data currently available in the network and may change when new data is inserted. Before we describe how data forms the tree structure and how peers recreate this tree structure, note that in 3nuts peers only store references to data. The actual data files remain at the peers owning them and these peers inform the peers responsible for maintaining their data's references regularly.

7.1.1 Basic Concepts: Data Tree and Network Tree

The *data tree* is the prefix tree (trie) defined by the identifiers of data available in the network (see Figure 7.1). Principally, any other tree based data structure could be used as well. The main reason why we prefer a trie to more sophisticated balanced trees is its simplicity. Our point is that simplicity can turn out to be crucial in a dynamic network under churn and allows to cope with higher churn rates. On the other hand using an unbalanced tree requires more elaborate mechanisms to balance the load among peers. Therefore, the load balancing mechanism has to be designed carefully so that it does not dissave the simplicity of the trie. There are many possibilities to define the load of a subtree, e.g. accesses frequency or number of data elements in a subtree. Here, we use the latter definition of load and balance the number of data elements managed by a peer. Note however, that 3nuts is able to support other definitions of load as well.

The peers recreate the data tree in a distributed and scalable way. Each node of the tree is replaced by a random network of peers, starting with the root of the tree which is replaced by a random network containing all peers. Then, each of the peers is assigned to one of the child nodes of the root using a simple randomized load balancing mechanism that assigns peers to subtrees with probabilities proportional to the load of the subtree. Peers assigned to the same child node (respectively subtree) then form another

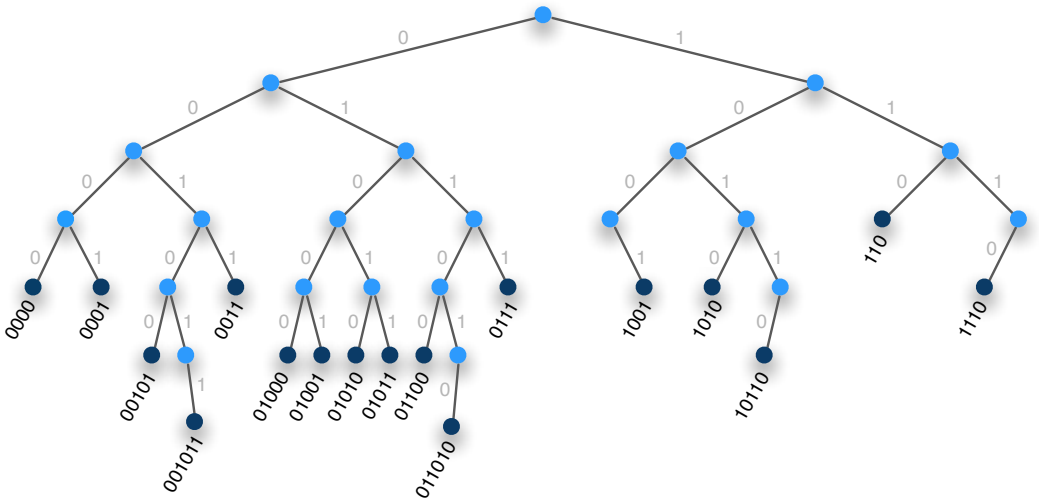


Figure 7.1: *Example of a prefix tree. Nodes storing data elements are depicted black.*

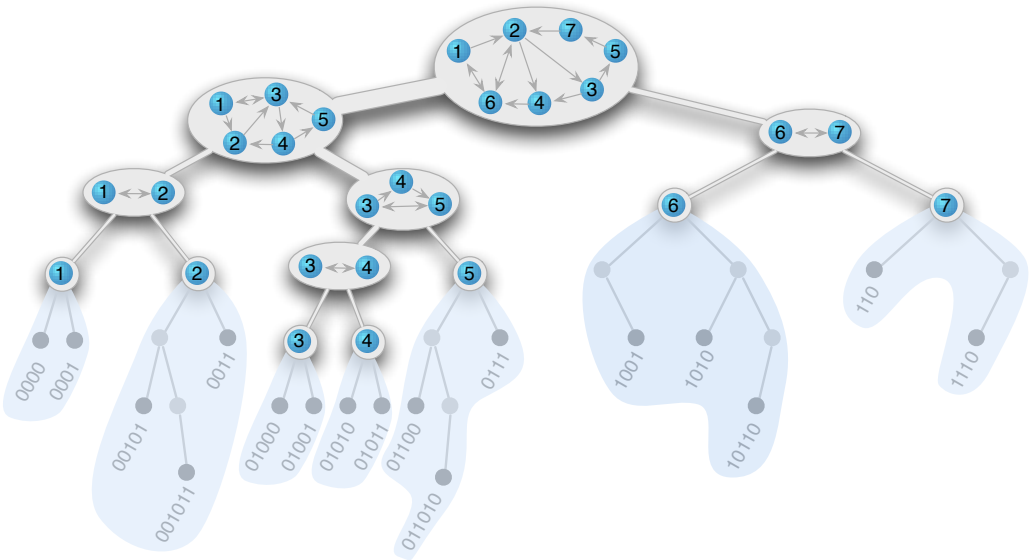


Figure 7.2: *Global view of the network tree with 7 peers.*

random network. This procedure is continued recursively until either a peer is the only one assigned to a subtree or a leaf of the data tree is reached (see Figure 7.2). For a peer this actually means to choose a path starting at the root of the data tree leading down the tree. We will refer to the combination of random networks and the data tree as *network tree* from now on. The random networks play an important role in 3nuts. Besides making the overlay robust, they are used to spread information about the tree structure, load of subtrees, etc., among peers. We will come back to the random networks in Section 7.1.3 and describe their maintenance in detail.

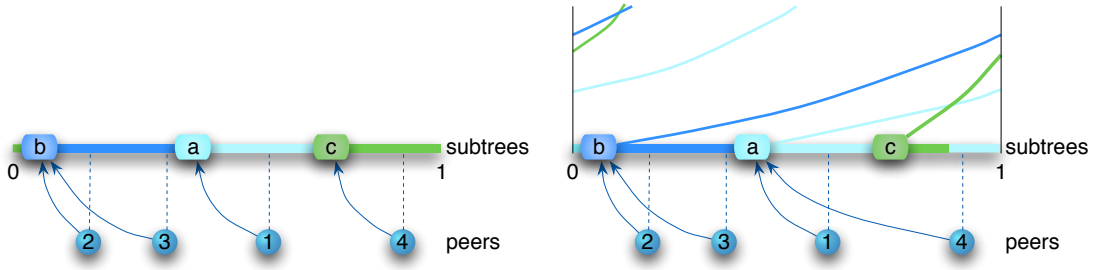
7.1.2 Peer Assignment, Load-Balancing, and Responsibilities

A peer is responsible for all data in the subtree rooted at the leaf of its path in the network tree (cf. subtrees highlighted light blue in Figure 7.2). Since 3nuts also allows to store data in internal nodes of the tree, each internal node of the network tree has a particular peer that is responsible for managing data in this node.

The recursive assignment of peers to subtrees is done using distributed heterogeneous hash tables (DHHT) [102], which is an extended form of consistent hashing, a.k.a. distributed hash tables (DHT) [61], to support non-uniform load distributions. In our case the actual number of peers assigned to a subtree depends on the load, i.e. the amount respectively popularity of data stored in this subtree. We will give a brief description of consistent hashing and then describe the weighting extension, in both cases with focus on our particular application. To avoid confusion it is important to recall that we assign peers to data (respectively subtrees) while the usual approach is just the opposite way around, i.e. data is assigned to peers. This allows us to preserve a given ordering of data elements and thus overcome the crucial limitation of DHT based peer-to-peer overlays to exact match queries.

We exemplify the peer assignment in an arbitrary node v of the tree with child nodes $v_1, \ldots, v_k$. Furthermore, we assume that peers $p_1, \ldots, p_m$ have been assigned to v. Consistent hashing uses a "two-sided" hashing into a continuous range $M = [0, 1)$ to assign peers to the subtrees rooted at $v_1, \ldots, v_k$. Peers and subtrees are mapped randomly into M by hash functions h_1 respectively h_2. Then, peers are assigned to the subtree which is closest to them in descending direction in M (see Figure 7.3). So far all peers and subtrees are handled as if they are uniform. While this is reasonable and intended in the case of peers, it is very likely that some subtrees hold more data than others and thus generate a higher load to the peers in these subtrees. So, when assigning peers uniformly to subtrees, it is very likely that the load is not spread evenly among the peers.

To take different weights of subtrees into account and to make the number of peers assigned to a subtree reflect its weight, the scheme is extended as follows: Let $w_1, \ldots, w_k \in \mathbb{R}^+$ denote the weights of the subtrees rooted at $v_1, \ldots, v_k$ and let $p'_i = h_1(p_i)$ and $v'_j = h_2(v_j)$ denote the position of peer p_i, respectively the subtree rooted at v_j, in M.



