**Julius-Maximilians-Universität Würzburg**
Institut für Informatik
Lehrstuhl für Verteilte Systeme
Prof. Dr. P. Tran-Gia

# Performance Analysis of Structured Overlay Networks

**Andreas Binzenhöfer**

Würzburger Beiträge zur
Leistungsbewertung Verteilter Systeme

Bericht 01/08

**Würzburger Beiträge zur**
**Leistungsbewertung Verteilter Systeme**

**Herausgeber**

Prof. Dr. P. Tran-Gia
Universität Würzburg
Institut für Informatik
Lehrstuhl für Verteilte Systeme
Am Hubland
D-97074 Würzburg

Tel.: +49-931-888-6630
Fax.: +49-931-888-6632
email: trangia@informatik.uni-wuerzburg.de

**Satz**

Reproduktionsfähige Vorlage vom Autor.
Gesetzt in LaTeX Computer Modern 9pt.

**ISSN 1432-8801**

# Performance Analysis of Structured Overlay Networks

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius–Maximilians–Universität Würzburg

vorgelegt von

**Andreas Binzenhöfer**

aus

Würzburg

Würzburg 2008

Eingereicht am: 04.12.2007
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr.-Ing. P. Tran-Gia
2. Gutachter: Prof. Dr.-Ing. J. Eberspächer
3. Gutachter: Prof. N. Wakamiya
Tag der mündlichen Prüfung: 06.02.2008

# Acknowledgements

This monograph summarizes five years of research studies, an everlasting struggle for publications, numerous perceived nervous breakdowns, and above all a very pleasant time with my colleagues who by now have all become dear friends of mine. This work would not have been possible if it had not been for them and a number of other people who all helped and supported me in different ways.

First of all I would like to thank my advisor Prof. Phuoc Tran-Gia who not only provided me with an excellent opportunity to work in a scientific environment but did so with great technical as well as interpersonal commitment. Due to his untiring efforts I was able to participate in both industrial as well as European research projects. Beyond that I was also given the opportunity to visit international conferences and colleagues, which was an invaluable experience I would not like to have missed. I would also like to express my gratitude to Prof. Jörg Eberspächer and Prof. Naoki Wakamiya who both acted as reviewers of this thesis and provided me with valuable comments in discussions at multiple occasions including joint projects, conferences, workshops, and short term visits. I am also very much obliged to Prof. Dr. Klaus Schilling and Prof. Dr. Dietmar Seipel for being available as examiners for my disputation. Furthermore, I would like to thank Markus Fiedler, Ilkka Norros, Masayuki Murata, and Krzysztof Pawlikowski for providing me the opportunity to visit their research laboratories.

My deepest appreciation goes to our secretary Mrs. Förster who with surpassing patience and dedication makes life a lot easier for us at the department. I would also like to thank the people at Bosch, Siemens, AOK, and DATEV who worked together with me over the last five years. From both communication and collaboration with them I gained a great deal of experience and never lost track of the practical aspects of my work. In addition to this I would like to thank Barbara Emmert, Stefan Mühleck, Björn auf dem Graben, Johannes Dölfel, Markus Weiß and especially Holger Schnabel, who all completed their diploma thesis under my supervision and thereby eased my workload.

I also had the privilege of working with amazing colleagues, who were always generous with their time and expertise. In particular, I'm very much indebted to Dirk Staehle who has been generous and unstinting with his advise and suggestions. He never got tired of discussing a problem over coffee and fascinatingly always came up with a proper solution. In addition to him I also would like to thank Tobias Hoßfeld, Simon Oechsner, Robert Henjes, and Oliver Rose who all helped me to solve an important coupon collectors problem. Rastin Pries who constantly challenged and occasionally outmatched me in a ferocious battle for the most delicious breakfast. Barbara Staehle who has an unmatched talent to transform scientific research results into a true piece of art. Michael Menth who is the living proof that it still pays off to be fair-minded. Rüdiger Martin and Andreas Mäder who both accompanied me for more than ten years on my journey through University. I also greatly benefited from conversations with Daniel Schlosser, Michael Duelli, Thomas Zinner, Jens Milbrandt, Stefan Köhler, and Kurt Tutschku. Furthermore, it is a pleasure to acknowledge my former colleague Kenji Leibnitz who despite a distance of roughly 9000 km became an even closer friend during the last years. Among the many researchers I met outside our department, Gerald Kunzmann became my closest colleague and a dear friend.

Special thanks go to my parents Alfred and Margot Binzenhöfer, as well as to my brother Stefan Binzenhöfer. My family provided me with the necessary means to adhere to my ideas and plans without ever questioning any of my decisions. Finally, I would like to offer heartfelt thanks to my wife Stefanie Binzenhöfer. Throughout the many years we spent together she has always been my help and strength. Whenever I needed a friend, she was there for me. Whenever I had doubts, she believed in me. I dedicate this to her.

# Contents

# 1 Introduction

In the last decades computer networks became one of the major building blocks of our society. Large parts of our private and business life already depend on this infrastructure for communication, information, and exchange of data. However, until today the Internet is still driven by algorithms and technologies which have been developed in the seventies and eighties. The traditional service architecture in the Internet, e.g., is based on the simple client-server principle. That is, a single central unit provides several clients with a service. Due to the continuously increasing number of both users and services in today's networks, it became time for a revolutionary rethink of this simple approach.

Step by step, new functionality, which the network inherently did not posses, is added. In large part this is done by establishing overlay networks, i.e. logical connections between users on top of the physical network. Such overlay networks are an enabling technology for user driven applications which shift the intelligence from within the network to its edges. The prime example for an overlay based application is the direct distribution of files among end-users without them being dependent on a company or any other central entity. While in the beginning the success of overlay networks was mainly driven by peer-to-peer (p2p) file-sharing, the underlying structures are by far not limited to the efficient distribution of media content to a large number of customers. They are, e.g., used for distributed network management or to build a global persistent data store where each participating user contributes some storage and bandwidth. Distributed Voice-over-IP platforms like Skype are another good example, as they relocate voice traffic from traditional telephone lines to the Internet using overlay networks.

However, randomly established overlay networks provide only a best effort service and cannot offer any guarantees or service level agreements. Therefore, a new generation of structured overlay systems based on Distributed Hash Tables (DHTs) is currently investigated in the research community. In such DHTs it is well-defined how the participating users are interconnected and how messages are routed in the overlay network. Inspired by this potential, the first business models based on overlay architectures have emerged. Companies start to discover the advantages of decentralized structured overlay networks. They are no longer dependent on a single central unit nor do they have to invest in server farms to guarantee the scalability of their systems. Together with those new systems, however, new challenges arise as well. Before structured overlay networks can successfully be used in a corporate environment, their performance needs to be understood in detail. The achievable level of performance determines in how far overlay architectures can be used to offer a reliable service and what service level agreements can be negotiated. Another problem is that such architectures are highly distributed and therefore appear as a black box to the operator. Yet an operator does not want to lose control over his system and needs to be able to continuously observe and examine its current state at runtime.

## 1.1 Contribution

The contribution of this monograph is two-fold. First, we evaluate the performance of structured overlay networks under different aspects and thereby illuminate in how far such architectures are able to support carrier-grade applications. To enable operators to monitor and understand their deployed system, we secondly introduce both active as well as passive methods to gather information about the current state of the overlay network.

In terms of performance, we deduce an analytical model for real-time applications based on the Chord algorithm. The main goal is to prove scalability for very large overlay networks to be able to guarantee certain quality of service demands in large peer populations. Furthermore, we evaluate the impact of highly probabilistic network delay variations, which also strongly influence the duration of searches. The stability of p2p overlay networks is also affected by the dynamic behavior of the end user. In this context, we show that the probability to lose the overlay structure of a Distributed Hash Table (DHT) is not negligible in all cases. In particular, we present an analytical expression that can be used to calculate the probability to lose the routing functionality of a DHT given a certain number of overlay connections. In order to understand the performance of structured overlay networks in greater detail, we introduce a discrete event simulator which is designed to handle a very large number of peers. We present simulative studies of the search duration, the overlay stability, and the maintenance traffic needed to stabilize the overlay structure. Based on these results, we unveil the weak points of structured overlay networks and pinpoint their root causes. For each problem we present an optimization, which eliminates the disadvantages and makes structured overlay networks more feasible for business applications.

Apart from performance concerns, one of the main reasons why telecommunication carriers are still hesitant to build p2p applications is the lack of control a provider has over the running system. The system appears as a black box to its operator such that he does not know anything about the current size, performance, or stability of its application. We therefore present and discuss different active and passive methods to gather information about a running p2p overlay network. We develop algorithms to estimate different system aggregates like the current number of peers in the overlay or the distribution of the session times of the participating peers. The advantage of these methods is that they operate passively and solely require information which is locally available to a peer. The estimates can also be regarded as a first step toward a self-organizing overlay network. That is, peers can use these estimates to dynamically adapt the maintenance overhead to the current situation in the overlay network. Finally, we introduce a scalable approach to actively create a snapshot of a running p2p system. The overhead involved in creating the snapshot is evenly distributed to the participating peers so that each peer has to contribute only a negligible amount of bandwidth. We discuss the collected information and the conclusions drawn from it.

## 1.2 Outline

The remainder of this monograph is organized as follows. Chapter 2 lays the foundation for the performance evaluation in the following chapters. We discuss the basic concepts behind overlay networks, the differences between the individual approaches as well as the main areas of application. Thereby, a special focus is laid upon structured overlay networks, which build the core of this work.

Chapter 3 gives a performance evaluation of structured overlay networks under different aspects. We first highlight the importance of considering both the functional and the stochastic scalability of such architectures. This is followed by a focused survey of current research in this area. To obtain a first understanding of the system performance, the peer distance distribution is calculated in a static overlay network. From this we derive quantiles for the duration of a search, which can be used by a provider to establish service level agreements. In addition, we validate the stability of the overlay by calculating realistic probabilities to lose the structure of the overlay in dependence of the peer behavior. The chapter closes with an in-depth study of structured overlay networks which use a special distance metric. Based on the results, we derive and evaluate different modifications and improvements to the original algorithm.

Chapter 4 investigates different possibilities of a single peer to obtain global knowledge of important system features. We develop a passive maximum likelihood estimator for the current peer population and evaluate the accuracy of the estimates in realistic scenarios. This is followed by a comparison of different methods to estimate the session time distribution of the participating peers based on local information. Thereby, the different algorithms are evaluated in terms of accuracy, responsiveness, and practicability. In addition to the estimation methods, we introduce a scalable algorithm to create a snapshot of the running system from a central position. This enables an operator to actively obtain an accurate and timely picture of its deployed overlay network and to initiate appropriate countermeasures if needed. Chapter 5 summarizes the main findings gained throughout the course of this work. Based on these, we draw conclusions and give an overview of open issues and possible approaches for future work.

# 2 Peer-to-Peer Key Technologies

In general, an overlay network can be described as a virtual network built on top of one or many already existing networks. In this sense, peer-to-peer (p2p) networks represent a special subset of decentralized overlays, where each participating peer simultaneously acts as client and server, the so called servent concept. Such p2p algorithms are used to provide connectivity among a large number of physically distributed peers which share a common interest, like the desire for a file. P2p overlay networks support a wide variety of applications, whereas the most common tasks are storage, search, and distribution of information and files. Thereby, the specific p2p protocol determines which and how many overlay neighbors a peer maintains connections to, how frequently those neighbors are contacted for maintenance purposes, and which other peers will be contacted when searching for information.

In literature as well as in practice numerous different p2p architectures and protocols can be found, which already account for the major part of traffic in the Internet today [29]. On an abstract level, those architectures can roughly be divided into pure and hybrid overlay networks. While in a pure p2p network all peers are equal and perform exactly the same tasks, there are some dedicated peers in hybrid networks which are assigned a special function. In the case of pure overlay networks, we further distinguish between unstructured and structured architectures. While peers in an unstructured overlay maintain connections to random other overlay peers (cf. Chapter 2.1), a structured p2p protocol exactly defines the relationship among overlay peers (cf. Chapter 2.2).

As the different mechanisms have been designed for different purposes, none of the above described approaches is clearly superior to all other approaches. In fact, they can be compared by many different metrics like the cost of a peer to participate in the network [30] or the number of offered features like the ability to perform full text searches [31]. While the most common application today is file sharing, p2p overlays can also be used for other purposes like permanent distributed storage [32–34], distributed VoIP services [17,35], or distributed network management [9,18,36]. However, the p2p paradigm should not blindly be applied to all areas of application as in some scenarios the disadvantages might outweigh the benefits [37,38]. In the following, we give an overview of the most important pure p2p architectures with a focus on structured overlay networks which form the core of this thesis. Furthermore, some examples of hybrid networks will be described in Chapter 2.3.

## 2.1 Unstructured Overlay Networks

The first pure p2p networks did not have any particular structure as participating peers maintained random connections to other peers in the overlay (cf. Chapter 2.1.1). Obviously those networks are easy to construct but hard to disable since they are fully decentralized and do not offer a single point-of-failure. The search for information or stored files is usually done by simple flooding, gossip-based algorithms, or random walks. While this only requires a very limited per-node state, it results in very poor search performance and does not guarantee that a search query will successfully be resolved, even if the queried information does exist in the network. The scalability of the data discovery process is furthermore limited by the overhead traffic generated during a search. Today those networks are mainly used to protect the anonymity of content providers and content seekers (cf. Chapter 2.1.2).

### 2.1.1 Gnutella: Distributed Search and Flooding

In this section we will focus on Gnutella v0.4 [39] as the most popular representative of pure unstructured overlay networks. Peers maintain connections to $c$ random peers in the overlay and the entire protocol is based on simple messages which are exchanged between the participating peers. The structure of such a message is shown in Table 2.1. Thereby the descriptor ID is used to uniquely

| **Fields** | Descriptor ID | Payload Descriptor | TTL | Hops | Payload Length |
|---|---|---|---|---|---|
| **Byte offset** | 0...15 | 16 | 17 | 18 | 19...22 |

Table 2.1: *Structure of a Gnutella message*

identify a message in the overlay network and the payload descriptor determines the kind of message which may be one of the following:

- **Ping:** Actively discover and probe for other peers in the overlay
- **Pong:** Response to a ping including address of a connected servent and information about number and total kB of files shared
- **Query:** Primary search message including the query descriptor
- **QueryHit:** Response to a query including number of results and how to obtain them
- **Push:** Simple mechanism to allow peers behind firewalls to contribute to file distribution

To join a Gnutella network a peer $p$ simply sends a ping message to an arbitrary peer which is already participating in the overlay network. As soon as this peer answers with a pong message peer $p$ is part of the Gnutella overlay. These ping messages are then repeated periodically and forwarded to all neighbors in order to stabilize the overlay network. Once peer $p$ finds out about $c$ other peers it is fully integrated into the overlay. Searches for content are then implemented using a simple flooding mechanism. Peer $p$ broadcasts a search query to its $c$ overlay neighbors which in turn forward the query message to all their overlay neighbors except the peer they received the message from. Peers which store the desired content answer with a query hit message which is sent along the reverse path. To control the overhead generated during query requests and responses, each query contains a time-to-live (TTL) counter which is decreased each time the message is forwarded. Query messages with a TTL=1 are no longer forwarded.

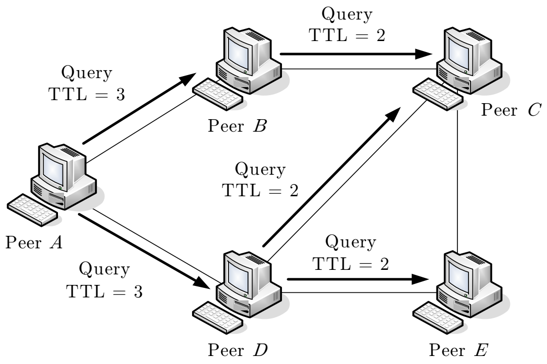


Figure 2.1: *Flooding in the Gnutella overlay*

Figure 2.1 illustrates a search issued by peer $A$ using a TTL value of 3. After decreasing the TTL to 2, the search is recursively forwarded by peer $B$ and peer $D$. Obviously, the main drawbacks of this flood-based search algorithm are slow, bandwidth intensive, and highly redundant queries which in addition cannot be guaranteed to be successful. Aberer et al. [40] showed that the standard parameters of Gnutella using a $TTL = 7$ and $c = 4$ overlay connections lead to

$$M_{query} = 2 \cdot \sum_{i=0}^{TTL} c \cdot (c-1)^i = 2 \cdot \sum_{i=0}^{7} 4 \cdot 3^i = 26240 \qquad (2.1)$$

messages per query including the responses. Newer versions of Gnutella therefore introduced the concept of Ultrapeers which is similar to the SuperPeer concept described in Chapter 2.3.3.