(a) DHT (b) DHHT

Figure 7.3: *Assigning 4 peers to subtrees a, b, and c using a DHT respectively DHHT (dashed lines show the positions peers have been hashed to in the $[0,1)$ interval).*

Then, we define a scaled distance function

$$L_w\left(p_i, v_j\right) = \frac{-\ln\left(\left(1-\left(p_i' - v_j'\right)\right) \mod 1\right)}{w_j},$$

with $x \mod 1 := x - \lfloor x \rfloor$. Now each peer p_i is assigned to the subtree rooted at the node v_j minimizing the term $L_w(p_i, v_j)$ (see Figure 7.3). For peer p_i and node v_j we will also refer to the value of this function as *height*.

If we further extend the scheme described so far to use double hashing, a peer p_i is mapped into M using $p_i' = h_1(p_i)$ as before, but each subtree rooted at v_j, $1 \leq j \leq k$, has an individual hash function h_{v_j}. A peer then calculates its heights for each v_j at position $h_{v_j}(p_i')$ and is assigned to the subtree minimizing the height. Using the combination of DHHT and double hashing the following theorem does hold.

Theorem 7.1 *Assigning peers to subtrees using the DHHT scheme in combination with double hashing it holds w.h.p. that*

$$\Pr[p_i \textit{ is assigned to } v_j] = \frac{w_j}{\sum_{l=1}^{k} w_l}.$$

Proof: The theorem is a direct consequence of Theorem 10 in [102]. ■

Hence, peers are assigned to subtrees with probabilities proportional to the weights of the subtrees. Note that the runtime of the assignment using double hashing is linear in k, i.e. the number of subtrees. However, in our scenario k is a small constant.

As already mentioned, every node of the network tree has a designated *responsible peer*. This peer has to, amongst other things, manage references to data stored in the node, create new subtrees when data is inserted, and delete empty subtrees. The responsible peer for a node v of the tree is the peer that has been assigned to v with the lowest height. So, the decision about responsibility is made in the parent node of v.

This choice is reasonable since the selected peer will be the last peer to leave the subtree rooted at v if this subtree's load decreases or the load of other subtrees rooted at v's siblings increases. Furthermore, the selection mechanism ensures that the selected responsible peers are truly random and thus the responsibility for internal nodes of the network tree is spread evenly among all peers.

When Hashing peers onto subtrees, it is possible that no peer is assigned to a specific subtree and thus no peer is responsible to manage the data in that subtree. This will principally happen to subtrees with low weight or leaves of the network tree. If there is such a vacant subtree, then a particular peer is selected to manage this subtree by a mechanism which we call *shanghaiing* (inspired by the english slang term, describing the common act of forcibly conscripting someone to serve a term working on a ship, usually after having been rendered senseless by alcohol or drugs, during the 19th century [88]). Therefor, the peer that has the lowest height for the vacant subtree is selected to be shanghaied. In the scenario of Figure 7.3(b) peer 4 would get shanghaied to be responsible for subtree c. Again, this choice is obvious since the chosen peer will be the first to be assigned regularly to the vacant subtree if the weight of the subtree increases. A shanghaied peer is responsible for a subtree until either another peer is assigned regularly to that subtree or a peer with lower height is shanghaied to be responsible for that subtree.

7.1.3 Maintaining Random Networks

Random networks play an important role in 3nuts. Their simplicity and provable robustness make them an ideal tool to improve the churn and fault resilience of a network. As we have seen in Section 7.1.1 all peers that have been assigned to a particular node of the network tree are connected by a random network. Here, we use d-out-regular multi-digraphs with small constant d, e.g. $d = 3$. The random networks are maintained using the Pointer-Push&Pull operation described in Chapter 4 (see Figure 4.3 on Page 45). Recall that a single Pointer-Push&Pull operation involves only two messages between two peers and thus can be used to replace the mandatory heartbeat (ping) messages used to verify the availability of neighbors in dynamic networks. Consequently Pointer-Push&Pull operations do not introduce additional traffic to the network.

We have also shown that Pointer-Push&Pull operations guarantee the resulting graph structure to be truly random in the limit (cf. Theorem 4.2, Page 50). An important consequence of Theorem 4.2 is that a peer will see every other peer participating in the same random graph, i.e. node of the network tree, over time. Hence, Pointer-Push&Pull operations constitute an excellent tool to exchange information about the tree structure, weights, etc. among peers without inducing additional traffic to the network. This fact is the main reason for us to prefer multi-digraphs over the more common domain of regular undirected graphs, which can be maintained using the related 1-Flipper operation described in Chapter 3.

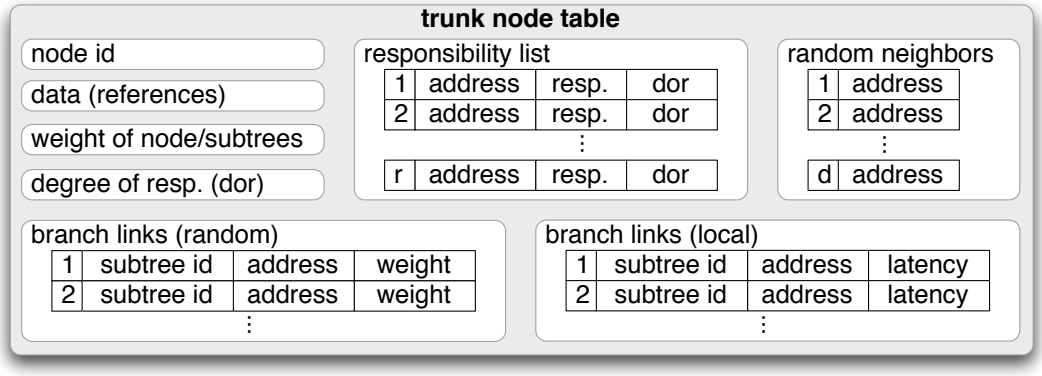


Figure 7.4: *Information a peer holds for a trunk node.*

7.1.4 A Peer's Local View

We have seen how peers recreate the data tree by choosing their path in the network tree and replacing nodes of the data tree with random networks. A peer therefore only has a limited local view of the network tree. We refer to the nodes a peer has been assigned to (either regularly or shanghaied) as *trunk nodes* of the peer. For each of its trunk nodes a peer maintains a so called *trunk node table* (see Figure 7.4). In this table node id, weight, subtrees, and (if the peer itself is responsible for the node) references to data are stored. Furthermore, the trunk node table contains the following lists:

Responsibility list A list of the r peers with the highest responsibility for this node. Responsibility is determined by the height with which a peer has been assigned to the node by the assignment process in the parent node (see Section 7.1.2). Here, lower height connotes higher responsibility. While it would be sufficient to store one peer responsible for the node, having a list of r peers with highest responsibility helps greatly to improve the stability of the network under churn. Here, choosing r as a small constant is completely sufficient.

Random neighbors A list of the d neighboring peers in the random network corresponding to this node of the network tree. This list is used to perform the regular Pointer-Push&Pull operations and thus the list is constantly refreshed and its randomness is guaranteed by Theorem 4.2.

Branch links (random and local) To allow efficient routing a peer maintains *branch links* to some peers of every subtree neighboring its own trunk nodes. We distinguish two types of these links: *random* branch links and *local* branch links. The former links point to truly random peers of a subtree. With each of these links the id and weight of the subtree as well as the address of the corresponding peer is stored. Moreover, each link is tagged with a timestamp and a link is replaced

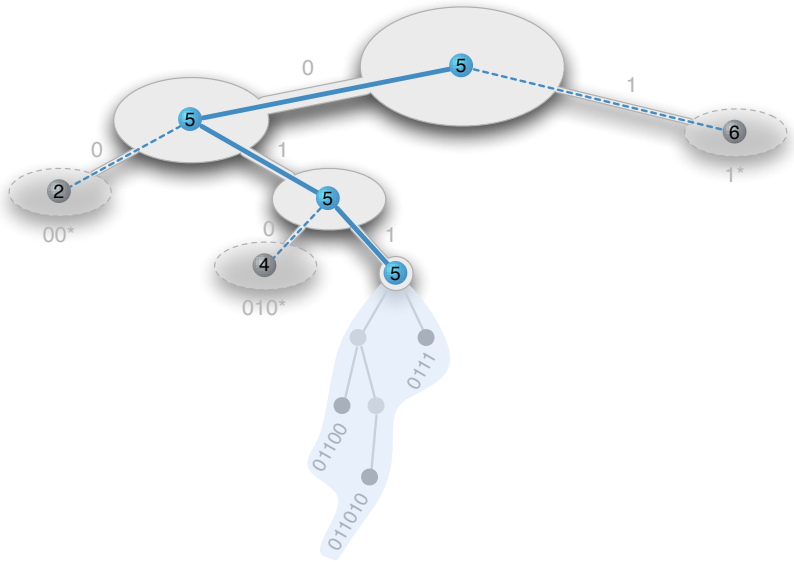


Figure 7.5: *The local view of peer* 5*. Branch links are depicted by dashed lines.*

whenever a peer participating in the subtree is met during Pointer-Push&Pull operations. The latter guarantees that branch links point to truly random peers. Figure 7.5 shows the local view with trunk nodes and branch links of peer 5 (cf. Figure 7.2 for the global view).

Local branch links are similar to random branch links with the exception that these do not point to random peers of a subtree but to latency wise close peers of a subtree. Again, Pointer-Push&Pull operations ensure that a peer sees all possible candidates for a particular local branch link over time and that the quality of these links improves quickly.

7.1.5 Initializing the Local View

To join the network, a peer contacts an arbitrary peer p that is already part of the network and then proceeds as described in Algorithm 7.1. The joining peer will copy p's trunk node table for the root of the network tree and then, based on the subtrees and weights given in the trunk node table, choose a subtree using the DHHT scheme. Using the list of branch links it is ensured that a peer p' that has chosen the same subtree can be contacted. Then, the same procedure is continued in the root node of the chosen subtree with peer p'. The algorithm terminates when either the joining peer is the only one assigned to a subtree or a leaf of the data tree has been reached.

The subtrees chosen in line 4 of the join algorithm heavily depend on the correctness of the branch link list and the weights obtained from the copied trunk node tables.

Algorithm 7.1 Join(peer p)

1: initialize trunk node table for root node by copying p's trunk node table
2: *node* $\leftarrow$ root of tree
3: **while** number of peers in *node* > 1 and *node* is not leaf of data tree **do**
4: *node* $\leftarrow$ subtree determined using DHHT
5: $p' \leftarrow$ peer participating in *node* found using branch link table
6: contact p' in *node* and initialize trunk node table for *node* by copying table of p'
7: **end while**

Hence, it is crucial that all peers have consistent and accurate information about the structure and weight of the tree. Before we describe how this information is exchanged among peers let us recall the definition of weight used here. The weight $w(v)$ of a node v of the network tree is given by the number of data items it stores.[1] The only peer allowed to set $w(v)$ is the peer responsible for v. The weight $w(T_v)$ of the subtree T_v rooted at v is determined by summing up $w(v)$ and the weights of the subtrees rooted at v's child nodes.

7.1.6 Maintaining the Local View

Peers exchange information about tree structure and weights in their trunk nodes' random networks. This is where Pointer-Push&Pull operations play an important role: Whenever two peers communicate during a Pointer-Push&Pull operation their responsibility list, branch link list, and weights of the node and subtrees is piggy-backed to the messages. A peer p that has communicated with a peer p' uses the obtained information to update its own trunk node table as follows:

- The weight of the node is updated. As mentioned above, the only peer allowed to set the weight of a node is the responsible peer. Since the weight of a node may change over time each weight distributed by the responsible peer is tagged with a timestamp. Thus, peers will only update this entry in the trunk node table if they receive a weight with newer timestamp.
- Entries of the branch link list corresponding to subtrees peer p' actively participates in (i.e. subtrees p' has trunk nodes in) are set to point to p'. This way the entries of the branch link list are continuously replaced by peers which are ensured to be reachable. Furthermore, entries for subtrees not existent in the own branch link list are copied and entries that have been identified to be dead during Pointer-Push&Pull operations or routing are replaced. Most importantly the combination of update procedure and Pointer-Push&Pull operation guarantees that branch links point to truly random peers and that over time all peers participating in the trunk node are contacted.

[1] Note that also internal nodes of the data tree may have load > 0, since we allow to store data on internal nodes.

- Entries of the responsibility list are replaced when peers with higher responsibility are found in the list of p' or added if their own list has less than r entries.

When changes to the weights or branch link list have been made, a peer recalculates its own assignment to the subtrees and may change its path, i.e. trunk nodes in the network tree, when necessary.

The exchange of information using Pointer-Push&Pull operations is closely related to randomized rumor spreading using push and pull operations introduced by Karp et al. [62]. A major difference making a formal analysis difficult is, that in our case the underlying communication network is not a complete graph and furthermore changes over time. Yet, if we assume the communication graph to be random we expect the dissemination of information by Pointer-Push&Pull operations to behave comparably as in [62], where $\mathcal{O}(n \ln \ln n)$ messages are sufficient to spread a rumor among n nodes w.h.p.

7.1.7 Routing

A lookup algorithm for the 3nuts network is given by Algorithm 7.2. The lookup is started at an arbitrary peer p and the only parameter is the identifier *key* of a data element. Since the data tree is a prefix tree defined by the data elements currently available in the network, the identifier *key* actually describes a path in the network tree. To reach the node of the network tree storing *key*, peer p follows the path *key* in its local view of the network tree until a leaf node is reached. This leaf node can either be a branch link or a trunk node in p's local view. In the former case the lookup is forwarded to the corresponding peer in the branch link list. In the latter case the lookup is forwarded to the peer responsible for the trunk node. Since number of data elements is typically considerably greater than the number of peers and most data elements reside in the leaves of the tree, it is very likely that the responsible peer is reached directly.

Algorithm 7.2 Lookup(*key*) at peer p

```
1: if p has branch link to a peer p' sharing longer prefix with key then
2:     forward Lookup( key ) to p'
3: else
4:     node ← last node of path key in local tree of p
5:     p' ← peer responsible for node
6:     if p = p' then
7:         return p
8:     else
9:         forward Lookup( key ) to p'
10:    end if
11: end if
```

The following theorem gives a bound for the number of hops needed by the lookup operation to reach the peer responsible for a particular data element.

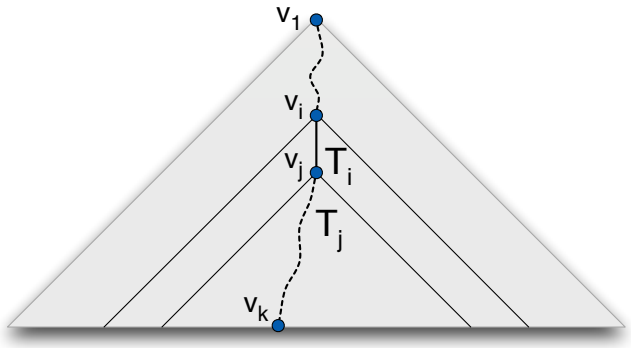


Figure 7.6: *Partitioning of the network tree into subtrees as used in Theorem 7.2. The path of the lookup is given by $P = (v_1, \ldots, v_i, v_j, \ldots, v_k)$. T_i is the smallest subtree rooted on P with $|T_i| \geq \frac{n}{2}$.*

Theorem 7.2 *In a 3nuts network with n peers the number of hops for a lookup operation is bounded by $\mathcal{O}(\log n)$ w.h.p., regardless of the structure of the data tree.*

Proof: To proof the theorem we will bound the number of hops needed to reach a subtree T containing at most half of the peers. Let $P = (v_1, \ldots, v_i, v_j, \ldots, v_k)$ be the path starting at the root of the network tree leading to the target node of the network tree. Furthermore let T_i and T_j be the subtrees rooted at v_i respectively v_j. We choose v_i and v_j such that $|T_i| \geq \frac{n}{2}$ and $|T_j| \leq \frac{n}{2}$. In other words: T_i is the smallest subtree rooted on P containing at least half of the peers.

The lookup starts at an arbitrary peer p in v_1, i.e. the root of the network tree. Let p' be the peer reached by the first hop. Since $|T_i| \geq \frac{n}{2}$ and branch links point to truly random peers, p' will lie in T_i with probability of at least $\frac{1}{2}$. If, on the other hand, p' does not lie in T_i, the same argumentation holds for the next hop from p', i.e.: the next hop from p' will lead to a peer in T_i with probability of at least $\frac{1}{2}$. So, we reach T_i with one hop with probability $\geq \frac{1}{2}$, with two hops with probability $\geq 2^{-2}$, and with k hops with probability $\geq 2^{-k}$. Thus, we have

$$E[\#\text{hops to reach peer in } T_i] \leq \sum_{k=1}^{k=\frac{n}{2}-1} k2^{-k} \leq 2\,.$$

Since T_i is the *smallest* subtree with $|T_i| \geq n/2$, once the lookup reached a node in T_i one more hop is sufficient to reach the subtree T_j with $|T_j| \leq \frac{n}{2}$. Therefore, in expectation at most 3 hops are needed to halve the number of peers. Due to the recursive structure of 3nuts the same line of arguments as presented above does hold for subtree T_j. This implies that after $\log n$ iterations respectively an expected number of $3 \log n$ hops, the lookup has reached v_k, i.e. a single peer.