### 2.1.2 Freenet: Anonymity Protection

Freenet [41] is an unstructured overlay which aims at anonymity rather than efficient content distribution. That is, the main goal of the Freenet overlay is to assure the anonymity of both the content publishers as well as the information consumers. In particular, the Freenet Project [42] focuses on freedom of speech, resistance to information censorship as well as privacy for information producers, consumers, and holders.

This is realized by assigning each stored file a Globally Unique Identifier (GUID) which is based on three different types of keys. At first a Keyword Signed Key (KSK) is derived from a string which is chosen by the user and intended to describe the file, e.g. **text/thesis/binzenhoefer**. In order to avoid conflicts which might occur when different users choose the same descriptive string for their files, an additional Signed Subspace Key (SSK) is used to determine the unique file key. Finally, a Content Hash Key (CHK) is calculated by directly hashing the content of the corresponding file. A more detailed description of the KSK, the SSK, and the CHK can be found in [41]. To insert a new file into the overlay, a user calculates both the SSK and the CHK of the file and inserts a pointer to the CHK under the hash value of the SSK, while the file itself is stored under the hash value of the CHK. Other users can then retrieve this information using a two-step search approach. At first they calculate the SSK from the descriptive string in order to lookup the CHK. In the second step they can then search for the actual file using the CHK. Thereby, it is still an unsolved problem how the searching user obtains the descriptive string of the file in the first place. In practice this is usually done by using other means of distribution like publishing it on a website.

To avoid the problems of redundant and unscalable flooding, Freenet applies a steepest-ascent hill-climbing search mechanism. Thereby, the main idea is to forward queries which cannot be answered locally to the peer which is believed to be the closest to a specific target. To this end, a peer maintains a routing table which, for each overlay neighbor, lists the keys which this neighbor is likely to hold. Figure 2.2 shows a typical example of a search in the Freenet overlay as described in [41]. Peer A forwards a search for information stored on Peer D to

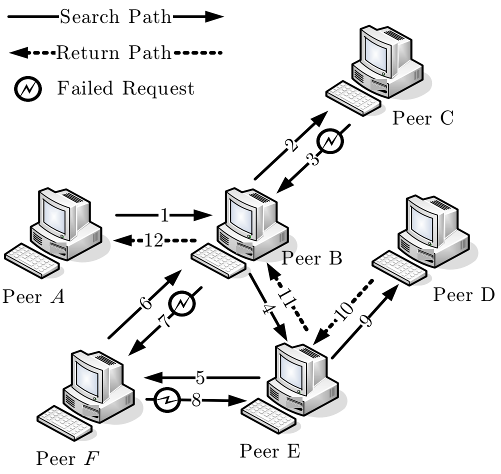


Figure 2.2: *Searching in the Freenet Overlay*

its only overlay neighbor B. Peer B recursively contacts Peer C, which does neither hold the desired information nor have any further overlay neighbors. Peer C therefore answers with a request failed message. Peer B then contacts its next overlay neighbor Peer E, which in turn contacts Peer F. Peer F forwards the query to Peer B, whereas Peer B detects a loop comparing the unique message ID to the temporary list of open queries which each peer maintains for a predetermined period of time. Due to the lack of further overlay neighbors, Peer F reports a request failed message back to Peer E, which forwards the query to Peer D. Finally, Peer D holds the desired file and returns it back to A via E and B. As subsequent queries are likely to take a similar path to the destination, Freenet applies a special caching algorithm, where peers on the return path locally store the most recently requested files.

The privacy and anonymity of Freenet are mainly achieved by the fact that the peer holding the searched information does not directly reply to the originator but sends the file hop-by-hop via the return path. In fact, each node involved in the entire search process does only know its immediate neighbors. Thus, no peer can be sure if it received the query request from the searching peer itself or from a forwarding peer nor can it be sure if the next peer is the recipient or merely another forwarding peer. Furthermore, messages with a hop-to-live value of zero are still forwarded with finite probability in order to make attacks on the privacy of Freenet peers more complicated. While the above described mix-net scheme guarantees the anonymity of authorship and readership, it is not very well suited for the exchange of large files. In practice, Freenet is therefore rather used for anonymous communication than for media content distribution. A more recent version of Freenet [43] further improves the privacy of its users by combining its small-world features with the idea of a Darknet, a small group of people trusting each other.

## 2.2 Structured P2P Networks

While the first unstructured p2p networks have been created and deployed by individual persons or commercial companies, structured p2p architectures were proposed within the research community in an effort to eliminate the problems which appeared in unstructured overlay networks. Thereby, all structured p2p algorithms share the same underlying principle. Each participating peer is assigned a unique $m$-bit identifier, i.e. an integer between 1 and $2^m$ which is derived by hashing a unique feature like the IP-address used by the peer. To maintain the overlay structure peers then establish connections to their $r$ closest overlay neighbors according to a special metric defined on their identifiers by the individual p2p protocol. In addition to this list of neighbors, each peer also maintains connections to more distant peers which are used as shortcuts in the overlay in order to guarantee fast searches.

To determine the exact peers at which a specific file is stored in the overlay, files are also hashed into the same identifier space in such a way that their identifiers intermix with the identifiers of the peers. This concept is illustrated in Figure 2.3 for a typical value of $m = 160$. The files are then stored at the closest

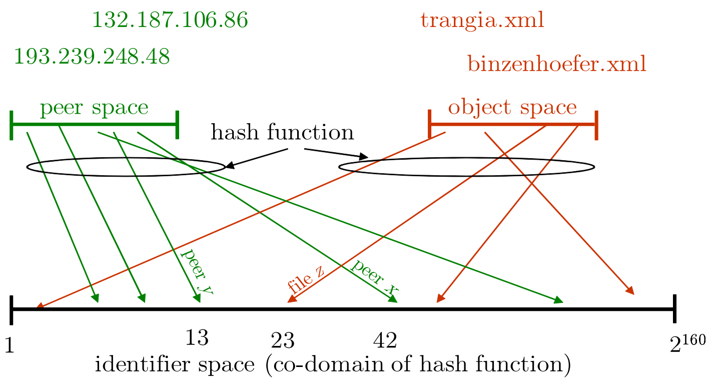


Figure 2.3: *Assignment of identifiers in a Distributed Hash Table*

peer according to the given metric. Due to this hash-based principle structured p2p networks are often referred to as Distributed Hash Tables (DHTs). As a result of the structure of such a DHT, it can be guaranteed that search queries will always terminate, either successfully by returning the file itself or by sending a file-not-available message. While DHTs avoid the scalability issues of unstructured p2p networks, they lack the possibility to perform full text searches and have to maintain their structure despite of churn, the continuous process of peers joining and leaving the overlay network at arbitrary times. In the following, we will give an overview of the most important DHT algorithms and their uses [44].

## 2.2.1 Chord

The Chord algorithm was introduced in 2001 by Stoica et al. [45] and became the most studied structured p2p overlay network. It is based on a simple ring structure and describes how new peers join the overlay, where files are stored, and how they can be retrieved by other peers.

### General Architecture

The general architecture is shown in Figure 2.4 which shows an exemplary Chord ring consisting of seven peers and four files using an identifier space with $m = 7$ bit and thus $2^m = 128$ possible identifiers. The participating peers are arranged on a logical ring structure in such a way that their identifiers are ascending in a clockwise direction. The first peer succeeding a peer $z$ in this clockwise direction is called the successor $s$ of $z$, the first peer in a counterclockwise direction is called the predecessor $p$ of $z$.

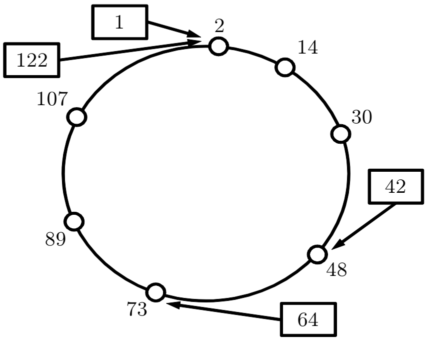


Figure 2.4: *Chord ring with seven peers and four files*

The location of files in the overlay is also well defined by a simple rule. All files whose identifiers fall between the identifier of peer $z$ and its successor $s$ are stored at peer $s$. In the example, file 42 is stored at peer 48 and file 64 at peer 73, respectively. Note that the identifier circle is based on a modulo arithmetic and thus both file 122 and file 1 are stored at peer 2.

## Join and Leave Events

To join a Chord-based overlay network, a peer $z$ needs to know an arbitrary peer $x$ which is already participating in the overlay. This is a general problem in p2p networks and usually solved by either contacting some well known peers, retrieving a list of online peers from a website, or using a central bootstrap server. Peer $x$ can then determine the identifier of the direct successor of the joining peer $z$ using the search algorithm of Chord, which will be described later. Once peer $z$ has the

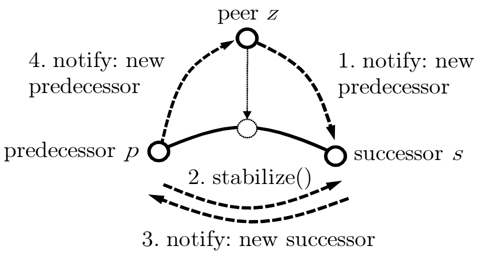


Figure 2.5: *Different steps of a join procedure in Chord*

contact information of its direct successor $s$ it starts the join procedure illustrated in Figure 2.5. At first it notifies $s$ that it is its new predecessor (cf. step 1). As soon as the old predecessor $p$ of $s$ contacts $s$ during the periodic stabilize routine (cf. step 2), peer $s$ will notify $p$ about its new successor $z$ (cf. step 3). Peer $p$ will then notify peer $z$ that it is its predecessor (cf. step 4), after which peer $z$ becomes fully integrated in the logical ring structure.

In principle, peers leaving the overlay could contact and inform their overlay neighbors in a similar way. In practice, however, an overlay network is likely to face unfriendly leaves and sudden node failures. To deal with such churn behavior the Chord algorithm performs some special maintenance routines.

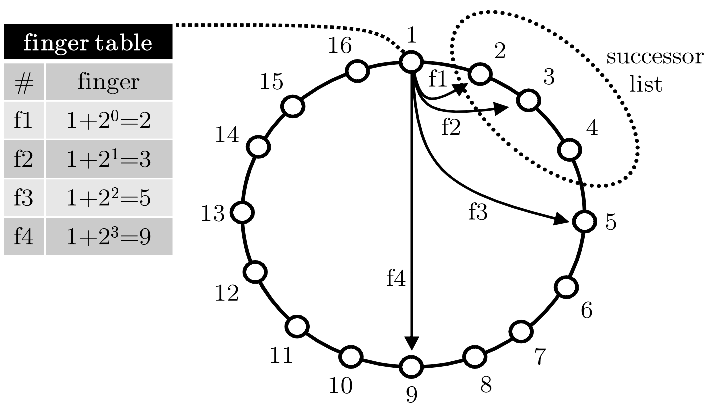
### Maintenance of the Overlay Structure

To maintain the logical ring structure of the overlay, each peer stores pointers to the first $r$ successors in a clockwise direction. Thus, if one of the peer's $r$ successors goes offline, the peer will still know the next $r - 1$ peers on the ring. Figure 2.6 shows a successorlist of size $r = 3$ for peer 1 in a Chord ring consisting of 16 peers. In order to detect changes in the neighborhood, peer $z$ periodically contacts its direct predecessor $p$ as well as its direct successor $s$ every $t_{stab}$ seconds and performs a stabilize routine. During this stabilization the two corresponding peers reconcile their neighborlists and adapt them accordingly. If a peer does not respond to a stabilize call, it is removed from the neighborlist and the next peer is contacted. The default value for the periodic contact interval is $t_{stab} = 30s$.

In practice, this simple stabilization algorithm does not suffice to handle high churn rates and has to be improved accordingly [21]. One possible solution is to make Chord symmetric and additionally maintain a list of predecessors [46]. Since it is impossible to avoid all failures a mechanism [14] has been proposed to recover from a loss of the ring structure, which might e.g. happen during a mass exit of a large number of peers. Besides the overlay structure, the availability of stored files must also be guaranteed. The Chord protocol itself does not directly specify any redundancy mechanisms. However, numerous different approaches have been proposed in literature. Ramabhadran et al. [47] and Datta et al. [48] give a good overview of redundancy algorithms and additionally compare the different approaches analytically to show which mechanism is best suited for which scenario.

### Search Algorithm and File Insertion

The entire functionality of the Chord algorithm is based on its search algorithm. To insert a file a peer searches for the first peer succeeding the file's identifier, where the file will then be stored. The file can be retrieved by other peers at a later point in time using the same search routine. If a peer searches a specific



| finger table | |
|---|---|
| # | finger |
| f1 | $1+2^0=2$ |
| f2 | $1+2^1=3$ |
| f3 | $1+2^2=5$ |
| f4 | $1+2^3=9$ |

Figure 2.6: *Assignment of identifiers and pointers in a Distributed Hash Table*

identifier, it will forward the query to its successor, which in turn will forward the query to its own successor until the search hits the peer which is responsible for the searched identifier. Once the responsible peer is found, it will transmit the answer directly to the originator, i.e. the peer seeking the information. Obviously this is very inefficient, as a peer needs $O(n)$ messages to complete this kind of search, where $n$ is the current number of peers in the overlay. To improve the search duration, a peer also maintains a finger table, i.e. a list of peers called fingers which are used as shortcuts through the ring to speed up the search process. Thereby, the $i$-th entry in a peers finger table contains the identity of the first peer that succeeds $z$'s own hash-value by at least $2^i - 1$ on the Chord ring. That is, peer $z$ with hash value $id_z$ has its fingers at $id_z + 2^i - 1$ for $i = 1$ to $m$, where $m$ is the number of bits used for the identifiers. Figure 2.6 illustrates a simple example using a Chord-ring consisting of 16 peers. Peer 1 has a four different fingers f1 to f4 pointing to $2(= 1 + 2^{1-1})$, $3(= 1 + 2^{2-1})$, $5(= 1 + 2^{3-1})$, and $9(= 1 + 2^{4-1})$, respectively. When searching a file, peer $z$ is now able to send the query to its finger, whose hash value most immediately precedes the hash value of the searched file. If this finger is not able to answer the search locally, it forwards the query accordingly. Otherwise, the search is finished and the finger directly returns the answer back to the searching peer $z$. This way queries can be answered using $O(log_2(n))$ messages. This kind of search is called recursive, since each peer participating in the search forwards the query recursively to its closest finger. It is, however, also possible to perform iterative queries [49], where each peer involved in the query reports back to the originator, as well as hybrid queries [50], where each peer recursively forwards the query and additionally sends an acknowledgment back to the originator.

### 2.2.2 Kademlia

The Kademlia algorithm [51] is another approach to implement a Distributed Hash Table which is based on a structured overlay network. Its main functionality is to offer participating peers the possibility to store small files in the overlay and to retrieve them at a later point in time. The main difference to the previously described Chord algorithm is the symmetric relationship between neighbors in the Kademlia overlay network, which enables peers to exploit any messages exchanged between them for the stabilization of the overlay structure. While Kademlia is less well understood than other DHT algorithms, it has been successfully tested in different deployed applications [52, 53].

#### System Description and Routing Table Structure

Kademlia also uses $m = 160$ bit identifiers for peers and files in the overlay. The distance between two such identifiers, however, is based on the XOR metric. That is, given two peers with identifiers $x$ and $y$ Kademlia defines the distance between these two peers as their bitwise exclusive or (XOR), $d(x, y) = x \oplus y$ which is interpreted as an integer. Most benefits of Kademlia result from the following properties of the XOR metric:

- $d(x, x) = 0$, $d(x, y) > 0$ if $x \neq y$: It is ensured that the distance between two peers is always positive.
- $\forall x, y : d(x, y) = d(y, x)$: This features implies that the connections between overlay neighbors are symmetric and peers mainly receive messages from peers which they also have in their own routing table.
- $d(x, y) + d(y, z) \geq d(x, z)$: The triangle property holds.
- $\forall x, \Delta > 0\ \exists_1\, y$: $d(x, y) = \Delta$: This unidirectionality ensures that lookups for the same identifier converge along the same path, regardless of where the search is started.