It remains to show that $\mathcal{O}(\log n)$ hops are sufficient to reach v_k with high probability. We have seen that

$$\Pr[\#\text{peers is halved within three hops}] \geq \frac{1}{2} + \frac{1}{4} = \frac{3}{4} .$$

Dividing the lookup into sequences of three hops allows us to reduce the analysis to a sequence of mutually independent random variables $X_1, X_2, \ldots, X_{c \log n}$ taking values 0 and 1 with $\Pr[X_i = 1] = \frac{3}{4}$ respectively, and $X = \sum_{i=1}^{c \log n} X_i$. Thus we can apply Chernoff bounds and show that for $c \geq 5$ we have $X > \log n$ with high probability, i.e. the target peer is reached after $3c \log n$ hops with high probability. The expected number of successful steps is given by

$$E[X] = \frac{3}{4} c \log n .$$

Choosing $\delta = 1 - \frac{4}{3c}$ and applying Chernoff bounds we have

$$\begin{aligned} \Pr[X \leq \log n] = \Pr[X \leq (1-\delta)E[X]] &\leq e^{-\frac{1}{2}\left(1-\frac{4}{3c}\right)^2 E[X]} \\ &\leq e^{-\frac{1}{2}\left(\frac{11}{15}\right)^2 \frac{3}{4} c \log n} \\ &\leq n^{-\frac{121}{600} c} \\ &\leq n^{-c'} . \end{aligned}$$

So, the probability to be successful less than $\log n$ times is polynomially small in n. This implies that the lookup operation will need at most $3c \log n$ hops w.h.p. if we choose $c \geq 5$. Since we did not make any assumptions on the structure of the network tree, this result does also hold for degenerated trees with linear depth. ■

It is notably and important that the bound given by Theorem 7.2 holds regardless of the structure of the data tree since real world data will not be uniformly distributed, but rather resemble a Zipf distribution. So, the data tree (and thus the network tree) will be unbalanced and exceed logarithmic depth. The reason that Theorem 7.2 holds for skewed data distributions is that branch links point to random peers of subtrees. Due to the properties of the Pointer-Push&Pull operation (see Theorem 4.2) and the way branch links are maintained (see Section 7.1.4), we can guarantee these to be truly random. Actually branch links are not only random but continually change (recall that whenever a peer communicates with another peer during a Pointer-Push&Pull operation that has been assigned to a subtree T neighboring its own path of trunk nodes the corresponding branch link will be updated). This feature implies that routing path are continually changing when random branch links are used for routing and thus the routing load will be spread evenly among peers. A typical measure with respect to routing load is the congestion. We formally define congestion as follows.

Definition 7.1 (Congestion) *The congestion of a peer p_i is the probability that p_i is active in the routing of a random lookup operation started at a random peer. The congestion of the network is the maximum congestion over all its peers.*

Assuming that the network is in a stable state and branch links are truly random we can bound the congestion of the lookup operation in 3nuts as follows.

Theorem 7.3 *The congestion of the 3nuts network is bounded by* $\mathcal{O}\left(\frac{\log n}{n}\right)$.

Proof: From Theorem 7.2 we know that the number of hops is bounded by $k \leq c \log n$ for constant c. Let $T_1, \ldots, T_k$ denote the subtrees reached during the k hops of the lookup operation and by $|T_i|$ we denote the number of peers that has been assigned to subtree T_i, $1 \leq i \leq k$. Now consider an arbitrary peer p_j. Note that p_j can only get active once during a lookup. To become active during the i-th hop peer p_j must have been assigned to T_i and must have been chosen as branch link by the peer reached by the previous hop. Since the assignment of peers to subtrees is done with DHHTs and $|T_i|$ peers have been assigned to T_i, the probability that peer p_j has been assigned to T_i is $|T_i|/n$. If the network is in a stable state, i.e. branch links point to truly random peers, the probability for a peer assigned to T_i to become active during the i-th hop is $1/|T_i|$. Thus we have

$$\Pr[p_j \text{ is active during hop } i] = \frac{|T_i|}{n}\frac{1}{|T_i|} = \frac{1}{n}.$$

Consequently the probability for p_j to become active during the whole lookup is given by $\frac{c \log n}{n}$.

■

Range Queries

Algorithm 7.2 can be extended to perform range queries easily. To search for all data elements in a range $[x, y]$ the peer initiating the query calculates the longest common prefix z of x and y. Then, the query is routed to the node v of the network tree representing z. The subtree rooted at v is the smallest subtree containing all elements in the range $[x, y]$. Starting from v, the query is forwarded to all child nodes holding data in the range $[x, y]$ in parallel until the leaf nodes of the network tree are reached.[2] Nodes receiving the query message will send their list of data elements to the peer that originated the query respectively forward the query to the responsible peer in case of internal nodes of the network tree.

7.2 Locality in 3nuts

Since the peers of a 3nuts network resemble the prefix tree defined by the data available in the network, closely related data elements are stored on network-wise close peers. In particular, the following theorem holds.

Theorem 7.4 *Let d be the distance of two data elements x and y in the tree metric. Then, x and y can be reached within d hops from one another in 3nuts.*

[2] Note that the border area of the subtree rooted at v may contain data elements not lying in $[x, y]$.

Proof: Let z denote the node of the network tree representing the longest common prefix of x and y. Note that z is present in the local tree of the peer responsible for x since z lies on the path starting at the root of the tree leading to x. So, no hops are needed to reach node z. From node z it is ensured that y can be reached within d hops since each hop will advance at least one level in the tree and the distance between z to y is bounded by d. The same line of arguments holds when routing from y to x. Recall that in any case the maximum hop distance is bounded by $\mathcal{O}(\log n)$ w.h.p., with n denoting the number of peers. ■

Recalling the three types of locality introduced in Section 6.1, Theorem 7.4 implies that 3nuts provides information locality and in the previous section we have already seen how range queries can be processed efficiently in 3nuts.

3nuts provides network locality, i.e. lookups with small latency, through the list of local branch links maintained in the trunk node tables. As discussed in Section 7.1.4 local branch links point to latency wise close peers of the corresponding subtrees and may be used for routing instead of random branch links. Initially the local branch link list is just a copy of the random branch link list. Latencies are measured during Pointer-Push&Pull operations and whenever a peer with lower latency is met, it will be saved in the local branch link list. Since a peer will see every other peer of the trunk node's corresponding random graph over time — and therefore will see all potential candidates for a local branch link — it is ensured that at some point the entries of the local branch link list point to the latency wise closest peers. Once again note that no additional network traffic is generated to improve the quality of the local branch link lists since latencies are obtained through Pointer-Push&Pull operations. In Section 7.3 we will experimentally verify the quality of local branch links and compare latencies when routing with random branch links and local branch links.

3nuts provides interest locality by allowing peers to *volunteer* for the responsibility of particular nodes of the data tree. Note that volunteering for the responsibility of a node does not relieve a peer from participating in the regular peer assignment using DHHTs described in Section 7.1.2, i.e. volunteering is completely independent of the regular peer assignment and therefore induces additional workload to a peer. On the other hand volunteering for the responsibility of a node allows a peer to dramatically decrease access times to parts of the data tree that are close to this node.

A peer p volunteering for a node v will actively participate in the path starting at the root of the data tree leading to v. For nodes of this path that are not coincident with the regular path p has been assigned to, p will have to maintain additional trunk nodes. Since the decision about responsibility is made using heights in the DHHT scheme (see Section 7.1.2), it is possible that p has to take over responsibility for internal nodes of the path in the network tree leading to node v. To avoid this, we make use of two special flags. The *volunteer* flag indicates that a peer volunteers to be responsible for this node and the *volunteer_down* flag indicates that a peer volunteers for a node further down the tree. The latter flag prevents a volunteering peer to be responsible for nodes on the path to v, i.e. a peer with *volunteer_down* flag always has lower responsibility than peers regularly assigned to a node. In the node that a peer is volunteering for, it sets

the *volunteer* flag and therefore always has higher responsibility than peers that were assigned regularly to the node. In case of several peers volunteering for the same node the heights determined in the DHHT scheme are used to determine responsibility.

7.3 Experimental Evaluation

To verify the robustness and practicability of 3nuts we used a prototype implementation in Java, which is available for download at `http://3nuts.upb.de`. All experimental results presented in this section have been generated using this prototype. For the experiments the degree of random networks was set to $d = 3$ and if not stated otherwise, the network consisted of $n = 2^{14}$ peers. Each peer stored five data elements giving a total of $81,920$ data elements available in the network. Most of the measurements have been performed with several types of data to verify the impacts of data distribution and degree of the data tree respectively network tree. These are:

Binary tree (uniform) A binary tree with data elements representing binary strings of length 40. The strings are chosen uniformly at random.

Binary tree (Zipf) A binary tree of data elements representing binary strings of length 52, chosen according to a Zipf distribution as follows. The leaf nodes of a complete binary tree of depth 20 have been assigned probabilities following a Zipf distribution with exponent set to 1. Data elements are placed by choosing a prefix of length 20 according to the assigned probabilities and concatenating a (unique) binary random string of length 32. So, some subtrees of the resulting data tree contain by far more data elements than others.

Dictionary A tree generated by choosing data elements uniformly at random from a list of english words (we used the freely available word list of the ispell [55]). In contrast to the binary trees, nodes of the dictionary tree have degree up to 26.

If not stated otherwise, the following measurements have been performed multiple times and the curves represent the mean of these measurements.