To construct the Kademlia routing table, the overlay neighbors of a peer are chosen according to the XOR metric. For each $0 \leq i < 160$ a peer $x$ stores a list of $k$ peers which is called $k$-bucket. Thereby, all contacts in the $i$th $k$-bucket have a distance between $2^{160-i} \leq d(x, y) < 2^{160-i+1}$ to peer $x$ according to the XOR metric.

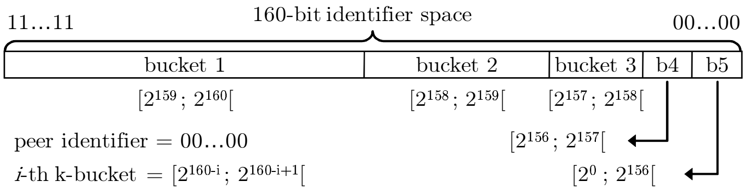


Figure 2.7: *Kademlia routing table for peer with id 00...00*

Figure 2.7 visualizes this concept for a peer $x$ with $id_x$ = 0 or 00...00 in bit notation. The first $k$-bucket contains $k$ peers with a distance between $2^{159}$ and $2^{160}$ to peer $x$. Note that this bucket covers half of the entire identifier space and thus about half of all online peers. In general, a peer can choose $k$ arbitrary peers out of all possible peers for each specific $k$-bucket. The default solution, however, is to sort the $k$-bucket entries by the time of last contact. Whenever a new peer arrives and the corresponding $k$-bucket is full, the least recently seen peer in the $k$-bucket is pinged. If this peer answers, the new peer is disregarded, otherwise the old outdated entry is replaced by the new peer. This mechanism is based on the assumption that the longer a peer has already been online, the more likely it is to stay online in the future.

The remaining $k$-buckets in Figure 2.7 illustrate the main idea behind the Kademlia routing table. Buckets which contain peers close to peer $x$, like buckets 4 and 5 in the example, are used to stabilize the overlay structure, while more distant buckets, like bucket 1, 2, and 3 in the example, are used as shortcuts to enable fast and scalable searches in the overlay network.

### Lookup Algorithm

To locate specific identifiers Kademlia uses a parallel lookup algorithm. The exact procedure is depicted in Figure 2.8. When searching, peer $z$ sends out $\alpha$ parallel queries to the $\alpha$ closest peers which it can find in its own $k$-buckets. Each of these peers then answers with the $k$ closest peers it knows. The searching peer waits for at least $\beta$ peers to answer before it enters the next search step. It then merges all newly learned contacts with the already known contacts and recursively queries the $\alpha$ closest to the searched identifier.

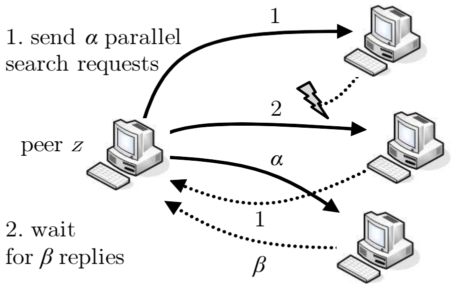

1. send $\alpha$ parallel search requests
1
2
$\alpha$
peer $z$
1
2. wait for $\beta$ replies
$\beta$

Figure 2.8: *Parallel lookup process of Kademlia*

This procedure is repeated until a recursion step fails to return any contacts which are closer to the searched id than those already learned. At this point, the $k$ closest peers found during the search are directly queried for the searched file. An obvious advantage of such parallel queries is that the search will not be delayed as long as there are less than $\alpha - \beta + 1$ timeouts per search step.

A newly joining peer may obtain some overlay contacts from an arbitrary peer already participating in the overlay network and can then perform a lookup for its own id using these contacts as a starting point.

## Dynamic System Evolution

To account for the changes in the overlay network, the Kademlia routing table is dynamically adapted to the current overlay structure. This is done by splitting and merging appropriate $k$-buckets according to the changes in the neighborhood of a peer. An example of the dynamic evolution of the routing table is given in

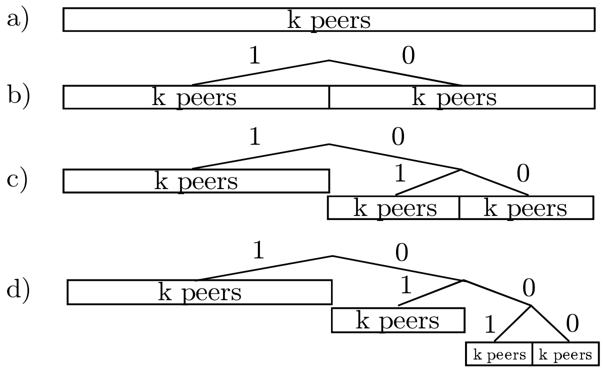


Figure 2.9: *Dynamic evolution of the routing table*

Figure 2.9 for a peer with id = 00...00. The peer initializes its routing table with a single $k$-bucket covering the entire id space, cf. Figure 2.9 a). As soon as the peer has more than $k$ overlay neighbors, it splits its routing table into two separate $k$-buckets, one bucket containing all peers whose id starts with 1 and another bucket containing all peers whose id starts with 0, cf. Figure 2.9 b). From this point on, the only bucket which can be further split is the bucket into which the peer's own id falls. The peer in the example has id = 00...00 and will therefore split the bucket starting with 0 as soon as it contains more than $k$ entries. In this case the bucket would be split into two buckets covering ids which start with 01 and 00, respectively (cf.Figure 2.9 c)). This bucket splitting algorithm is continued as indicated in Figure 2.9 d) as long as the peer learns new contacts in its vicinity, i.e. as long as the overlay is growing in size. As soon as peers leave the network and the overlay becomes smaller, split buckets are merged back together. If, e.g., the buckets starting with 01 and 00 in Figure 2.9 d) together contain less than $k$ entries, they are merged in such a way that the routing table again looks like in Figure 2.9 c).

### Maintenance of the Overlay Structure

In order to maintain the overlay structure in times of churn, the Kademlia protocol extracts information from all messages exchanged between peers. That is, Kademlia uses all traffic between participating peers in order to stabilize the relationship between overlay neighbors. Each time a peer contacts another peer, both peers check their corresponding $k$-bucket and insert the other peer if appropriate. In addition to this, a $k$-bucket is refreshed as soon as there was no change in this bucket for at least an hour. To perform such a bucket refresh, a peer simply issues a search for a random id in the same $k$-bucket.

The redundancy of stored files is maintained in a similar way. Initially, files are stored at the $k$ closest peers, which are called the replication group. If a peer in this replication group did not receive a specific file from another peer within the last 60 minutes, it republishes this file to the entire replication group. To do so, it searches for the $k$ closest peers to the file's identifier and transmits the file to each of them. Those peers store the file locally and update their timers accordingly. If a republishing peer itself is no longer among the $k$ closest peers, it deletes the file from its local storage. In order to avoid that files are stored forever and to guarantee that lost files will be recovered, the original publisher of the file must re-insert the file every 24 hours on default. Depending on the application, longer intervals or alternative maintenance strategies [54] might be more appropriate.

## 2.2.3 Pastry

Pastry [55] is a structured overlay network developed by Microsoft Research and Rice University. It serves as a basis for different kinds of applications. A typical example for such an application is PAST [56], which uses the Pastry overlay to provide a global, persistent storage utility. Files or documents are hashed and persistently stored at $k$ well defined overlay peers, from which they can be retrieved at a later point in time. SCRIBE [57] is another example for an application which uses Pastry to realize a scalable group communication system based on a distributed publish/subscribe service. Interested peers can subscribe to a certain topic by storing their contact information at the Pastry peer which is closest to identifier of the topic.

### Architecture of a Pastry Peer

Similar to Chord, Pastry arranges the participating peers on a circular identifier space which ranges from 0 to $2^{128}-1$. The corresponding 128bit identifiers are interpreted as a sequence of digits with base $2^b$ in such a way that $b$ consecutive bits represent one digit. For $b=2$ a possible identifier for a peer p could, e.g., be 20132301. To establish the overlay structure, each peer maintains connections to three different kind of neighbors. These neighbors are used to build the routing table, the leaf set, and the neighborhood set as shown in Figure 2.10.

The routing table of a Pastry peer consists of up to $\left\lfloor \frac{128}{b} \right\rfloor$ different rows, whereas each row has $2^b-1$ entries. Thereby, for $0 \leq i < \left\lfloor \frac{128}{b} \right\rfloor$ the $i$th row of the routing table does only contain peers which share the first $i$ digits with

Peer ID 20132301

Routing Table

| | | | |
|---|---|---|---|
| -0-3201231 | -1-1203123 | 2 | -3-0021331 |
| 0 | 2-1-301230 | 2-2-112302 | 2-3-032132 |
| 20-0-32102 | 1 | 20-2-31021 | 20-3-20132 |
| 201-0-3120 | 201-1-1302 | 201-2-1230 | 3 |
| 2013-0-213 | 2013-1-031 | 2 | 2013-3-130 |
| 20132-0-22 | | 20132-2-03 | 3 |
| 0 | | | 201323-3-2 |
| | 1 | | |

Leaf Set

| Smaller | | Larger | |
|---|---|---|---|
| 20132232 | 20132123 | 20132330 | 20133101 |
| 20132033 | 20132021 | 20133213 | 20133321 |

Neighborhood Set

| | | | |
|---|---|---|---|
| 20312032 | 32133213 | 01202231 | 11203322 |
| 22303121 | 10232021 | 33023120 | 03211320 |

Figure 2.10: *Pastry routing table for* $b = 2$ *and* $k = m = 8$

the local peer, but differ from it in the $i + 1$th digit. Figure 2.10 shows the routing table for peer 20132301 for $b = 2$ resulting in $2^2 - 1 = 3$ routing entries per row. In the $i$th row the $i$th digit of the local peer is highlighted in gray and the first digit in which the routing entries differ from the local peer is shown between hyphens. Note, that there are possibly many peers which fit into a particular row of the routing table. In principle, the local peer may choose its routing entries randomly from all possible entries. In practice, however, a peer should prefer physically close peers, i.e. peers to which it has a small physical delay. The actual size of the routing table depends on the current number of peers in the overlay. Assuming uniformly distributed peer identifiers, there are approximately $\lceil log_{2^b} n \rceil$ occupied rows in the routing table, where $n$ is the current number of peers in the overlay. The empty entries in the last three rows in the example arise as there are not enough peers with the corresponding distance to the local peer in the overlay.

The leaf set is a symmetric list of the numerically closest peers in terms of their identifiers. That is, each peer maintains an additional list of $k$ overlay neighbors which contains the $\frac{k}{2}$ numerically closest peers with a larger id as well as the $\frac{k}{2}$ numerically closest peers with a smaller id as compared to the peer's own id. This list is used to stabilize the overlay structure and to perform the final step of a search. Finally, the neighborhood set contains a list of $m$ peers which are physically close to the local node. This list is absolutely independent of the numerical identifiers of the peers and also not used for routing purposes. Typical values for the Pastry parameters are $b = 4$ and $k = m = 2^b$ or $k = m = 2 \cdot 2^b$.

## Routing in Pastry

The routing process in Pastry is based on the routing table as well as on the leaf set of a peer. Figure 2.11 illustrates the routing procedure for a peer p which is searching for an id xyz. At first the peer checks if the searched id is covered by its leaf set. If this is the case, it can forward the query to the numerically closest peer in the leaf set which should be able to answer the query. If xyz is not covered by any peer in the leaf set, peer p forwards the query to a peer in its routing table whose identifier is at least one digit closer to the target id xyz as compared to its own id. To reach the searched id as fast as possible, peer p should choose the routing entry which has the longest common prefix with the searched id. In the rare case that peer p does not find any peer in its routing table which shares a longer matching prefix with the searched id than itself, it combines all peers of its routing table, its leaf set, and its neighborhood set into a temporary list. From this list it then chooses a peer which shares at least as long a prefix with the search id as the local peer itself but is numerically closer to it.

In the construction of its routing table, a Pastry peer prefers peers with good latency. For each row of its routing table, a peer chooses the physically closest peers out of all peers which have a numerical distance fitting into the correspond-

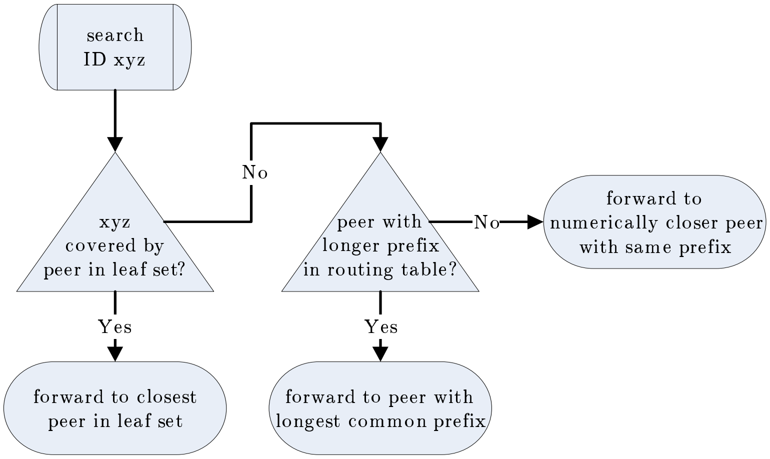

search ID xyz
xyz covered by peer in leaf set?
No
peer with longer prefix in routing table?
No
forward to numerically closer peer with same prefix
Yes
Yes
forward to closest peer in leaf set
forward to peer with longest common prefix

Figure 2.11: *Routing procedure of a Pastry peer*

ing row. At first, a joining node initializes its routing table entries by obtaining adequate entries from other peers. It then periodically contacts the peers in its neighborhood set to obtain and propagate information about physically close peers independent of their numerical ids. With these peers it can then exchange numerically adequate entries to improve its routing table in terms of latency.

The choice of the parameter $b$ determines a trade-off between the size of the routing table $\lceil log_{2^b} n \rceil \cdot \left(2^b - 1\right)$ and the average number of hops required for a search $\lceil log_{2^b} n \rceil$. In the worst case, when many nodes fail simultaneously, the number of routing steps might be linear in the number of online peers. However, this scenario is unlikely as for this to happen all routing entries of all peers involved in the search process have to fail. From a global point of view, the delivery of messages can be guaranteed as long as no peer loses all of its leaf set entries with smaller or larger id simultaneously, i.e. as long as less than $\frac{k}{2}$ peers with consecutive ids fail simultaneously.

### Maintenance of the Overlay Neighbors

The Pastry algorithm is designed for long-lived applications like permanent storage or publish/subscribe which are expected to attract peers with relatively long online times. On these grounds it is assumed that a newly joining peer p already knows the contact information of another peer b which is already part of the overlay and physically close to p. Based on this physical proximity, peer p can initialize its own neighborhood set with the neighborhood set of peer b. Peer p then asks peer b to locate that peer z which is numerically closest to p in the Pastry overlay. All peers on the path from b to z additionally report back to p. Peer p exploits this information to construct its own routing table by copying the entries from the $i$th row of the peer contacted in the $i$th step of the search.

Peer failures are handled in different ways, depending on where they occurred. If one of the peers in the leaf set of a peer fails, the peer immediately contacts the peer with the highest or lowest id in its leaf set, depending on whether the failed peer had a higher or lower id than the local peer. It then asks this peer for its leaf set and updates its own leaf set accordingly. If a peer in row $i$ of the routing table fails, the local peer contacts another entry in the same row of the routing table and asks this peer for an appropriate entry. If no such peer is found the local peer extends its search to further away peers until it finds an appropriate replacement. While the neighborhood set is not used in the process of routing, it is required to exchange information about physically nearby peers. To keep this set up to date, each member is contacted periodically. If a peer does not respond, the local replaces this entry by asking other members of its neighborhood set for appropriate entries.

## 2.2.4 Content Addressable Networks (CAN)

Ratnasamy et al. [58] observed that an essential part of any p2p-based network is to provide an index for files stored in the overlay. Algorithms which specialize on storing, retrieving, and deleting files in the overlay have become known as content addressable networks (CANs). In principle, a CAN performs the same tasks as a distributed hash table. The main difference to other DHTs, however, is that peers are assigned random coordinates in the overlay, while files are still hashed and stored at the closest peer in the overlay.

## Overlay Topology and Routing

In CAN peers are located in a $d$-dimensional Cartesian coordinate space on a $d$-torus. Thereby each peer is responsible for a specific subpart of the coordinate space, which is called the zone of the peer. Figure 2.12 shows an example of a 2-dimensional CAN overlay which is partitioned into 21 different zones. Note, that the coordinate space wraps on a $d$-torus, which is not shown in the figure. To insert a new file into the overlay, it is mapped onto a point P using a uniform hash function. Each peer is responsible to store all files which are mapped into its zone. Other peers can then retrieve a desired file by locating the peer which manages the corresponding zone.

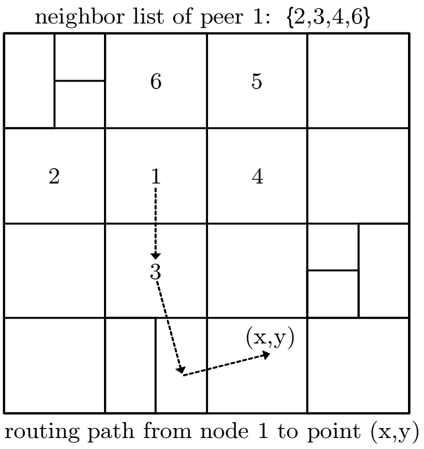

neighbor list of peer 1: {2,3,4,6}
6
5
2
1
4
3
(x,y)
routing path from node 1 to point (x,y)

Figure 2.12: *Structure of a two dimensional CAN overlay network*

To enable the routing of messages, each peer maintains pointers to a set of neighbors in the overlay. Thereby in a $d$-dimensional space, two nodes are neighbors if their zones overlap along $d - 1$-dimensions and touch each other along 1 dimension. The overlay neighbors of peer 1 in Figure 2.12 are 2, 3, 4, and 6, since their zones overlap along one dimension. Note that peer 5 is not a neighbor of peer 1, since its zone touches the zone of peer 1 along all dimensions but does not overlap along any dimension. When searching in the overlay space a peer utilizes its neighbor list. Figure 2.12 shows the routing path of peer 1 searching for a specific point (x,y). To reach this point, the query is recursively forwarded in a greedy way to the neighbor which is closest to the destination. Note that in the case of a failure, e.g. if a neighbor is offline, there are multiple paths leading through different neighbors which can be used to route the query to its destination. As shown in [58] the average path length is in $O(n^{\frac{1}{d}})$, where $n$ is the current number of peers in the overlay.

## System Evolution under Churn

Bootstrapping in CAN is done using a simple domain name system (DNS) based approach. Each CAN overlay has an associated domain name. If a new peer p wants to join the overlay, it performs a DNS lookup of this domain and retrieves the address of one ore more bootstrap servers. Peer p then chooses a random point in the coordinate space as its own location and searches for the peer which is currently responsible for the corresponding zone. It then contacts this peer and the old zone is split into two new zones. Thereby the order along which dimensions a zone is split is predetermined. In a 2-dimensional space, zones are first split along the x-axis and then along the y-axis. In Figure 2.13 the joining peer 4 randomly chose a point in the zone of peer 3, which is thus split into two halves managed by peer 3 and peer 4, respectively.

When a peer leaves the overlay network other peers have to take over the responsibility for the now unoccupied zone. This is done by merging appropriate zones. If possible, a single peer merges its zone with the unoccupied zone

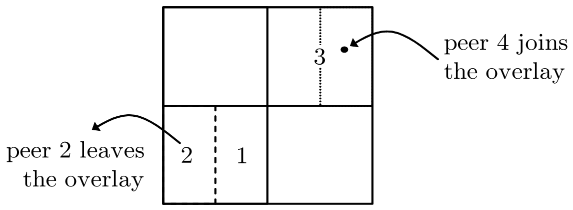


Figure 2.13: *Maintenance of the CAN overlay network structure*

into a new valid zone. Otherwise the neighbor with the smallest zone takes over the unoccupied zone and temporarily handles both zones, which might e.g. be L-shaped. If necessary a zone-reassignment algorithm is triggered later. In Figure 2.13 peer 2 leaves the overlay network and peer 1 subsequently takes over the old zone of peer 2.

## Design Improvements and Open Issues

A general problem of structured overlay networks is that proximity in the overlay does not directly reflect physical proximity. The main advantage of CAN is that it offers different possibilities to consider the physical proximity of peers in the construction of the overlay. One possible approach [59] is that each peer pings a set of landmark servers and chooses its position in the CAN overlay according to the measured delays. Another possibility [60] is to choose coordinates in a 2-dimensional CAN overlay according to the physical location of the machine the software is running on, e.g. by taking the machine's GPS coordinates. Both approaches greatly reduce the latency of routing in the overlay.

The path length in terms of overlay hops can also be reduced by using more dimensions $d$ or by constructing several CAN overlays in parallel, whereas each peer is assigned a different zone in each coordinate space. However, both modifications come at the cost of increased complexity and per node state. Zone overloading, where multiple peers are responsible for the same zone, also reduces the average path length while simultaneously improving the fault tolerance, again at the cost of an increased system complexity.

While CAN gives good theoretic insights into the design of structured overlays in general, it also raises many questions and problems in practice. In contrast to other overlay topologies, the structure of the CAN overlay depends on the order in which peers join and leave the network. That is, for a given set of participating peers and their corresponding locations in the overlay, the exact layout of the individual zones is not fixed but depends on how the system evolved over time. This often results in an unbalanced distribution of zones, which becomes even more critical when many peers choose a position close to each other within a small area of the overlay. In such a case searches will not take the optimal path but might already require multiple overlay hops just to leave the local area.

## 2.3 Hybrid Architectures

P2p networks which are deployed in a practical environment are often built for a special purpose. Their architecture typically is a mixture of different concepts well adapted to the intended use. In general, those networks can be subsumed under the term hybrid approaches.

The most common area of application of such networks is file-sharing or content distribution, which is usually realized using a two step approach. In the first step the participating peers locate other peers in the network which offer or are interested in the same content. In the second step the peers manage the exchange of data among each other by organizing who will download what from whom and when. The basic principle is multiple source download, where files are divided into small parts called chunks and peers can issue multiple download requests to different providing peers which serve several downloaders in parallel.

In this chapter we briefly describe the basic mechanisms of overlay networks offering a variety of files which are indexed by a central server (cf. Chapter 2.3.1), as well as of overlays which specialize in the efficient distribution of just one single file, cf. Chapter 2.3.2. Since hybrid approaches are by far not limited to file-sharing applications, we also discuss SuperPeer-based overlay networks (cf. Chapter 2.3.3), which use concepts like NAT-traversal and overlay re-routing to enable applications like large scale distributed VoIP platforms.

## 2.3.1 Client-Server-based Overlay Architectures

In a client-server based overlay network, peers share and download files among each other, while an index of all shared files is compiled and distributed by one or possibly more central servers. While Napster [61] was the first p2p network based on this principle, the eDonkey network [62, 63] became far more popular in terms of active users, especially in southern Europe. As shown in Figure 2.14,

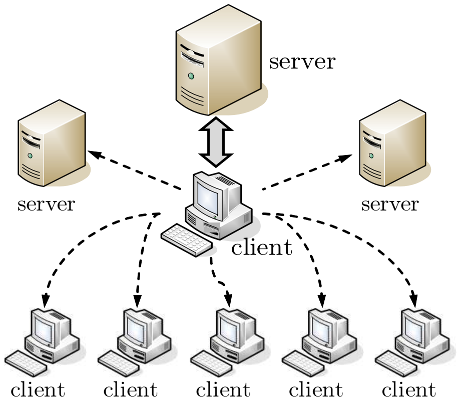


Figure 2.14: *Basic architecture of the eMule network.*

an eDonkey client connects to an index server and requests a list of all peers which already share or are also interested in the same files the client intends to download. If the index server does not return any or too few matching results, the client may resubmit the same query to another eDonkey index server. In the eMule network [64], an extension to the original eDonkey network, the pool of central servers was replaced by Kad [65], a structured overlay network based on the Kademlia protocol. As soon as the peer has obtained a list of other peers interested in the same content, it connects to these peers and starts to exchange the desired files as well as further information about who is sharing the same content.

The size of the files shared among the peers usually goes from some megabytes, e.g. for audio files, up to the gigabyte range, e.g. for video files. For this reason each file is divided into several smaller parts called chunks, which again are divided into individual blocks. Figure 2.15 illustrates this procedure us-

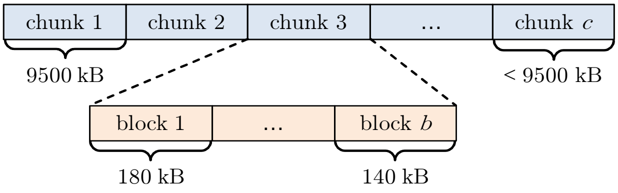


Figure 2.15: *eMule file as divided into chunks and blocks.*

ing parameters found in the source code of eMule 0.48a [64]. The file is split into $c$ chunks, whereas the first $c - 1$ chunks are of size 9500 kB and the last chunk contains the remaining part of the file. Each chunk is in turn split into $b-1$ blocks of size 180 kB and one block of size 140 kB. This way, a peer does not have to download the entire file to contribute to the dissemination of the file but may start sharing as soon as it obtained the first chunk of the file. Thus, this concept enables the process of multiple source download where each peer may download and upload different parts of the file from and to multiple peers at the same time. In this context, each peer maintains an upload queue, which is based on a simple credit system and determines which requesting peer will be served next. The position of a peer in this queue is determined by its score, whereas the score is calculated as $score = (rating \cdot time\ in\ queue\ in\ seconds)/100$. The initial $rating$ of a peer is 100 and will be multiplied by a value between 1 and 10 according to the peer's credits as well as by a value between 0.2 and 1.8 depending on the file priority. Thereby credits can be earned by uploading chunks to the specific peer.

### 2.3.2 Content Distribution Networks (CDN)

Content distribution networks are overlay networks which are established in order to distribute one single file as fast and as efficiently as possible. The most prominent example is the BitTorrent protocol [66], which divides the offered file into 256 kB chunks consisting of 16 kB subpieces and then coordinates the exchange of these chunks among the participating peers. The basic principle of BitTorrent is illustrated in Figure 2.16. At first, each peer interested in a particular file needs

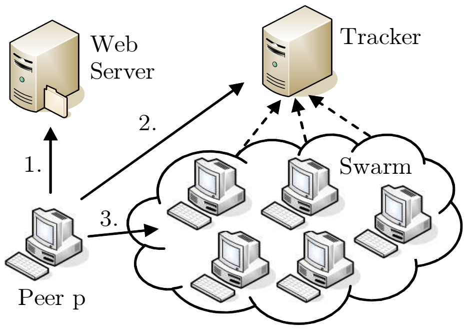


Figure 2.16: *Basic principle of the BitTorrent overlay network.*

to obtain the corresponding .torrent file, usually from a web server as shown in step one of the figure. This .torrent file contains meta-information including the size of the file, the hash values of the individual chunks, and the contact address of the responsible tracker. The tracker is a central entity which keeps track of all peers currently sharing the same file and which is used by the peers to find other peers to download from, cf. step two in the figure. Note, that the central tracker does not necessarily have to be a single server. In the latest version of BitTorrent it is, e.g., replaced by a distributed Kademlia network. As soon as a peer obtained a list of other overlay peers it joins this so called swarm (cf. step three) and starts to exchange data with the other peers in the swarm.

Thereby, the BitTorrent protocol exactly specifies how the exchange of the chunks is arranged among the peers. In particular, the chunk selection strategy is based on the following principles:

- Strict priority: All subpieces of one chunk have to be downloaded before any subpiece of any other chunk may be downloaded.
- Random first chunk: The first chunk to be downloaded is always selected randomly from all possible chunks in order to avoid that rare chunks are disseminated slower than popular chunks.
- Rarest first: Apart from the first chunk, a peer always downloads the chunk which it believes to be the rarest in its swarm.
- Endgame mode: The last subpieces of the last missing chunk are requested from multiple sources simultaneously and then downloaded from the fastest source. This avoids the problem of starvation which occurs when downloading the last subpiece from a slow peer.
- Tit-for-tat: Peers prefer to upload to peers from which they currently download or have successfully downloaded from in the past. This mechanism is closely related to game theory, whereas a peer refuses to upload to, i.e. chokes, all peers per default and only unchokes those peers which offer the best download rate. To enable new or more suitable peers to join or improve the dissemination process, optimistic unchoking is applied, where a random peer is served regardless of its previous contributions.

While the BitTorrent protocol proved to be scalable and efficient [67, 68], it still has to struggle with an uneven chunk distribution. This issue is addresses by Avalanche [69, 70] which applies the concept of network coding [71] in the field of content distribution. With network coding, peers do no longer exchange chunks but linear combinations of chunks along with the randomly chosen coefficients. As soon as a peer obtains enough linearly independent chunk combinations it can reconstruct the original file. This way, the rarest chunk problem is avoided and a peer no longer has to keep track of the current chunk distribution in the overlay.

### 2.3.3 SuperPeer-based Architectures

SuperPeer-based overlay architectures have been developed in an effort to combine the advantages of both the classic client-server approach and the p2p paradigm that all peers should have the same functionality. Such approaches exploit the fact that in a p2p network not all peers are equal but heterogeneous in many aspects like available bandwidth, offered processing power, or their online session times. That is, peers which are more reliable and provide more resources than other peers become SuperPeers and as such act as servers to other peers. This

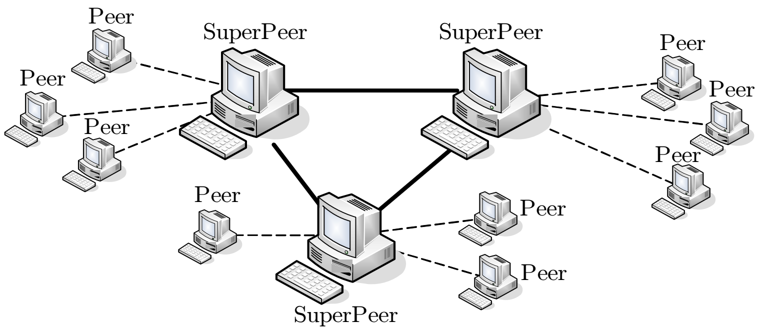


Figure 2.17: *Example of the SuperPeer architecture.*

way, they simultaneously provide the search efficiency of a centralized solution as well as the robustness to attacks provided by distributed architectures. Figure 2.17 shows an example of a SuperPeer-based overlay topology. Each SuperPeer acts as a server for a number of regular peers, while the SuperPeers themselves maintain separate connections to each other. Thereby, the interconnection between the SuperPeers may be realized in different ways like using a fully meshed topology, a structured overlay network, or a simple gossip based flooding mechanism. According to a measurement-based analysis [72], the latter approach is, e.g, used in KaZaA [73], a very popular but proprietary SuperPeer-based overlay network.

Depending on the intended purpose of the particular overlay network, its SuperPeers will be selected for different reasons. In real-time applications, SuperPeers might be chosen based on their latency to other peers, while in a very restrictive environment the number of open ports might be the determining factor. In general, there are various design issues to be solved in order to decide which peers to promote to SuperPeers. The decision might, e.g, be made by a central control entity or based on a distributed algorithm. If a global view of the network is available, the top $x$ percent of all peers can directly be selected as SuperPeers. Otherwise peers might promote themselves to SuperPeers depending on whether they meet some given requirements or not. Furthermore, the number of peers served by an individual SuperPeer also significantly influences the overall performance of the system. A good overview of how to approach these questions is given in [74].