7.3.1 Routing

Figure 7.7 shows the average number of hops needed by the lookup operation for different data distributions and networks up to 2^{14} peers. A single measurement for a fixed network size and data distribution was done by performing 10^6 random lookup operations chosen from all possible combinations of peers and data elements. Here, the two curves representing the binary trees are almost equal, implying that the DHHT based load balancing performs excellently and the scalability of 3nuts is not affected by non-uniform data distributions. In case of the dictionary data distribution fewer hops are needed since data and network tree have substantially higher degree and thus the

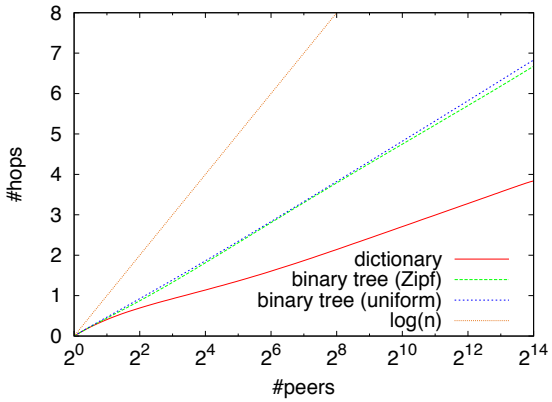


Figure 7.7: *Average number of hops needed by the lookup operation.*

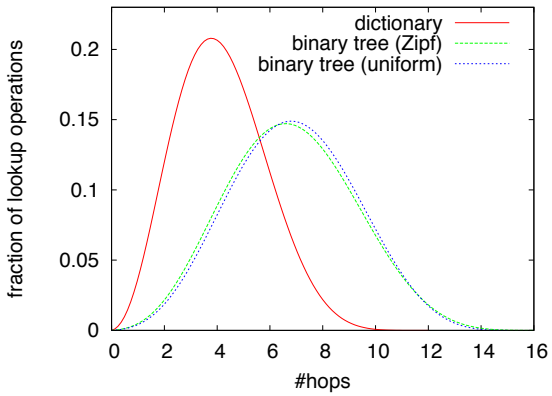


Figure 7.8: *Distribution of number of hops needed by the lookup operation.*

depth of the network tree is smaller than in case of the binary trees. All curves lie beneath the $\log n$ curve showing that the performance of the lookup operation is better than the performance formally proven in Theorem 7.2.

In Figure 7.8 the hop distribution measured in the network with 2^{14} peers is shown. The number of hops can vary between 0 (if the peer initiating the lookup itself is responsible for the data item) and the depth of the network tree. The latter is a merely theoretical bound since it is very unlikely to advance only one level of the tree with each hop of the lookup. However, greater depth of the data tree leads to higher variance of the hop distances. Nevertheless, the maximum hop distance measured in case of the binary trees is still bounded by $\log n$.

To evaluate the benefit of using local branch links instead of random branch links we used the Georgia Tech Internetwork Topology Model (GT-ITM) [122] to model the underlying physical network of the 3nuts overlay. Figure 7.9 shows the average latencies for different data distributions when using random respectively local branch links during the lookup operation. First of all, we observe that the average latency measured for routing with random links in binary trees exceeds the average latency measured for routing with random links in the dictionary tree by a factor of 1.8. This is explained by the larger number of hops needed in the comparatively deep binary trees. When using local branch links, latencies are reduced significantly for all three types of data trees. Notably, average latencies measured for dictionary and binary trees then only differ by a factor of 1.1, i.e. the binary trees benefit more from the use of local links. This is explained by the fact that the last few hops are by far the most "expensive" ones during a lookup operation since the number of peers to choose from decreases with each hop, i.e. it is more unlikely that latency wise close peers are among the peers to choose from. Thus, the last hop makes up a large fraction of the total latency.

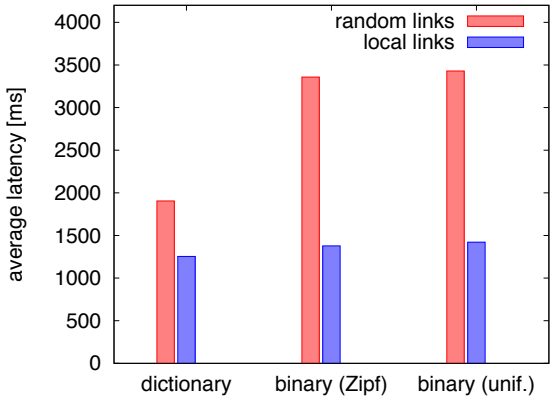


Figure 7.9: *Average latency of the lookup operation: random branch links vs. local branch links.*

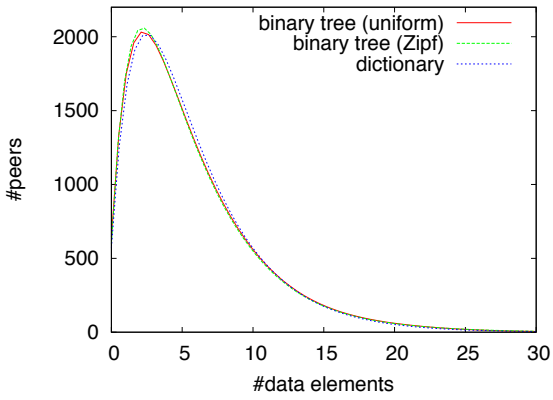


Figure 7.10: *Load balancing: uniform distributed data, Zipf distributed data, and dictionary.*

7.3.2 Load Balancing

Figure 7.10 shows the distribution of data load among peers. Each peer contributes 5 data elements and thus the average load is as well 5. Since the DHHT based load balancing makes use of pseudo-random hash functions there are some peers exceeding the average load by factor 5. Anyhow, 90% of the peers have at most twice the average load. While having peers exceeding the average load by factor 5 is for sure not optimal, one has to recall the simplicity and elegance of the load balancing scheme used here. It is in particular remarkable that the decisions a peer makes when choosing its path are completely independent of the decisions made by other peers. So, a peer usually does not have to change its path, respectively responsibility, when further peers enter the network. Coping with not 100% fair load balancing is the price to pay for this simplicity. One should not forget that this simplicity may turn out to be crucial to keep the network stable under churn. Furthermore, a peer noticing that its load exceeds the average by a large factor still has the option to leave and rejoin the network. Due to the randomized nature of the load balancing scheme used here, it will most likely be assigned to a different part of the tree.

7.3.3 Degree

Another interesting measure is the number of links, i.e. neighbors, a peer has to maintain. Figure 7.11 shows the sum of branch links and neighbors in the random networks per peer, while Figure 7.12 shows the number of branch links per peer only. Of course the degree of a peer depends on the degree and the depth of the data tree: high degree of the data tree involves a large number of branch links and higher depth of the data tree involves a larger number of trunk nodes and thus random networks, a peer is

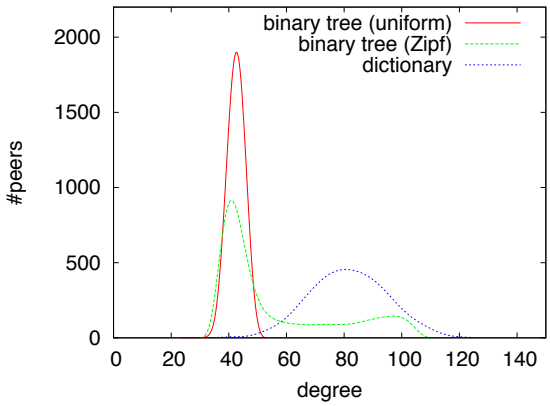


Figure 7.11: *Degree distribution: number of branch links and random neighbors per peer.*

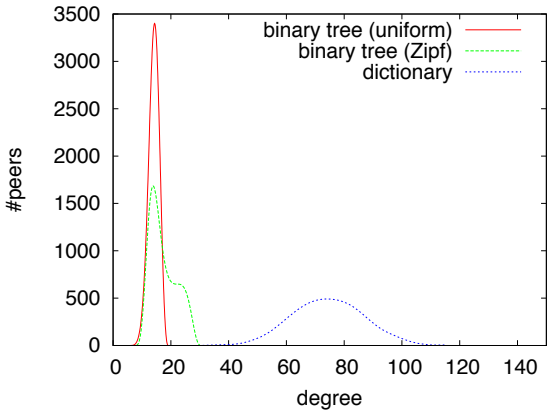


Figure 7.12: *Degree distribution: number of branch links per peer.*

participating in. We start by analyzing the uniformly distributed binary tree. Considering the number of branch links only (Figure 7.12), a peer's degree is comparable to the degree of a peer in a standard DHT based network like Chord [114], i.e. in case of the uniformly distributed binary tree the average degree is 14.5, i.e. around $\log n$, and the expected degree in a plain Chord network consisting of 2^{14} peers is 15. Taking links in random networks into account (Figure 7.11), the average degree increases to 42 since a peer has to maintain three additional links for every trunk node. Note however, that especially the links in the random networks are extremely easy to maintain, as pointed out in Chapter 4 already.

Zipf distributed data leads to slightly increased average degree compared to the uniformly distributed binary tree. This is explained by the fact that data and network tree have greater depth. In case of the dictionary tree the high fan out of the data tree imposes a large number of branch links to the peers. A solution to reduce the number of branch links here is to convert the tree into a binary tree, i.e. interpret the characters of data elements bit wise instead of byte wise. Furthermore, the depth of the tree and thus the average peer degree can be reduced by using radix trees [48] instead of a simple prefix tree. A radix tree, also known as Patricia trie/tree, or crit bit tree, is a specialized form of prefix tree. In contrast to a regular prefix tree, the edges of a radix tree are labeled with sequences of characters rather than with single characters. This allows to reduce the depth of the tree if sub-paths without branches exist in the tree, which is likely when using bit wise interpretation of characters.

Considering the additional features 3nuts provides compared to DHT based networks, e.g. information locality and network locality, the "costs" for these features, i.e. the increased degree, are negligible. Moreover, links in 3nuts are very easy to maintain: While links in most peer-to-peer networks have to point to one particular peer,

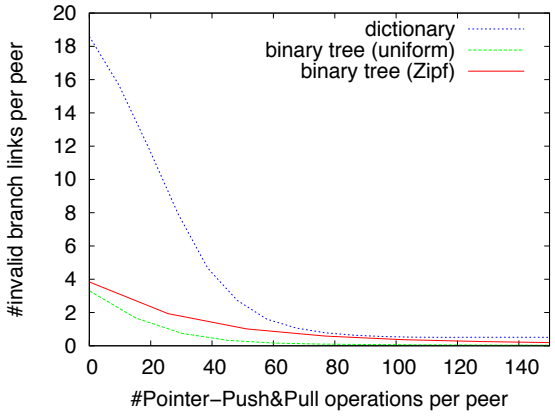


Figure 7.13: *Evolution of invalid branch links after failure of* 25% *of peers.*

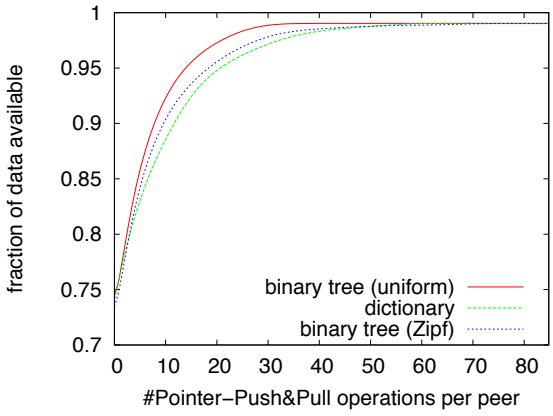


Figure 7.14: *Evolution of data availability after failure of* 25% *of peers.*

a link in 3nuts may always point to a random peer out of a large set of candidates.[3] This is especially true for the random networks, where even multi-edges are allowed so that a failing neighbor can simply be replaced by duplicating another link — Pointer-Push&Pull operations ensure the network structure to be re-randomized after a short time only.

7.3.4 Dynamics and Robustness

We also verified the robustness of 3nuts when a substantial fraction of the peers fails unexpectedly. A network of 10^4 peers was generated and once the network stabilized, 25% of the peers were removed simultaneously. Figure 7.13 shows the evolution of the average number of invalid random branch links per peer (local branch links were neglected here). Due to the higher fan-out, the number of branch links is significantly higher in case of the dictionary tree and thus the number of invalid links is also higher compared to the binary trees. The axis of abscissae shows the total number of Pointer-Push&Pull operations per peer, i.e. cumulative for all random networks a peer participates in. In case of the binary trees about 20 random Pointer-Push&Pull operations are sufficient to reduce the number of invalid links per peer to 2 while this takes about 50 operations in case of the dictionary tree. Here, the comparably strong decrease during the first 50 operations is explained by the randomness of Pointer-Push&Pull operations and the larger number of invalid links, i.e. links are "checked" randomly and the larger the number of invalid links, the higher the probability to find these. Note however that the dictionary tree may be converted to a binary tree as mentioned above. Furthermore, when encountering an invalid branch link during a lookup operation, the lookup can simply be forwarded to a random neighbor in the trunk node. Since branch links are

[3] With exception of the lowest levels of the network tree.

ensured to be random, the branch link list of this neighbor most likely has a different link to the designated subtree.

Figure 7.14 shows the evolution of data availability in the same scenario. Data availability was checked by performing 10^6 random lookup operations respectively. Here, a lookup was considered as failed whenever an invalid branch link was encountered and furthermore the repair process — usually initiated when invalid links are encountered during a lookup — was disabled. In other words: Trunk node tables were only updated/repaired by Pointer-Push&Pull operations. The different data distributions, respectively tree types, behave almost equal. Right after the removal of peers about 75% of the data remaining in the network is still available and after 30 to 50 Pointer-Push&Pull operations per peer, data availability of 99% is reached. We stopped the experiment at 99% availability since the removal of $2,500$ peers and thus $12,500$ data elements also involves changes in the network tree and some peers change their path, etc.

8

Conclusion and Outlook

The 3nuts peer-to-peer network introduced in the second part of this thesis combines unstructured and structured peer-to-peer networking concepts, i.e. random networks, prefix trees, and distributed hash tables, to overcome their individual shortcomings. To the best of our knowledge 3nuts is the first peer-to-peer network providing interest locality, network locality, and information locality at the same time. The practicability of 3nuts has been affirmed by a prototypical implementation ready for practical use, experimental evaluation, and verification on a mathematical level.

The 3nuts architecture has been designed around the Pointer-Push&Pull operation, whose properties make it an excellent maintenance operation for dynamic networks. Replacing the heartbeat (ping) messages between peers, Pointer-Push&Pull operations are used in 3nuts to:

- maintain truly random networks to replace nodes of the data tree and thus make the network robust,
- exchange information among peers and thus give peers a coherent view of the tree structure,
- maintain branch links and guarantee them to be truly random, thus allow efficient routing,
- and measure round trip times (RTT) to adapt the overlay to the underlying physical network and thus allow routing with small latency.

A possible drawback of the 3nuts architecture is the potentially high degree of the network, which can be caused by highly skewed data distributions resulting in deep paths. However, as pointed out in Section 7.3.3 one should keep in mind that maintenance of links is comparably cheap in 3nuts and higher degree also implies higher robustness. If necessary, the degree can possibly be reduced by making use of radix trees or balanced trees instead of simple prefix trees.

Outlook

While making extensive use of DHTs for load balancing, 3nuts is able to overcome their restriction to exact match queries. However, range queries can still be viewed as a restriction when considering the types of queries possible in unstructured peer-to-peer networks. Finding a way to support queries with regular expressions, e.g. search for substrings, in a scalable way would constitute a significant advance.

Considering network locality, i.e. the interplay between the overlay network and the physical network, there is vast number of possibilities to improve this interplay. First of all, while the local branch links used in 3nuts allow to reduce latencies during routing, it is not clear if the chosen routes are optimal since the local branch link of each hop is a local optimum only. Furthermore, it may be worthwhile to consider other optimization criteria than the latency of a lookup, e.g.:

Failure resilience The neighbors of a peer could be chosen such that the paths leading to these in the physical network are node disjoint as far as possible.

High throughput The topology of the overlay could be constructed such that the available bandwidth of connections is taken into account and the congestion induced to the physical network by the overlay is minimized.

Traffic costs From a network providers perspective it is desirable to reduce interdomain traffic for the simple reason of reducing costs. The overlay could be constructed such that routes preferably stay in the same autonomous system or the number of autonomous systems crossed by a route is minimized.

It can also make sense that a network allows to choose among different optimization criteria if the networks provides different types of service, e.g. lookup messages in a network could be performed on routes with low latency and transfer of large files on a different route with worse latency but higher bandwidth.

Pointer-Push&Pull operations constitute an interesting tool to explore the physical network — with local handshake operations only and very little network traffic — since a peer will meet every other peer of the network over time. Certainly, it has to be examined which information about the physical network can be obtained by a peer, at which cost, and how useful the obtained information is at all.

A

Pointer-Push and Pointer-Pull Operations

The Pointer-Push operation is a simple graph transformation defined on a path of two edges applicable to multi-digraphs.

Definition A.1 (Pointer-Push Operation) *Consider a multi-digraph $G = (V, E, \#_E)$ and nodes v_1, v_2, $v_3 \in V$ forming a directed path $P = (v_1, v_2, v_3)$ in G. Then, the Pointer-Push operation $PUSH_P$ transforms graph G to a graph $PUSH_P(G) = (V, E', \#_{E'})$ with*

$$E' = (E \setminus (v_1, v_2)) \cup (v_1, v_3).$$

Figure 4.1 illustrates the Pointer-Push operation. A randomized version of the Pointer-Push operation is given by Algorithm A.1.

Algorithm A.1 Random Pointer-Push

1: Choose random node $v_1 \in V$
2: $v_2 \leftarrow$ random node $\in N^+(v_1)$
3: $v_3 \leftarrow$ random node $\in N^+(v_2)$
4: $E \leftarrow (E \setminus (v_1, v_2)) \cup (v_1, v_3)$

The following Lemma shows that the Pointer-Push operation is sound.

Lemma A.1 *Applying random Pointer-Push operations to a connected multi-digraph G will preserve connectivity of G. Furthermore the outdegree of each node in G will stay the same.*

Proof: Concerning the outdegree note, that one of the outgoing edges of v_1 is deleted and one new edge starting at v_1 is created. Connectivity is preserved because all participating nodes of a Pointer-Push operation stay connected. However, the resulting graph may be weakly connected. ■

For the asymptotic behavior of the random Pointer-Push operation the following theorem holds.

Theorem A.1 *A series of random Pointer-Push operations will transform any connected multi-digraph into a connected set of stars in the limit with probability* 1.

Proof: When applying random Pointer-Push operations there is a non-zero probability that a node creates self edges. Eventually, all outgoing edges of a node will point to itself and therefore this node will be a sink. Every time the second node of a Pointer-Push operation is such a sink $s \in V$, the third node will also be s. Therefore, the random Pointer-Push operation will create edges pointing to the sink what in turn further increases the probability of random Pointer-Push operations to end in a sink. Note, that there is no way to remove edges pointing to a sink. So, in the long run the graph will consist of at least one sink and all other nodes pointing directly to sinks. ■

Intuitively, nodes with higher indegree are prone to receive even higher indegree. The above theorem shows, that this causes the Pointer-Push operation to not provide generality.