Aside from file-sharing, SuperPeer-based architectures may also be used as the basis for a variety of other applications. Skype [35], e.g., uses a SuperPeer-based overlay topology to realize a distributed Voice-over-IP (VoIP) service. In this context, the SuperPeers are not only used to take off the load of the central index server and to thus provide a scalable service, but also for firewall traversal and call re-routing as shown in Figure 2.18. Regular peers which are behind a firewall or Network Address Translation (NAT), like peer A and peer B in the figure, are usually not able to open a direct connection to each other. Therefore a SuperPeer can be used to establish an indirect communication channel by relaying the packets exchanged between those two peers. That is, independent of the actual routing path on the IP-layer, packets from peer A to peer B and vice versa are re-routed in the overlay network via a SuperPeer. Note, that this mechanism

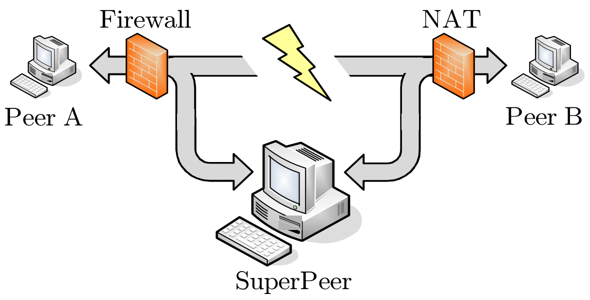


Figure 2.18: *Overlay re-routing in SuperPeer-based network topologies.*

is not limited to the case of a missing direct connection, but may also be used if the regular path between two peers is congested while both peers still have a good connection to the SuperPeer. This is another good example of how overlay architectures can be and already are used to shift the intelligence from the center to the edge of the network.

# 3 Performance Analysis of Structured P2P Networks

Structured overlay architectures have been proposed in an effort to resolve the problems of the classic client-server paradigm. They offer a greater flexibility, are robust against single points of failure, and most importantly are scalable in the sense that each new participant also adds new resources to the network. However, besides these benefits there are also new challenges which arise as a consequence of the distributed nature of such systems. Their building blocks (the participating peers), e.g., are not as reliable as the components of a high performance server. In spite of the instabilities both the structure of the overlay as well as the availability of stored resources must be maintained.

Thus, the performance of such systems has to be analyzed in more detail, before they can be applied in a corporate environment. To be able to guarantee certain levels of quality, we need to understand their limits and derive important performance measures like failure probabilities or search delay quantiles. In this chapter, we will first motivate that in this context not only the scalability in terms of system size, but rather stochastic influences like the behavior of the user can cause severe problems. This will be followed by an overview of current issues in the field of structured overlay networks as well as different approaches proposed in literature to solve them.

We provide a detailed mathematical analysis of the lookup process in Chord, the most prominent structured p2p algorithm. In particular we derive the peer distance distribution as well as the search delay distribution in dependence of varying network conditions. To investigate the influence of stochastic user behavior on the robustness of the system, we show the limits of ring-based overlay topologies by calculating the probability to lose the structure of the overlay. Finally, we evaluate the performance of more complex structures based on the XOR metric by large scale simulation. We reveal the inherent problems of such systems and propose modifications, which significantly improve their performance.

## 3.1 Functional and Stochastic Scalability

Today, scalability is the most important performance measure a carrier grade system has to withstand. It indicates whether a system is going to work on a large scale or not. In general, the question scalability asks, is: If a solution works for 10 customers, does it also work for hundreds, thousands, or even millions of customers? So far, scalability mainly referred to the mere size of a system. Most studies were intended to determine if a system at hand does work for growing customer clusters. We summarize this kind of analysis under the term functional scalability. It tells us whether the fundamental logic of a solution is scalable.

The mere size of a system, however, is not the only factor in terms of scalability a running application has to cope with. There are more and more system parameters having a stochastic character. Consider, e.g., the stochastic behavior of customers. There are numerous different random variables describing values like the inter-arrival time, the mean on-line time, and the query rate of customers of large scale systems. In p2p networks this stochastic behavior is defined as the autonomy of the participating peers, i.e., the peers may join or leave the system arbitrarily. This leads to the requirement to evaluate p2p algorithms with respect to the stochastic on-line behavior, which is summarized under the term "churn" [75]. This unpredictable stochastic behavior of the end user results in a highly dynamic evolution of the p2p network and thus has a significant impact on the functionality of the system [76]. The customer, however, is not the only factor introducing probabilistic properties into the system. A running system also faces

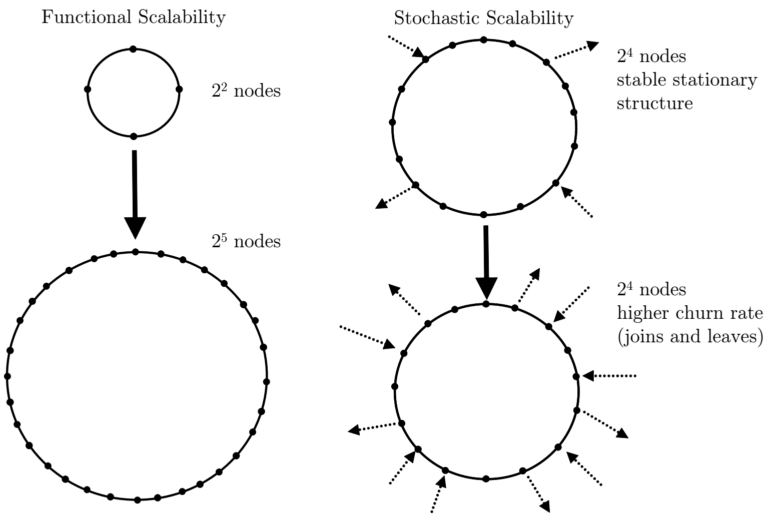


Figure 3.1: *On the definition of stochastic scalability*

stochastic network loads, probabilistic variations in traffic volumes, and random transmission delays, to name just a few. Thus, in order to provide stochastic scalability, p2p networks with resilience requirements have to be able to survive in case of stochastic breakdowns. Stochastic scalability can be analyzed combining methods and techniques of both probability theory and performance analysis.

Figure 3.1 visualizes the difference between functional and stochastic scalability. The functional scalability verifies whether the interworking logic is extendable to larger crowds of customers. It mathematically analyzes whether the functionality of a system, like the search delay in the indicated Chord ring, also works for a large number of customers. Stochastic scalability on the other hand tries to verify whether a system can sustain the stochastic behavior of its components. It investigates whether a system can cope with the non-deterministic arrival, departure, and query times of the participating customers. With respect to our Chord ring example, stochastic scalability raises the question whether a system which can sustain minor churn rates also works under extreme high churn rates? That is, we want to know how long the average customer has to stay online in order to guarantee the functionality of the running system.

In the end a successful system must be scalable in both a functional and a stochastic way. Without functional scalability a system will collapse under its own size, without stochastic scalability a system will collapse under the random variations of its components. In the following, our goal is therefore to better understand the dynamics of large scale overlay networks such as structured p2p systems. If we want to build reliable large scale information sharing platforms based on p2p mechanisms we need to master the complexity of such systems. Investigating both the functional and the stochastic scalability, we will be able to get those systems under control and achieve carrier grade availability systems in a resource-efficient but also simple manner.

## 3.2 General Approaches and Related Work

Structured p2p networks [45, 51, 55, 58] are often regarded as a further development of unstructured p2p systems. The initial design of structured overlays, however, shows significant scaling and performance problems. In this context, Ratnasamy et al. [77] give a good overview of open questions in the field of DHT-based overlays. Among others, the problems in terms of functional and stochastic scalability, the heterogeneity of the participating peers as well as the general resilience of the overlay structure add up to a new field of open research.

There are several different approaches to evaluate the performance of structured overlay networks. Loguinov et al. [78] study the routing performance, the diameter, and the degree of different structured p2p algorithms using methods from graph theory. While each architecture is best suited for a particular scenario, they propose de Bruijn graphs as a good compromise for structured p2p networks. Ramabhadran et al. [47] evaluate the life time of replicated resources using a simple Markov model which is based on the gambler's ruin problem. The redundancy is limited by both the total number of replicas which can be stored in the network and the repair overhead needed to create new replicas. Due to the complexity of structured overlays, most studies, however, are based on discrete event simulators [79], whereas the simulation of large scale p2p networks still requires appropriate abstractions [80]. As a final step, proof-of-concept studies can also be realized by prototype emulation in a world wide testbed like PlanetLab [81].

In regard to overlay routing, one of the main problems is that proximity in the overlay does not reflect physical proximity. Jain et al. [82] studied the relative delay penalty, which is a measure of the additional packet delay introduced by the overlay, and found a latency stretch that is longer by a factor of two or more as compared to optimal routing. To address this issue, Proximity Neighbor Selection (PNS), Proximity Route Selection (PRS), and Proximity Identifier Selection (PIS) were proposed and evaluated [83]. Dabek et al. [49] showed that such latency optimizations can reduce the time required to locate and fetch data in the overlay by a factor of two. Chun et al. [84] present a more generalized model for neighbor selection and conclude that the choice of neighbor selection algorithms drives a tradeoff between performance and resilience to attacks. This effect can be reduced by including node liveness information [85], where each node attempts to populate its routing table with neighbors which tend to stay alive for a long time

Liben-Nowell et al. [86] analyzed the evolution of continually running p2p networks and observed [76] that the rate at which peers join and leave the network is the most informative performance measure as it might cause the network to split, create loopy cycles, or entirely destroy the structure of the overlay. To understand how random departure decisions of end-users affect the connectivity of p2p networks, Leonard et al. [87] used a mathematical approach to investigate the general resilience of random graphs to lifetime-based node failures and derived the expected delay before a user is forcefully isolated from the graph. Li et al. [88] compared the performance of distributed hash tables under churn using discrete event simulations and derived a performance vs. cost framework for evaluating DHT design tradeoffs [89].

In the research community, Chord has become the most studied algorithm, possibly since its ring structure is comparatively easy to analyze. Kong et al. [90] derived an analytical framework for characterizing the performance of DHTs under random failures. They showed that ring-based overlays like Chord as well as XOR-based topologies like Kademlia are capable of routing to a constant fraction of the network even if there is a non-zero probability of random node failures. The original Chord algorithm, however, lacks numerous features which are inherent to other structured overlay algorithms. As a consequence thereof, numerous work exists which presents modifications to the original Chord algorithm. Mesaros et al. [46] address Chord's lack of symmetry by introducing symmetric neighbor lists as well as a symmetric finger table structure. This improves the possibilities for proximity neighbor selection, increases the resilience to node failures, and decreases the lookup failure rate under churn. While most studies of DHTs under such churn conditions depend on simulation as the primary investigation tool, Krishnamurthy et al. [91] presented a complete analytical study of churn using a master-equation-based approach. From this they derived the fraction of failed or incorrect successor and finger pointers for any rate of churn and stabilization as well as any system size. It follows that more maintenance overhead must be invested in times of higher churn. Kunzmann et al. [21] evaluated the corresponding costs to maintain the topology structure of Chord and to keep the routing entries of the participating peers up to date. They demonstrated that the number of peers with errors in their list of successors is independent of the size of the overlay ring, but is heavily influenced by the online session duration of peers. By taking up the symmetric neighbor lists from [46] and by sending notification messages as soon as a node observes a topology change in its neighborhood, the stabilization algorithm of Chord was modified to handle high churn rates. Independent of the stabilization algorithm, there still remains a risk of losing all successors or splitting the overlay into two separate rings. A redirection mechanism [14] was proposed to deal with such scenarios. It is able to automatically recover from a partitioning of the overlay network.

Mahajan et al. [92] observed that statically configured overlays lead to high costs in the average case and poor performance under worse than expected conditions. They developed a self-organizing algorithm for Pastry-based systems which continuously monitors the environment and automatically adapts the maintenance overhead to the observed conditions. Castro et al. [93] further evaluated the performance of Microsoft's Pastry implementation using large scale simulations and injected real traces of node arrivals and departures which were gained by measurements in deployed p2p systems. They also suggest to adapt the maintenance traffic dynamically since failure rates vary significantly with both daily and weekly patterns. In the regarded scenarios MSPastry was able to provide a reliable overlay structure with a maintenance overhead of less than half a message per second per node. Lam et al. [94] further developed these results by regarding the performance of hypercube-based overlay architectures in general. Wang et al. [95] evaluated the resilience of CAN under failures using a Markov-chain based approach for systems with relatively stable size and uniformly distributed nodes. Based on their results, they propose to add finger neighbors to CAN following the small-world model [96], which significantly improves the performance in terms of the average path length. Wu et al. [97] followed a more general approach and compared different lookup strategies like recursive, iterative, and parallel queries independent of the DHT architecture. From this, they derived design guidelines for the development of new overlay topologies. Zoels et al. [98] provided a cost model for hierarchical overlay structures which are composed of superpeers and leafnodes. This specific overlay structure is especially interesting for highly heterogeneous environments, where the weaker nodes become leafnodes which use the stronger superpeers as proxies.

Recent studies increasingly deal with the maintenance of replicas in structured overlay networks. In general, there are two popular redundancy schemes: simple replication and erasure coding, where each object is divided into $n$ differently encoded fragments, whereas any $m < n$ fragments suffice to recreate the original data. Rodrigues et al. [99] give a good comparison of both schemes. In particular, they show that in some cases the benefits from coding are limited and may not be worth the additional effort. Bhagwan et al. [100] quantify the storage overhead required to deliver a specified level of availability. They conclude that high availability under churn is best achieved by applying erasure coding while adding additional redundancy in terms of replication to the system. Datta et al. [48] perform a Markovian time-evolution analysis to compare different maintenance strategies under churn. Assuming exponential online and offline durations, they derive the probability mass function of the number of replicas available in steady state. Sit et al. [101] show that reactively creating new replicas by responding to failures in the system leads to bandwidth spikes at the peers. They propose to constantly create new replicas according to a user-specified bandwidth limit in order to smooth the bandwidth usage over time. Finally, Rhea et al. [75] discuss different algorithms to handle churn in a DHT. Experiments in a testbed environment using ModelNet as a network emulator showed that it is possible for a DHT to operate at churn rates which are higher than those observed in deployed p2p systems.

## 3.3 Delay Analysis of Chord-based Overlay Networks

So far, in best effort file-sharing systems the search delay was not really critical to the end-user since file download time exceeded the preceding lookup time of the files location by magnitudes. In structured p2p networks, however, the time needed to complete a search for resources stored in the overlay is the most important performance measure. Real-time applications with certain quality of service demands, like VoIP telephony, chatting, or instant messaging are dependent on the time needed to find their communication partner. By design, DHT based p2p algorithms are able to retrieve information stored in an overlay network consisting of $n$ peers by using $O(\log_2(n))$ messages to other peers. This statement, however, is very vague, since it only tells us the order of magnitude of the search delay and does not provide us with sufficient details on search duration statistics.

In addition to the functional scalability in terms of the system size, the highly probabilistic physical link delay also strongly influences the performance of searches in such p2p overlay network. As a first step toward analyzing the performance of structured p2p networks, we therefore evaluate the impact of the system size as well as the impact of the network delay variation on search times in DHT based p2p systems [26]. The main goal is to prove functional and stochastic scalability in very large Chord rings [24], to be able to guarantee certain quality of service demands in large peer populations. The enormous complexity of such systems makes an evaluation by simulation on packet level rather intractable. We therefore deduce an analytical performance model for real-time applications based on the Chord algorithm. While the calculation of the mean of the search duration is quite straightforward, the computation of the quantiles of the search duration is more complex. The quantiles, however, have an important impact on the quality of service experienced by the end user. Making some plausible assumptions, we therefore derive the entire delay distribution function. To capture the influences of the physical path delay, the impact of network delay variation is taken into consideration as well.

### 3.3.1 Computation of the Peer Distance Distribution

The basic idea of the search delay analysis is to first derive the distribution of the number of overlay hops it takes to contact random overlay peers and to then calculate the delay of the corresponding overlay paths. We therefore use the following random variables:

$T_N$: describes the delay of a query packet, which is transferred from one overlay peer to another overlay peer

$T_A$: represents the time needed to transmit the answer from the peer (having the answer) back to the originator of the search

$T$: describes the total search duration from the initial query packet until the answer arrives at the searching peer

$X$: indicates how many times a query has to be forwarded until it reaches the peer having the answer. $X$ will be denoted as the peer distance

$H$: number of overlay hops needed to complete a search, i.e. the number of forwards of the query plus one hop for the transmission of the answer

$n$: size of the Chord-ring

Note, that we distinguish between $T_N$ and $T_A$, as the size (and therefore the delay) of a search packet and an answer packet may be unequal. The answer might, e.g., consist of multiple packets containing a detailed reply to the query. In the first step, we compute the probability distribution of the peer distance $X$. Since according to the Chord algorithm a search is recursively forwarded to the closest finger, we are able to calculate the number of hops needed to reach the peer, that answers a specific query. From this, we derive the probability $p_i = P(X = i)$ that the searched peer is exactly $i$ hops away from the searching peer.

**Special Case of Binary Exponential Peer Populations**

To provide an overview, we start with Chord rings whose size is a power of 2. In an overlay network of this specific size $n = 2^k$, $k = \log_2(n)$ is an integer and each peer has $k$ distinct fingers. Thereby, the $i$*-th* finger of a peer $z$ is used for searches for all peers whose corresponding hash values fall in the region between $[id_z + 2^{i-1}, id_z + 2^i - 1]$ where $id_z$ is the hash value of peer $z$. This can be illustrated using the example in Figure 3.2. In this context the 4*th* finger of peer 1, which is pointing to peer 9, is responsible for all peers between $[9, 16] = [1 + 2^{4-1}, 1 + 2^4 - 1]$.

Taking this into account, we construct Table 3.1 consisting of four columns. The first column represents the peer distance $X$. The second column states the

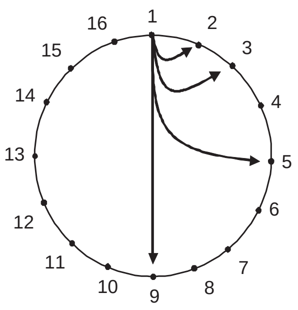


Figure 3.2: *Finger-table of peer 1 in a 16 peer Chord ring*

number of hops $H$ needed to complete a search. In the case of $X = 0$ the searched document is directly stored at the searching peer. A search answered locally likewise requires 0 hops. To complete a search answered by a peer that is $X > 0$ hops away, however, we need $X$ hops to reach that peer and one additional hop to send the answer back to the originator. Altogether $X + 1$ hops are needed to perform this kind of search. Column 4 finally describes the random variable $T$ representing the time needed to complete such searches by adding $X$ times the delay of a forwarded query packet plus the time needed to transmit the answer back to the originator.

The probability $p_i = P(X = i)$ in column 3 is governed by the following theorem:

**Theorem.** *The probability that the searched peer is exactly $i$ hops away from the searching peer in a Chord ring of size $2^k$ (and thus with log$_2(2^k) = k$ fingers) with symmetric search space and uniformly distributed keys is*

$$p_i = P(X = i) = \frac{\binom{k}{i}}{2^k} \tag{3.1}$$

*Proof.* We argue by induction.

Table 3.1: *Peer distance distribution and search time*

| $X$ | $H$ | $P(X=i)$ | search time $T$ |
|---|---|---|---|
| 0 | 0 | $p_0 = \frac{\binom{\log_2(n)}{0}}{n}$ | 0 |
| 1 | 2 | $p_1 = \frac{\binom{\log_2(n)}{1}}{n}$ | $T_A + T_N$ |
| 2 | 3 | $p_2 = \frac{\binom{\log_2(n)}{2}}{n}$ | $T_A + T_N + T_N$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| i | i+1 | $p_i = \frac{\binom{\log_2(n)}{i}}{n}$ | $T_A + \sum_{x=1}^{i} T_N$ |
| $\log_2(n)$ | $\log_2(n)+1$ | $p_{\log_2(n)} = \frac{\binom{\log_2(n)}{\log_2(n)}}{n}$ | $T_A + \sum_{x=1}^{\log_2(n)} T_N$ |

*Basis:*

For $k = 0$, in a ring with $2^0 = 1$ node, there is exactly $1 = \binom{0}{0}$ node, that is 0 hops away from the only peer $z$. Therefore $p_0 = \frac{\binom{0}{0}}{2^0}$.

For $k = 1$, in a ring with $2^1 = 2$ nodes, there is exactly $1 = \binom{1}{0}$ node, that is 0 hops away from peer $z$ and exactly $1 = \binom{1}{1}$ node, that is 1 hop away from peer $z$. Therefore $p_0 = \frac{\binom{1}{0}}{2^1}$ and $p_1 = \frac{\binom{1}{1}}{2^1}$.

*Induction hypothesis:*

Assume the theorem is true for $y \leq k$

*Induction step:*

Prove the theorem is also true for $k+1$

To calculate the number of peers that are $i$ hops away from a peer $z$ in a Chord ring of size $2^{k+1}$, we divide the Chord ring into two parts consisting of the first $2^k$ and the last $2^k$ peers respectively. We then calculate the number of peers that are $i$ hops away from peer $z$ in those two parts of the original ring and simply add those two numbers up.

First $2^k$ nodes: In a Chord ring of size $2^{k+1}$ a peer $z$ has $k+1$ fingers. The first $k$ fingers are responsible for the first $2^k$ nodes. By induction hypothesis there are exactly $\binom{k}{i}$ peers that are i hops away from peer $z$ in this part of the ring.

Last $2^k$ nodes: The $(k+1)$-th finger covers the remaining $2^k$ peers in the original Chord ring. By induction hypothesis there are exactly $\binom{k}{m}$ peers that are m hops away from the $(k+1)$-th finger in this part of the ring. Since the $(k+1)$-th finger is exactly 1 hop away from peer $z$, there are $\binom{k}{i-1}$ peers in this part of the ring that are $i$ hops away from peer $z$ (one hop to reach the finger-peer and $i-1$ hops to reach the corresponding peer).

Altogether there are $\binom{k}{i}+\binom{k}{i-1} = \binom{k+1}{i}$ peers that are exactly $i$ hops away from peer $z$. Since there are $2^{k+1}$ peers the probability that another peer is exactly $i$ hops away from peer $z$ is $p_i = \frac{\binom{k+1}{i}}{2^{k+1}}$.

*Conclusion:*

Together, the basis and the induction step imply that the theorem holds for all possible cases, i.e., in a Chord ring of size $n = 2^k$ the probability that the searched peer is exactly $i$ hops away from the searching peer $z$ is

$$p_i = \frac{\binom{k}{i}}{2^k}.$$

$\square$

This result is also consistent with [78], where the Gaussian peer distance distribution assumed in [45] was shown to actually stem from a binomial distribution.

**Arbitrary Number of Peers**

So far we considered the special case of a binary exponential peer population. We now extend the model to an arbitrary number of peers under the same assumptions made before. In a Chord ring of arbitrary size $n$ we have $k = \lceil \log_2(n) \rceil$ distinct fingers. That is, a Chord ring of this size maintains just as many different fingers as a Chord ring of size $m = 2^k$, the next largest power of 2. Since we are assuming serially numbered peers, the *i-th* finger of a peer $z$ still points to the same peer $id_z + 2^{i-1}$ for $i = 1$ to $\lceil \log_2(n) \rceil$. In other words we can compare Chord rings of arbitrary size $n$ to Chord rings of binary exponential size $m$, except that $m - n$ peers are missing between the last finger and the searching peer itself.

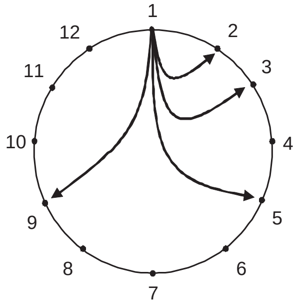


Figure 3.3: *Finger-table of peer 1 in a 12 peer Chord ring*

Figure 3.3 illustrates this issue for a Chord ring of size 12. The figure resembles Figure 3.2 insofar as searches for peers 1 to 8 originating at peer 1 still require the same number of hops. The only difference is that the last finger pointing to peer 9 is now covering less peers and is thus responsible for less searches. On account of this, we divide a Chord ring of arbitrary size into two parts to calculate the peer distance distribution $X$. Thereby, the first part consists of the first $2^{k-1}$ peers, while the second part includes the remaining peers. In conjunction with the preceding theorem we conclude the following corollary, calculating the number $f_n(i)$ of peers in a Chord ring of arbitrary size $n$ that are $i$ hops away from the searching peer $z$:

**Corollary.** *The number $f_n(i)$ of peers in a chord ring of arbitrary size $n$ (and therefore with $\lceil log_2(n) \rceil$ distinct fingers) that are $i$ hops away from the searching* *peer is:*

$$f_n(i) = \begin{cases} \binom{k}{i}, & \text{\textit{if} } n = 2^k \\ \\ \binom{k-1}{i} + f_{n-2^{k-1}}(i-1), & \text{\textit{if} } 2^{k-1} < n < 2^k \end{cases} \tag{3.2}$$

The corollary exploits the fact that there are no changes in the structure of the first $2^{k-1}$ peers compared to an independent Chord ring of size $2^{k-1}$ and recursively calculates the hops needed for searches covered by the last finger. Note that in the recursive calculation we have to subtract one hop needed to reach the responsible finger. Finally to get $p_i$ in the arbitrary case, $f_n(i)$ will be divided by $n$:

$$p_i = \frac{f_n(i)}{n} \tag{3.3}$$

Note that the theorem as well as the corollary both rely on an abstract model which is based on serially numbered peer ids. That is, in a Chord ring of size $n$ the peer ids are numbered $1, 2, ..., n$, such that peer number $z$ has $id_z = z$. In a real Chord ring of size $n$, identifier space size $N = 2^j$, and equally distributed peers, however, peer number $z$ has

$$id_z = 1 + (z-1) \cdot \left\lceil \frac{N}{n} \right\rceil$$

In this case the assumption that the fingers of peer number 1 are directly pointing to peers number

$$1 + 2^{i-1},\ i = 1, ..., k \tag{3.4}$$

is no longer obvious. Instead, according to the Chord algorithm the fingers of peer number 1 are pointing to the first peers, whose ids are directly succeeding $1 + 2^{j-b},\ b = 1, ..., j$ on the Chord ring, respectively. That is, the fingers are pointing to peers number $z_b$ where

$$z_b = \min\{x : 1 + (x-1) \cdot \left\lceil \frac{N}{n} \right\rceil \geq 1 + 2^{j-b}\},\ b = 1, ..., j \tag{3.5}$$

In the special case of binary exponential peer populations $n = 2^k$ it follows that

$$\begin{aligned} z_b = & \min\{x : 1 + (x-1) \cdot \frac{2^j}{2^k} \geq 1 + 2^{j-b}\} & (3.6) \\ = & \min\{x : x \geq 1 + 2^{k-b}\} & (3.7) \end{aligned}$$

Since $x$ has to be an integer the fingers of peer 1 are thus pointing to peers number $1+2^{k-b}$, $b = 1, ..., k$, which are exactly the same peers as in Equation 3.4 where the peer ids were numbered serially. In the special case of $n = 2^k$ the proof can thus be extended to non-contiguous peer ids.

## 3.3.2 Analytical Model of the Search Delay

As a result of the last section we now know the peer distance distribution $X$. From this we derive the length in hops of the path a particular search-query takes through the network. We also know the probability $p_i$ that a search takes exactly this path. Using these basic relations we can compute the distribution of the search delay as a function of the network delay characteristics. First, the basic relations in our model are illustrated followed by the direct computation of the mean and the variation of the search duration.

The phase diagram of the search delay is depicted in Figure 3.4. The search starts at the left side of the figure. A particular path $i$ is chosen with probability $p_i$ where phase $i$ consists of $i$ network transmissions $T_N$ to forward the query to the closest known finger and one network transmission $T_A$ to send the answer back to the searching peer. Again note, that we distinguish between $T_N$ and $T_A$ as the size and therefore the delay of the answer might be significantly larger then the simple search request itself.

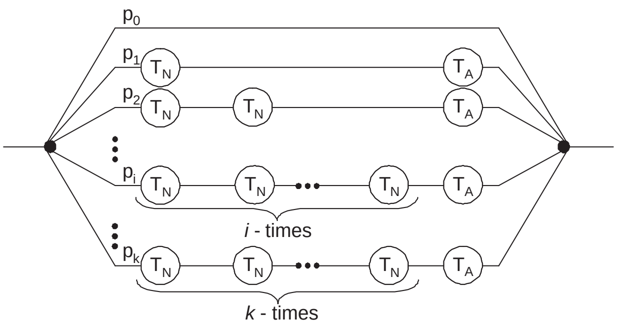


Figure 3.4: *Phase diagram of the search duration $T$*

By means of the phase diagram, the generating function, and the Laplace-Transform respectively can be derived to cope with the case of discrete-time or continuous-time network transfer delay. The generating function of the search delay is accordingly:

$$X(z) = p_0 + \sum_{i=1}^{k} p_i \cdot X_A(z) \cdot X_N^i(z) \tag{3.8}$$

and the Laplace-Transform:

$$\Phi(s) = p_0 + \sum_{i=1}^{k} p_i \cdot \Phi_A(s) \cdot \Phi_N^i(s) \tag{3.9}$$

The mean and the coefficient of variation of the search delay can also directly be calculated as such:

$$E[T] = \sum_{i=1}^{k} p_i \cdot E[T|k=i] \tag{3.10}$$

$$= \sum_{i=1}^{k} p_i \cdot (E[T_A] + i \cdot (E[T_N])) \tag{3.11}$$

$$E[T^2] = \sum_{i=1}^{k} p_i \cdot E[T^2|k=i] \tag{3.12}$$

$$= \sum_{i=1}^{k} p_i \cdot (VAR[T_A] + i \cdot VAR[T_N] \tag{3.13}$$

$$+ (E[T_A] + i \cdot E[T_N])^2) \tag{3.14}$$

and

$$c_T^2 = \frac{E[T^2] - E[T]^2}{E[T]^2} \tag{3.15}$$

### 3.3.3 Influence of Stochastic Network Conditions

In this section we present numerical results to illustrate the dependency of the search duration on the stochastic variation of the network transfer delay and to give insight into the functional scalability of Chord-based overlay networks. First, we will show the mean and the coefficient of variation (CoV) of the search duration. Subsequently, the shape of the search delay distribution function will be discussed, followed by the quantile analysis, i.e. the guaranty that $\alpha$ percent of searches will need less than $t$ seconds.

Regarding the results in this section, the delay $T_N$ is assumed to be identical to the delay $T_A$. To unify the following parametric study the delay $T_N$ is further modeled by means of a two-parameter negative-binomially distributed random variable. If not stated otherwise, the coefficient of variation $c_{T_N}$ of $T_N$ is set to

1 and the mean $E[T_N]$ of $T_N$ is set to 50ms, since 50ms is the value assumed in the original Chord Paper [45]. Furthermore, we divide the obtained results by $E[T_N]$ where appropriate to obtain more general conclusions.

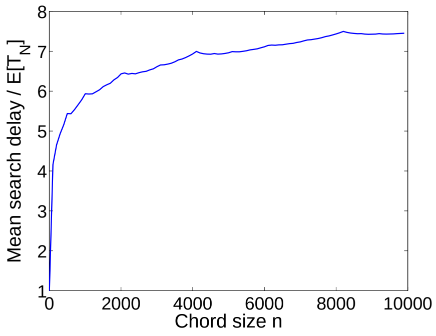


Figure 3.5: *Impact of the Chord size on the mean search delay*

Figure 3.5 shows the mean search delay as a function of the size of the Chord ring. We can observe that the search delay rapidly increases at smaller values of $n$, but stays moderate for very large peer populations. The curve is not strictly monotonically increasing as expected since a small decrease can be seen when the population $n$ just exceeds a binary exponential value $2^i$. This effect can be explained as follows: Once the size of the population crosses the next power of 2, the finger table of each peer grows by one entry according to our assumptions. Thus, the mean search duration slightly decreases at this point.

The coefficient of variation $c_T$ of the search delay $T$ is depicted in Figure 3.6 as a function of the peer population, for different transmission delay coefficients of variation. The variation of the search duration increases with $c_{T_N}$, the coefficient of variation of a single overlay hop. However, it decreases as the Chord size increases, due to the increasing number of hops needed in larger Chord populations. That is, the variance of the overall search duration is reduced due to the increasing number of convolutions of overlay hops.