We can overcome the problem of further increasing the indegree of nodes which already have high indegree with the following graph transformation, called Pointer-Pull operation.

Definition A.2 (Pointer-Pull Operation) *Consider a multi-digraph $G = (V, E, \#_E)$ and nodes v_1, v_2, v_3, $v_4 \in V$ forming a directed path $P = (v_1, v_2, v_3, v_4)$ in G. Then, the Pointer-Pull operation $PULL_P$ transforms graph G to a graph $PULL_P(G) = (V, E', \#'_E)$ with*

$$E' = (E \setminus (v_3, v_4)) \cup (v_3, v_1).$$

The Pointer-Pull operation is illustrated in Figure 4.2 and a randomized version is given by Algorithm A.2.

Algorithm A.2 Random Pointer-Pull

1: Choose random node $v_1 \in V$
2: $v_2 \leftarrow$ random node $\in N^+(v_1)$
3: $v_3 \leftarrow$ random node $\in N^+(v_2)$
4: $v_4 \leftarrow$ random node $\in N^+(v_3)$
5: $E \leftarrow (E \setminus (v_3, v_4)) \cup (v_3, v_1)$

As in case of the Pointer-Push operation, the Pointer-Pull operation does not change the outdegree of any node. The intuition, in contrast to the Pointer-Push operation, is that applying random Pointer-Pull operations may balance the indegree of the nodes. This is because the starting node v_1 of each operation will increase its indegree by one and v_1 is chosen uniformly at random. Furthermore, nodes with high indegree have a higher probability to be endpoint of a Pointer-Pull operation and therefore, higher probability to get their indegree decreased. Even if this is the case, the following theorem shows a major drawback of the Pointer-Pull operation.

Theorem A.2 *Starting with an arbitrary multi-digraph G with n nodes, random Pointer-Pull operations disconnect G into n components of single nodes with slopes in the limit.*

Proof: From every digraph, this terminal graph is reachable by a series of random Pointer-Pull operations. Furthermore, applying random Pointer-Pull operations to the

terminal graph will transform the terminal graph to itself. So, in the limit the Markov chain described by the random Pointer-Pull operation converges to this terminal graph. ■

Bibliography

[1] Karl Aberer. P-Grid: A self-organizing access structure for p2p information systems. In *CoopIS '01: Proceedings of the Sixth International Conference on Cooperative Information Systems*, 2001.

[2] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-Grid: a self-organizing structured P2P system. *SIGMOD Record*, 32(3):29–33, 2003.

[3] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. P-Grid: Dynamics of self-organizing processes in structured p2p systems. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science (LNCS)*. Springer Verlag, 2005.

[4] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[5] Noga Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory. Series B*, 38(1):73–88, 1985.

[6] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[7] James Aspnes and Gauri Shah. Skip graphs. In *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[8] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37, November 2007.

[9] Greg Barnes and Uriel Feige. Short random walks on graphs. In *STOC '93: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 728–737. ACM Press, 1993.

[10] Salman Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *CoRR*, abs/cs/0412017, 2004.

[11] Sonny Ben-Shimon and Michael Krivelevich. Vertex percolation on expander graphs. *European Journal of Combinatorics*, 30(2):339 – 350, 2009.

[12] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[13] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[14] Manuel Blum, Richard M. Karp, Oliver Vornberger, Christos H. Papadimitriou, and Mihalis Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letters*, 13(4/5):164–167, 1981.

[15] Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1:311–316, 1980.

[16] Béla Bollobás. Random graphs. *Combinatorics*, 52:80–102, 1981.

[17] Béla Bollobás. The isoperimetric number of a random graph. *European Journal of Combinatorics*, 9:241–244, 1988.

[18] Sherif M. Botros and Steve R. Waterhouse. Search in jxta and other distributed networks. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P 2001)*, pages 30–35, 2001.

[19] V. Bourassa and F. Holt. Swan: Small-world wide area networks. In *Proceedings of the International Conference on Advances in Infrastructure (SSGRR 2003w)*, 2003.

[20] Peter Buser. A note on the isoperimetric constant. *Annales Scientifiques de l'École Normale Supérieure. Quatrième Série*, 15(2):213–230, 1982.

[21] Hyunseok Chang, Sugih Jamin, and Wenjie Wang. Live streaming performance of the zattoo network. In *IMC '09: Proceedings of the Ninth ACM SIGCOMM Internet Measurement Conference*, pages 417–429, New York, NY, USA, 2009. ACM.

[22] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[23] Jeff Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. In *Problems in analysis (Papers dedicated to Salomon Bochner, 1969)*, pages 195–199. Princeton University Press, Princeton, N. J., 1970.

[24] Fan R. K. Chung. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[25] JXTA Community. JXTA — company spotlight archive. `https://jxta.dev.java.net/companyarchive.htm`, 2007.

[26] Colin Cooper, Martin Dyer, and Catherine Greenhill. Sampling regular graphs and a peer-to-peer network. In *SODA '05: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 980–988, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[27] Colin Cooper, Martin Dyer, and Catherine Greenhill. Sampling regular graphs and a peer-to-peer network. *Combinatorics, Probability and Computing*, 16(4):557–593, 2007.

[28] Colin Cooper, Martin Dyer, and Andrew J. Handley. The flip markov chain and a randomising p2p protocol. In *PODC '09: Proceedings of the Twenty-Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 141–150, New York, NY, USA, 2009. ACM Press.

[29] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using P-trees. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 25–30, New York, NY, USA, 2004. ACM Press.

[30] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a dht for low latency and high throughput. In *NSDI '04: Proceedings of the First Symposium on Networked Systems Design and Implementation*, pages 85–98, March 2004.

[31] Persi Diaconis and Laurent Saloff-Coste. Comparison theorems for reversible markov chains. *The Annals of Applied Probability*, 3(3):696–730, 1993.

[32] Jozef Dodziuk. Difference equations, isoperimetric inequality and transience of certain random walks. *Transactions of the American Mathematical Society*, 284(2):787–794, 1984.

[33] Cedric du Mouza, Witold Litwin, and Philippe Rigaux. SD-Rtree: A scalable distributed Rtree. In *ICDE '07: Proceedings of the Twenty-Third International Conference on Data Engineering*, pages 296–305, April 2007.

[34] Martin Dyer, Leslie Ann Goldberg, Mark Jerrum, and Russel Martin. Markov chain comparison. *Probability Surveys*, 3:89–111, 2006.

[35] Tomas Feder, Adam Guetz, Milena Mihail, and Amin Saberi. A local switch markov chain on given degree graphs with application in connectivity of peer-to-peer networks. In *FOCS'06: Proceedings of the Fourty-Seventh Annual IEEE Symposium on Foundations of Computer Science*, pages 69–76, October 2006.

[36] George H. L. Fletcher, Hardik A. Sheth, and Katy Börner. Unstructured peer-to-peer networks: Topological properties and search performance. In *AP2PC '04: Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing*, pages 14–27, July 2004.

[37] Pierre Fraigniaud and Philippe Gauron. D2B: a de Bruijn based content-addressable network. *Theoretical Computer Science*, 355(1):65–79, 2006.

[38] Joel Friedman and Jean-Pierre Tillich. Generalized Alon–Boppana theorems and error-correcting codes. *SIAM Journal on Discrete Mathematics*, 19(3):700–718, 2005.

[39] Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB '04: Proceedings of the thirtieth International Conference on Very Large Data Bases*, pages 444–455. VLDB Endowment, 2004.

[40] Pu Gao. The connectivity of the random regular graphs generated by the pegging algorithm. *Journal of Graph Theory*, 2010. To appear.

[41] Pu Gao and Nicholas Wormald. Short cycle distribution in random regular graphs recursively generated by pegging. *Random Structures and Algorithms*, 34(1):54–86, 2009.

[42] Bugra Gedik and Ling Liu. Reliable peer-to-peer information monitoring through replication. In *Proceedings of the Twenty-Second International Symposium on Reliable Distributed Systems*, pages 56–65. IEEE Computer Society, October 2003.

[43] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *INFOCOM '04: Proceedings of the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2004.

[44] The Gnutella protocol specification v0.4. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2001.

[45] Catherine Greenhill, Fred B. Holt, and Nicholas Wormald. Expansion properties of a random regular graph after random vertex deletions. *European Journal of Combinatorics*, 29(5):1139–1150, 2008.

[46] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *ACM SIGOPS Operating Systems Review*, 37(5):314–329, 2003.

[47] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[48] Gernot Gwehenberger. Anwendung einer binären Verweiskettenmethode beim Aufbau von Listen. *Elektronische Rechenanlagen*, 10(5):223–226, 1968.

[49] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in dht-based peer-to-peer networks. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 242–259, London, UK, 2002. Springer-Verlag.

[50] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[51] Nicolas Heine. Analyse von Graphtransformationen zur Aufrechterhaltung dynamischer Zufallsnetzwerke. Master's thesis, University of Paderborn, 2009.

[52] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distribted object location in a dynamic network. In *SPAA '02: Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 41–52, New York, August 10–13 2002. ACM Press.

[53] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[54] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with pier. In *VLDB '2003: Proceedings of the Twenty-Ninth International Conference on Very Large Data Bases*, pages 321–332. VLDB Endowment, 2003.

[55] International ispell. `http://www.lasr.cs.ucla.edu/geoff/ispell.html`.

[56] H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. BATON: a balanced tree structure for peer-to-peer networks. In *VLDB '05: Proceedings of the Thirty-First International Conference on Very Large Data Bases*, pages 661–672. VLDB Endowment, 2005.

[57] H. V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, Rong Zhang, and Aoying Zhou. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *ICDE '06: Proceedings of the Twenty-Second International Conference on Data Engineering*, Washington, DC, USA, 2006. IEEE Computer Society.

[58] M.R. Jerrum and A. J. Sinclair. Fast uniform generation of regular graphs. *Theoretical Computer Science*, 73:91–100, 1990.

[59] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks — a case study of Gnutella. Technical report, University of Cincinnati, 2001.

[60] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS '03: Proceedings of the Second International Workshop on Peer-to-Peer Systems*, pages 98–107, 2003.

[61] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC '97: Proceedings of*

the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pages 654–663, El Paso, Texas, 4–6 May 1997.

[62] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *FOCS '00: Proceedings of the Fourty-First Annual Symposium on Foundations of Computer Science*, page 565, Washington, DC, USA, 2000. IEEE Computer Society.

[63] Minkyu Kim and Muriel Medard. Robustness in large-scale random networks. In *INFOCOM '04: Proceedings of the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2004.

[64] Ching Law and Kay-Yeung Siu. Distributed construction of random expander networks. In *INFOCOM '03: Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 2133–2143, 2003.

[65] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *INFOCOMM '05: Proceedings of the Twenty-Fourth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 225–236, March 2005.

[66] Mei Li, Wang-chien Lee, and Anand Sivasubramaniam. DPTree: A balanced tree based indexing framework for peer-to-peer systems. In *ICNP '06: Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 12–21, Washington, DC, USA, 2006. IEEE Computer Society.

[67] Chu Yee Liau, Wee Siong Ng, Yanfeng Shu, Kian-Lee Tan, and Stéphane Bressan. Efficient range queries and fast lookup services for scalable p2p networks. In *DBISP2P '04: Proceedings of the Second International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Lecture Notes in Computer Science. Springer, 2004.

[68] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, pages 233–242, New York, July 21–24 2002. ACM Press.

[69] Laszlo Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdös is Eighty*, 2:353–398, 1996.

[70] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the Sixteenth International Conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM Press.

[71] Peter Mahlmann, Thomas Janson, and Christian Schindelhauer. 3nuts: A locality-aware peer-to-peer network combining random networks, search trees, and dhts. Technical Report tr-ri-09-309, Heinz Nixdorf Institute, Paderborn, Germany, December 2009.

[72] Peter Mahlmann and Christian Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *SPAA'05: Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 155–164, New York, NY, USA, 2005. ACM Press.

[73] Peter Mahlmann and Christian Schindelhauer. Distributed random digraph transformations for peer-to-peer networks. In *SPAA '06: Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 308–317, New York, NY, USA, 2006. ACM Press.

[74] Peter Mahlmann and Christian Schindelhauer. *Peer-to-Peer-Netzwerke: Algorithmen und Methoden*. Springer-Verlag Berlin, 1st edition, June 2007. ISBN 3540339914.

[75] Peter Mahlmann and Christian Schindelhauer. Random graphs for peer-to-peer overlays. In *The European Integrated Project "Dynamically Evolving, Large Scale Information Systems (DELIS), Proceedings of the Final Workshop*, number 222, pages 1–22. HNI Verlagsschriftenreihe, Paderborn, 27 - 28 February 2008.

[76] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[77] Brendan D. McKay and Nicholas C. Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11:52–67, 1990.

[78] Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa. P2PR-Tree: An R-tree-based spatial index for peer-to-peer environments. In *Current Trends in Database Technology - EDBT 2004 Workshops*, pages 516–525, 2004.

[79] Alberto Montresor, Mark Jelasity, and Ozalp Babaoglu. Chord on demand. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 87–94, Washington, DC, USA, 2005. IEEE Computer Society.

[80] Moni Naor and Udi Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA '03: Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 50–59, New York, NY, USA, 2003. ACM Press.

[81] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *IPTPS '03: Proceedings of the Second International Workshop on Peer-to-Peer Networks*, pages 88–97, 2003.

[82] Sotiris E. Nikoletseas, Krishna V. Palem, Paul G. Spirakis, and Moti Yung. Connectivity properties in random regular graphs with edge faults. *International Journal on Foundations of Computer Science*, 11(2):247–262, 2000.

[83] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

[84] S. Oaks, B. Traversat, and L. Gong. *JXTA in a Nutshell*. O'Reilly Press, 2002.

[85] Andy Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Media, March 2001.

[86] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *FOCS '01: Proceedings of the Fourty-Second IEEE Symposium on Foundations of Computer Science*, pages 492–499, Washington, DC, USA, 2001. IEEE Computer Society.

[87] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 21(6):995–1002, 2003.

[88] Bill Pickelhaupt and Kevin Starr. *Shanghaied in San Francisco*. Mystic Seaport Museum, 1970.

[89] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, New York, June 1997. ACM Press.

[90] Project jxta. `http://www.jxta.org`.

[91] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[92] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Brief announcement: prefix hash tree. In *PODC '04: Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, pages 368–368, New York, NY, USA, 2004. ACM Press.

[93] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Prefix hash tree: An indexing data structure over distributed hash tables. Technical report, Intel Research Berkeley, 2004.

[94] Dana Randall and Prasad Tetali. Analyzing glauber dynamics by comparison of markov chains. In *LATIN '98: Proceedings of the Third Latin American Symposium on Theoretical Informatics*, pages 292–304, London, United Kingdom, 1998. Springer-Verlag.

[95] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[96] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *P2P '01: Proceedings of the First IEEE International Conference on Peer-to-Peer Computing*, pages 99–100, Washington, DC, USA, 2001. IEEE Computer Society.

[97] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: search methods. *Computer Networks*, 50(17):3485–3521, 2006.

[98] Jordan Ritter. Why gnutella can't scale. no, really. `http://www.darkridge.com/~jpr5/doc/gnutella.html`, 2001.

[99] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, United Kindgom, 2001. Springer-Verlag.

[100] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[101] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN '02: Proceedings of the 2002 SPIE Conference on Multimedia Computing and Networking*, 2002.

[102] Christian Schindelhauer and Gunnar Schomaker. Weighted distributed hash tables. In *SPAA '05: Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 218–227, New York, NY, USA, 2005. ACM.

[103] Stefan Schmid and Roger Wattenhofer. Structuring unstructured peer-to-peer networks. In *HiPC '07: Proceedings of the Fourteenth Annual IEEE International Conference on High Performance Computing*, LNCS 4873, Berlin, Heidelberg, Germany, December 2007. Springer.

[104] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *P2P '01: Proceedings of the First IEEE International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2001. IEEE Computer Society.

[105] Hendrik Schulze and Klaus Mochalski. ipoque internet study 2008/2009. `http://www.ipoque.com/resources/internet-studies`, 2009.

[106] Clay Shirky. What is p2p... and what isn't, November 2000. `http://missingmanuals.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html`, O'Reilly.

[107] Andreas Stathopoulos. Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. *SIAM Journal on Scientific Computing*, 29(2):481–514, 2007.

[108] Andreas Stathopoulos and James R. McCombs. Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues. *SIAM Journal on Scientific Computing*, 29(5):2162–2188, 2007.

[109] Angelika Steger. *Diskrete Strukturen (Band 1)*. Springer-Verlag, January 2001.

[110] Angelika Steger and Nicholas C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999.

[111] Ralf Steinmetz and Klaus Wehrle. Peer-to-peer-networking & -computing. *Informatik-Spektrum*, 27(1):51–54, 2004.

[112] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2005.

[113] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[114] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[115] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the Sixth ACM SIGCOMM Conference on Internet Measurement*, pages 189–202, New York, NY, USA, 2006. ACM Press.

[116] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. *IEEE/ACM Transactions on Networking*, 16(2):267–280, 2008.

[117] D. A. Tran and T. Nguyen. Hierarchical multidimensional search in peer-to-peer networks. *Computer Communications*, 31(2):346–357, 2008.

[118] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project jxta 2.0 super-peer virtual network, May 2003.

[119] Udi Wieder. *The Continuous-Discrete Approach for Designing P2P Networks and Algorithms*. PhD thesis, Weizmann Institute of Science, 2005.

[120] Nicholas C. Wormald. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory, Series B*, 31(2):156–167, 1981.

[121] Nicholas C. Wormald. Models of random regular graphs. In *Surveys in Combinatorics*, pages 239–298. Cambridge University Press, 1999.

[122] Ellen Zegura, Kenneth Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *INFOCOM '96: Proceedings of Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, volume 2, pages 594–602, 1996.

[123] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *IPTPS '05: Proceedings of the Fourth International Workshop on Peer-To-Peer Systems*, February 2005.