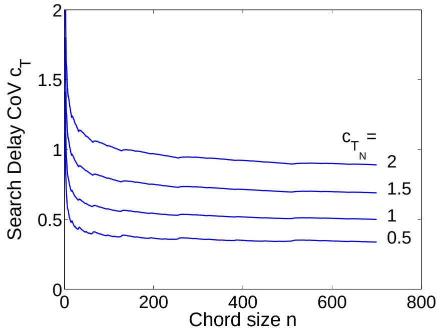


Figure 3.6: *Search delay variation as a function of the peer population*

This effect is also illustrated in Figure 3.7, where the dependency of $c_T$ on $c_{T_N}$ is analyzed. Again it can be seen that $c_T$ is smaller than $c_{T_N}$, as the convolution of multiple overlay hops reduces the coefficient of variation of the overal search duration. The size of the Chord population itself has a comparatively small effect on $c_{T_N}$.

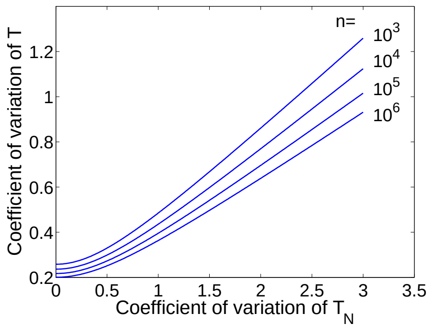


Figure 3.7: *Dependency of $c_T$ on $c_{T_N}$*

In Figures 3.8 and 3.9 we study the dependence of the entire distribution function of the search delay on the network latency variation $c_{T_N}$ and the peer population $n$, respectively. The size of the peer population in Figure 3.8 is set to $10^5$ peers. As expected, the probability that a search takes longer increases together with the coefficient of variation of the network latency $c_{T_N}$. The curves in Figure 3.8 intersect as they share the same mean $E[T_N]$ but have different coefficients of variation $c_{T_N}$.

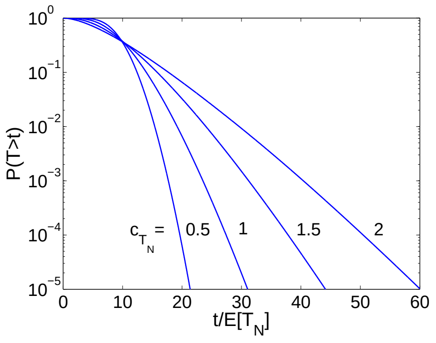


Figure 3.8: *Distribution function of the search delay*

Figure 3.9 proves the scalability of the search delay. By increasing the size of the Chord ring from $10^3$ peers to $10^6$ peers the search delay distribution does not escalate exponentially but increases by a linear factor. The chosen values of $n$ correspond approximately to current file sharing networks like the eDonkey network [62].

Figures 3.10 and 3.11 depict the quantile of the search delay $T$. In Figure 3.10 different quantiles for the search delay are taken as a parameter. For example the curve with the 99%-quantile indicates that 99 percent of search durations lie below that curve. For a peer population of, e.g., n=3000 in 99 percent of all cases the search delay is less then roughly 15 times the average network latency. It can be seen that the curves indicate bounds of the search delay, which can be used for

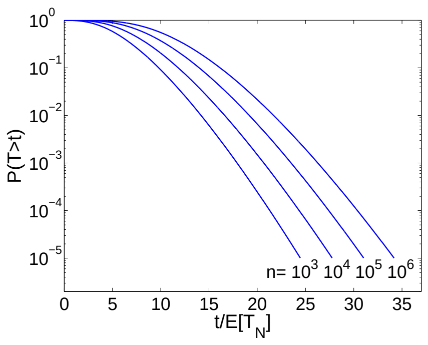


Figure 3.9: *Distribution function of the search delay*

dimensioning purposes, e.g., to know the quality of service in a search process with real-time constraints like looking at a phone directory, taking into account the patience of the users. Compared to the mean of the search delay the quantiles of the search delay are on a significantly higher level. Still the search delay scales in an analogous manner for the search delay quantiles.

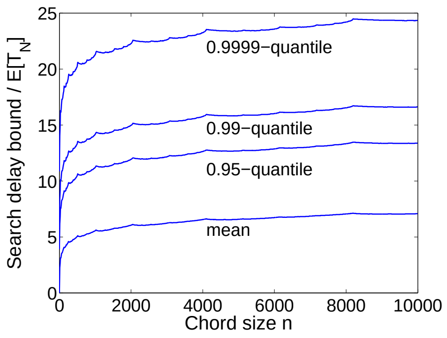


Figure 3.10: *Search delay quantiles*

Figure 3.11 depicts the 99%-quantile of the search delay, again with the coefficient of variation of $T_N$ as a parameter. There are five vertical lines at $n$=512, 256, 128, 64, and 32 to point out the previously mentioned oscillations at $n = 2^i$. The larger $c_{T_N}$ we chose, i.e. the more variation there is in the network delay, the

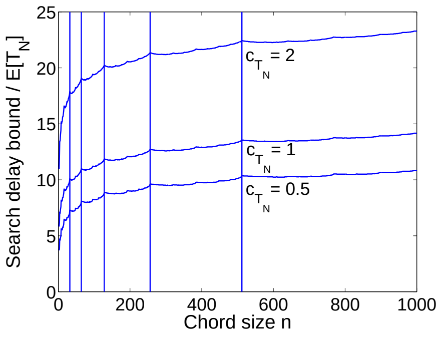


Figure 3.11: *Influence of the CoV of $T_N$ on the search delay quantiles*

larger is the 99%-quantile of the search duration. It is therefore more difficult to guaranty Service Level Agreements in networks with larger delay variation. Time outs, e.g. have to be set to higher values accordingly.

## 3.4 Evaluation of the Stability of Ring-based Architectures

The analysis in the last section represents a first step to validate the functional scalability in terms of the system size as well as the stochastic scalability in terms of network delay variation. However, the stability of structured overlay networks is also strongly affected by the dynamic behavior of the end user [20]. When many peers leave the network simultaneously, the overlay may be split into several disjoint networks or even collapse entirely. In case of such an inconsistent overlay state, successful searches can no longer be guaranteed and it might even not be possible to reestablish a stable overlay network again. A general analysis of the evolution of such systems can be found in [76] and [75]. In this section, we concentrate on ring-based overlay networks and analyze the way they preserve reachability and stability of the overlay network. The stability of such systems depends on the number of overlay connections a peer maintains. In contrast to previous studies [45] we show that the probability to lose the overlay structure of a ring-based DHT is not negligible in all cases. In particular, we present an analytical expression that can be used to calculate the probability to lose the routing functionality of a DHT given a certain number of overlay connections. We are able to evaluate the consequences of maintaining too many or too few overlay connections in a running system. The analysis can also be used to compute the actually necessary number of overlay connections to guarantee a stable overlay network.

### 3.4.1 Abstract Mathematical Model

In general, a p2p overlay network is connected if there exists a route from every peer to every other peer. In ring-based overlay structures this is achieved by each peer storing pointers to the first $r$ successors on the ring, i.e. pointers to the first $r$ peers that follow the peer in a clockwise direction on the ring. Thus, if one of the peer's $r$ successors goes offline, the peer will still know the next $r-1$ peers on the ring. If a peer, however, loses all its $r$ successors, the ring will be disconnected. According to previous studies [45] the connectivity of a Chord ring can be obtained with high probability as long as $r = \Omega(\log_2(n))$, where $n$ is the current number of peers in the ring. In this context, the network is assumed to stay connected, even if every peer fails with probability $\frac{1}{2}$. The proof relies on the fact that even though every individual peer fails with probability $\frac{1}{2}$, it is very unlikely that all $O(\log_2(n))$ successors of a peer fail at the same time. The conclusion that thus all peers stay connected with high probability, however, misses a subtle point. Although a local disconnection (one specific peer loses all its successors) might be very unlikely, one can not draw the conclusion that a global disconnection (at least one peer in the overlay loses all its successors) is very unlikely as well. To gain a better understanding of this subtle but important point, we introduce some definitions:

- $p_{fail}$: probability that a node fails
- $p_{ld}(r)$: probability that a specific node loses all its $r$ successors and gets locally disconnected
- $p_{gd}(n, r, p_{fail})$: probability of a global disconnection, i.e. the probability that at least one peer gets locally disconnected in a network of size $n$, where each node knows $r$ successors and each node fails with probability $p_{fail}$

The probability for a local disconnection can then easily be calculated as

$$p_{ld}(r) = p_{fail}^{r}. \tag{3.16}$$

Obviously, the more successors a peer has, the less likely it gets locally disconnected. Since, in general, peers maintain a successor list of size $r = O(\log_2(n))$, a local disconnection is less likely in larger networks. However, based on this observation alone, we can not conclude that the probability of a global disconnection is comparably small as well. The more nodes there are in the overlay network, the higher the probability that at least one of them gets locally disconnected. In other words, there is a trade-off between these two mechanisms. On the one hand, the larger the overlay ring becomes, the more successors are maintained by a peer, resulting in a smaller probability for a local disconnection. On the other hand, the larger the overlay ring becomes, the more peers run the risk of getting locally disconnected, resulting in a higher probability for a global disconnection.

To estimate the stability of a ring-based overlay structure, we need to calculate the probability $p_{gd}(n, r, p_{fail})$ of a global disconnection, i.e. the probability that at least once $r$ or more contiguous peers fail on the ring. As an approximation we neglect the ring structure of the overlay network and imagine the overlay peers arranged in an ascending row as shown in Figure 3.12. We regard the probability

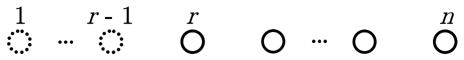


Figure 3.12: *The $n$ peers of a Chord ring arranged in an ascending row.*

$p_{rd}(x, r, p_{fail})$ that at least once $r$ or more contiguous peers fail in such a row of $x$ peers. Moreover, we can assume a random distribution of failures, since the hash function distributes peers equally in the identifier space and physical proximity therefore does not reflect overlay proximity. For the sake of simplicity, we use the short notation $p_{rd}(x)$ instead of $p_{rd}(x, r, p_{fail})$ where appropriate. Obviously, the probability that $r$ or more peers fail in a row of less than $r$ peers is zero, as indicated by the dotted peers in Figure 3.12. If we consider the same probability in a row of exactly $r$ peers, all peers have to fail accordingly. The corresponding equations are:

$$p_{rd}(x, r, p_{fail}) = 0 \qquad \text{if } x < r \tag{3.17}$$

$$p_{rd}(x, r, p_{fail}) = p_{ld}(r) = p_{fail}^{r} \qquad \text{if } x = r. \tag{3.18}$$

In case of $x > r$ we obtain:

$$p_{rd}(x) = p_{rd}(x-1) + (1 - p_{rd}(x-r-1)) \cdot (1 - p_{fail}) \cdot p_{ld}(r). \tag{3.19}$$

The probability $p_{rd}$ is defined recursively. To calculate $p_{rd}(x)$, we take the probability $p_{rd}(x-1)$ that there was at least one local disconnection in the first $x-1$ peers and add the probability that the first local disconnection occurs at peer $x$. The second term of this sum is best explained using Figure 3.13. There are two

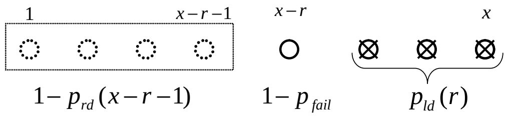


Figure 3.13: *Probability, that the first local disconnection occurs at peer $x$.*

requirements in order that the first local disconnection occurs exactly at peer $x$. First of all, there must not be a local disconnection in the first $x - r - 1$ peers as indicated by the box in Figure 3.13. Secondly, peer $x - r$ must not fail, while all of the last $r$ peers have to fail to cause the disconnection at peer $x$.

According to Equations 3.17 and 3.18, the first local disconnection can occur at peer $r$. Thus, there are still $r - 1$ peers that could experience a local disconnection but are not accounted for in our equation. To improve the accuracy of our approximation, we add $r - 1$ peers at the end of the row as shown in Fig 3.14. Thus, there are $n$ peers in a row of $n + r - 1$ peers that can experience a lo-

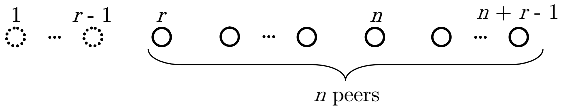


Figure 3.14: *The first $r - 1$ peers are added to the end of the row.*

cal disconnection. The resulting approximation for the probability of a global disconnection in a Chord ring of size $n$ is:

$$p_{gd}(n, r, p_{fail}) \approx p_{rd}(n + r - 1, r, p_{fail}). \tag{3.20}$$

The reason for the approximation is that we neglect the ring structure of the overlay network. In fact the probability is slightly overestimated since the $r - 1$ peers we added at the end of the row are obviously correlated with the first $r - 1$ peers in the row. That is, there are some failure patterns in the last $r - 1$ recursion steps that have already been taken into account before and are thus counted twice. Note that the formula is not limited to the special case of

$$r = \lceil \log_2(n) \rceil.$$

In fact, we are able to evaluate the consequences of using too large or too small values for $r$, i.e. of using more or less than $\log_2(n)$ successors.

### 3.4.2 Derivation of Realistic Failure Probabilities

In the previous section we were simply assuming values for $p_{fail}$, the probability that a node fails. In practice, however, there is not much sense in saying a node fails with a certain probability, without specifying a corresponding time frame. To guarantee overlay stability, a peer refreshes its successor list every $t_{stab}$ seconds by periodically calling a *stabilize()* procedure. This *stabilize()* function takes care that a peer's successor list is up to date by merging its list with the list of its closest successor. Thus, a peer gets locally disconnected if all of its known successors go offline between two *stabilize()* calls. Therefore, one should consider the probability that a peer fails within this periodic update interval instead of assuming some arbitrary values for $p_{fail}$.

On account of this, we regard $E_{on}$, the average online time of a peer in seconds. Assuming that the online time is exponentially distributed with $\lambda_{on} = \frac{1}{E_{on}}$ it follows that

$$A(t) = 1 - e^{-\lambda_{on} t} \tag{3.21}$$

is the distribution function of the online time of a single peer. Due to the memoryless property of the exponential distribution the probability that a peer goes offline within $t_{stab}$ seconds is:

$$p_{fail} = A(t_{stab}) = P(A \leq t_{stab}). \tag{3.22}$$

Note that the probability that a peer goes offline and online again within $t_{stab}$ seconds is neglected in this context. We can then use this $p_{fail}$ in Equation 3.20 to calculate the probability of a global disconnection within $t_{stab}$ seconds. The probability of a global disconnection increases with the number of *stabilize()* calls. The longer the overlay ring exists, the greater the probability of a global disconnection within its lifetime becomes. The probability $p_{it}(n, i)$ that a ring of size $n$ gets globally disconnected sometime within $i$ *stabilize()* calls can be calculated as follows:

$$p_{it}(n, i) = 1 - (1 - p_{gd}(n, r, p_{fail}))^{i}. \tag{3.23}$$

Note, that in the context of peer lifetimes heavy-tailed distributions are more resilient than those with light-tailed (e.g., exponential) distributions [87].

## 3.4.3 Validation of the Stability of the Overlay Structure

In this section we concentrate on the evaluation of Chord, the most prominent structured p2p overlay network. The results regarding the problem of a disconnection, however, are valid for any ring-based overlay network. At first we have a closer look at the probability of a local disconnection. Figure 3.15 illustrates the probability of a local disconnection (cf. Equation 3.16) against the overlay size for three different failure probabilities of a peer. The number of successors is thereby set to $\lceil \log_2(n) \rceil$. As expected the probability of a local disconnection strongly decreases with the size of the overlay network. This is obviously due to the fact that a peer maintains more successors in larger networks and is thus less likely to be disconnected. Note that in a ring of size $n = 10^6$ and a failure probability of $p_{fail} = \frac{1}{2}$ we have a very low probability of a local disconnection of about $10^{-6}$.

To show that based on these facts alone, we can not derive a very low probability for a global disconnection as well, we calculate the probability of a global dis-

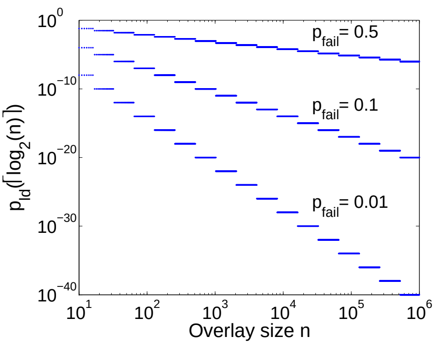


Figure 3.15: *Probability of a local disconnection for different values of* $p_{fail}$

connection for $p_{fail} = \frac{1}{2}$. Figure 3.16 shows this probability (cf. Equation 3.20) for networks of size $n = 2^k$, where each peer maintains a successor list of size $r = \log_2(n) = k$. The probability of a global disconnection does indeed de-

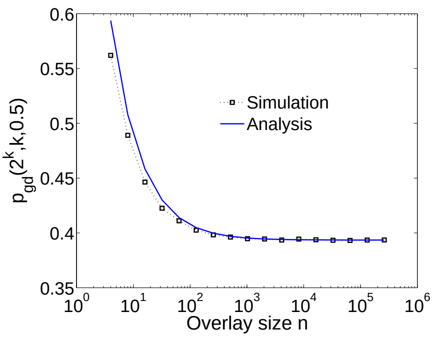


Figure 3.16: *Global disconnection probability in the special case of* $p_{fail} = \frac{1}{2}$.

crease with the size of the overlay network. However, it does not approach zero but asymptotically reaches a probability of about 40 percent. So when every node fails with probability $p_{fail} = \frac{1}{2}$ and every peer maintains a successor list of size $r = \log_2(n)$ Chord does not stay connected with very high probability but gets disconnected with a probability of roughly 40 percent.

To confirm this result we simulated the probability of a global disconnection by generating snapshots of rings of a specific size and counted the percentage of those rings that did not get disconnected after 50 percent of all peers failed. The simulations were repeated until the confidence intervals became smaller than 0.001. For smaller values of $n$ the results obtained by our analysis are slightly above the simulated values as the analysis does not take the ring structure into account. The error becomes negligible for overlay sizes above $n = 100$.

In practice, however, a failure probability of $p_{fail} = \frac{1}{2}$ is obviously too pessimistic. To obtain realistic values for $p_{fail}$ we evaluate Equation 3.22 for different average online times of a peer and different values of $t_{stab}$. Figure 3.17 shows that even if the average peer only stays online for 10 minutes and successor lists are only refreshed every 60 seconds, the probability that a peer fails within this frame of time is still less than 10 percent.

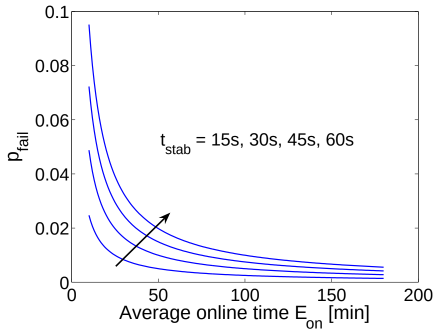


Figure 3.17: *Failure probabilities in dependency of the average online time*

In the following analysis we therefore concentrate on $p_{fail}$= 0.1, 0.05, and

0.01. Figure 3.18 illustrates that a global disconnection is very unlikely for these values of $p_{fail}$. Even for a peer failure probability of 10 percent, a Chord ring of size $10^5$ will be globally disconnected with a probability of less than $10^{-12}$. The staircase shape of the curve arises from the fact that the plot is done for arbitrary $n$ and corresponding successor lists of size $r = \lceil \log_2(n) \rceil$. So whenever the overlay size $n$ crosses a power of two, each peer starts to maintain one additional successor in its successor list. Therefore, the probability of a disconnection abruptly decreases whenever a power of two is exceeded. It then slightly increases until the next power of two, since the probability of a local disconnection stays the same, but there are more peers that can get disconnected and cause a global disconnection.

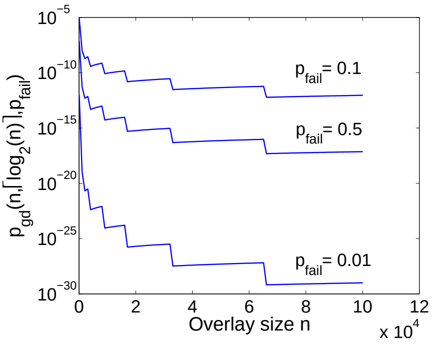


Figure 3.18: *Probability of a global disconnection with $\lceil log_2(n) \rceil$ successors.*

So far, the results relied on a dynamic adaptation of the size of a peers successor list. In practice, however, it is more common to choose a fixed successor list size a priori. Figure 3.19 illustrates the probability of a global disconnection for fixed successor list sizes of 3, 6, and 9. The failure probability of a peer is set to $p_{fail} = 0.01$. As we can see, the probability of a disconnection increases with the overlay size but scales very well to larger networks. Moreover, the or-

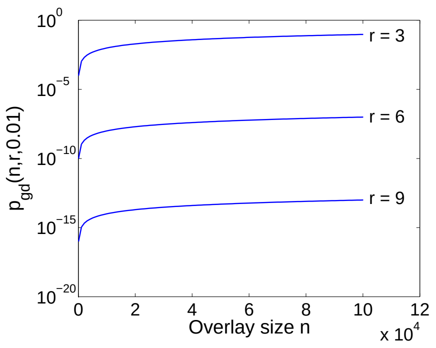


Figure 3.19: *Impact of a fixed number of successors on the global disconnection.*

der of magnitude of the probability of a global disconnection can be adjusted by choosing the corresponding successor list size. Obviously, less than $\lceil \log_2(n) \rceil$ neighbors are sufficient to guarantee a stable ring when we assume a realistic failure probability of $p_{fail} = 0.01$.

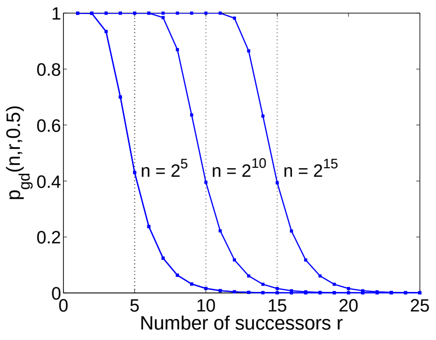


Figure 3.20: *Global disconnection probability for different successor list sizes.*

To illustrate the effects of extremely high failure probabilities we plot the probability of a global disconnection against the number of successors $r$. In Figure 3.20 we show the results for a peer failure probability $p_{fail} = \frac{1}{2}$ and three different ring sizes $n = 2^5$, $2^{10}$, and $2^{15}$. The vertical black dotted lines represent the suggested successor list size $\lceil \log_2(n) \rceil$. Again, the suggested number of successors results in a disconnection probability of about 40 percent. To guarantee a global disconnection probability close to zero in this example, a peer has to maintain a successor list of size $\lceil \log_2(n) \rceil + 7$ or more.

Note that so far we calculated disconnection probabilities within one single *stabilize()* period. However, the probability of a global disconnection increases over time. The longer the Chord ring exists, the greater the probability that it gets disconnected within its lifetime. Figure 3.21 plots the probability that a ring gets disconnected sometime within $i$ *stabilize()* calls against the number of *stabilize()* calls for different global disconnection probabilities (cf. Equation 3.23). Assuming a *stabilize()* period of length $t_{stab} = 30$ seconds, $8 \cdot 10^4$ *stabilize()*

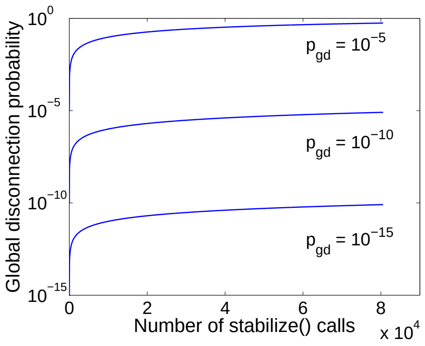


Figure 3.21: *Probability of a global disconnection after $i$ stabilize() calls*

calls roughly correspond to one month. Thus, the probability that a Chord ring gets disconnected sometime within the first month of its lifetime is by magnitudes greater than the same probability within one single *stabilize()* period.

## 3.5 Simulative Evaluation of a Carrier-Grade Kademlia Network

Most scientific studies as well as our analysis in the last section concentrate on Chord or other ring-based overlay structures. This is probably due to the fact that even though all DHTs share the same basic principle, the ring-structure is by far the most easy to analyze. The majority of actually deployed overlay networks, however, make use of the more complex Kademlia protocol [51]. It replaces the server in the latest eMule modifications and is used as a distributed tracker in the original BitTorrent as well as in the Azureus client [52]. The latter continuously attracts more than one million simultaneous users world wide. Despite all this there are only few scientific papers evaluating the performance of the Kademlia algorithm. In [89] and [90] the performance of different DHT algorithms including Kademlia is evaluated and compared. Modifications to support heterogeneous peers are introduced in [102]. Finally in [103] an analysis of the lookup performance of Kad, the Kademlia-based DHT used in eMule, is given. The authors examine the impact of routing table accuracy on efficiency and consistency of the lookup operation and propose adequate improvements.

Like all structured p2p networks, Kademlia has explicitly been designed to scale to a large number of peers in the overlay. Therefore the real scalability issue is not in terms of system size but in terms of churn [75]. That is, the frequency at which peers join and leave the system has significantly more influence on its robustness and stability than the mere size of the system itself. In this section we therefore uncover the problems caused by churn and show how to avoid them [13]. In particular, we study the search duration, the overlay stability, and the required maintenance traffic. We will then describe the weak points we discovered and pinpoint their root causes. For each problem we will present an optimization, which eliminates the disadvantages and makes Kademlia a protocol more feasible for business applications. Even though the algorithms will be explained in the context of Kademlia, they are by no means restricted to this protocol. Especially the downlist and the Betarepublish mechanisms can easily be applied to other DHTs like Pastry, CAN, or Chord.

### 3.5.1 Description of the Simulation Environment

Discrete event simulation is a powerful tool to gain insight into complex processes at the desired level of abstraction. There exist several p2p simulators in literature, a good overview and comparison is given in [79]. However, each of these simulators comes with its disadvantages such as little flexibility, poor or no documentation, implementation errors, no extensibility, or missing features, but most importantly the lack of scalability. In order to make the simulation of large scale p2p systems more feasible, we therefore developed our own simulation environment (cf. Figure 3.22) as well as different implementation techniques [11, 12]. The global user behavior is described in a special source file using our own script

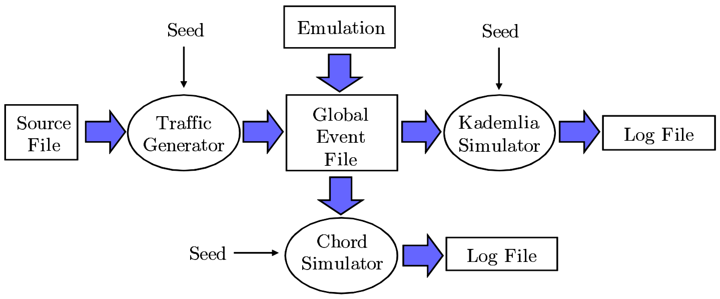


Figure 3.22: *Sketch of the simulation environment*

language. That is, the source file contains an abstract description of all global events like joins, leaves, or searches which are independent of the underlying p2p network. The simple line *join 500 10*, e.g., denotes that 500 peers should join the overlay network at intervals of 10 seconds. The traffic generator translates this description into actual events which can then be used as input for the simulation of different protocols. This makes it easy to compare the performance of different DHT protocols in a given scenario. It is furthermore also possible to extract the global events from an emulation or prototype study and re-simulate the scenario to validate the accuracy of the simulation.

In order to evaluate the different performance aspects of Kademlia, we implemented a discrete event simulator in ANSI-C according to the algorithms in [51]. That is, for each $0 \leq \; i < 160$ a peer keeps a bucket of $k$ peers of distance between $2^{160-i}$ and $2^{160-i+1}$ from itself according to the XOR metric. Thereby the routing table is adapted dynamically. That is, each peer starts with one single bucket covering the entire address space and recursively splits the bucket containing the peer's own ID as soon as this bucket holds more than $k$ entries. When many peers leave the system, Kademlia merges the corresponding buckets accordingly. Furthermore, a peer is able to insert documents into the overlay network. To guarantee their availability, each of these documents is stored at the $k$ closest peers to the document's ID. If the document was not received from another peer for $T_{rep}$ minutes, the corresponding peer republishes the document, i.e. it sends the document to the remaining $k-1$ peers of the replication group. When searching for a document a peer recursively sends parallel queries to the $\alpha$ closest peers it knows. The next recursion begins as soon as the peer received $\beta$ answers. This guarantees that a searching peer will only run into a timeout if $\alpha - \beta + 1$ peers do not answer within one specific search step. If not stated otherwise, we use the default parameters $T_{rep} = 60$ minutes, $\alpha = 3, \beta = 2$, and $k = 20$.

To model end user behavior, we randomly chose join and leave events for each peer. To be comparable to other studies in literature a peer stays online and offline for an exponentially distributed time interval with a mean of $E_{on}$ and $E_{off}$ respectively. When online, the peer issues a search every $E_{search}$ minutes, where the time between two searches is also exponentially distributed. Using different distributions mainly changes the quantitative but not the qualitative statements. In each simulation we use a total of 40000 peers, which we found to be sufficiently large to capture all important effects regarding the overlay size, and set

$E_{on} = E_{off}$, resulting in an average overlay size of 20000 peers. To increase the credibility of our results [104], we include the 95 percent confidence intervals where appropriate.

### 3.5.2 Improving the Search Efficiency

The success and duration of a search for a document heavily depend on the correctness of a peer's pointers to other peers, i.e. on the correctness of the peer's routing table. In Kademlia the most crucial pointers are those to its $k$ closest neighbors in the overlay. We measure the correctness of these pointers using two different variables:

- $P_h$: States how many of its current $k$ closest neighbors a peer actually holds in its k-buckets.
- $P_r$: Represents the number of correct peers out of the $k$ closest neighbors, which a peer actually returns when asked for.

Ideally a peer would not only know but also return all of its $k$ neighbors. However, our simulations show that the standard implementation of Kademlia has problems with $P_r$. We set $k = 20$ and simulated the above described network for different churn rates. Figure 3.23 illustrates $P_h$ and $P_r$ in dependence of the churn rate. The mean online/offline time of a peer was variied between 10 and 180 minutes. Even though on average a peer knows almost all its neighbors ($P_h$ close to 20), it returns significantly less valid entries when queried ($P_r$ as low as 13). The shorter a peer stays online on average, the less valid peers are returned during a search. The problem can be tracked down to the fact that there are still many pointers to offline peers in the corresponding k-bucket of the peer. The reason is that there is no effective mechanism to get rid of out-dated k-bucket entries. Offline entries are only eliminated (or moved to the cache) if a peer runs into a timeout while trying to contact an offline peer. A peer which identifies an offline node, however, keeps that information to itself. Thus, it is not unlikely that a

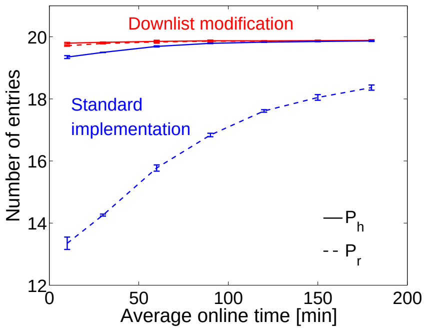


Figure 3.23: $P_h$ *and* $P_r$ *in dependence of the churn rate*

node returns offline contacts as it has very limited possibilities to detect offline nodes. As a result more timeouts occur and searches take longer than necessary. Another problem is that searches are also getting more inaccurate, which has negative effects not only on the success of a search but also on the redundancy of the stored documents. The reason is that due to the incorrect search results documents will be republished to less than $k$ peers or to the wrong peers.

**Solution - Downlists** The primary reason for the above mentioned problem is that so far only searching peers are able to detect offline nodes. The main idea of our solution to this problem is that a searching peer, which discovers offline entries while performing a search, should share this information with appropriate other peers. To do so, a peer maintains a downlist consisting of all peers which it discovered to be offline during its last search. At the end of the search the corresponding entries of this downlist are sent to all peers which gave those entries to the searching peer during its search. These peers then also remove the received offline entries from their own k-buckets. This mechanism helps to get rid of offline entries by propagating locally gained information to where it is needed. With each search offline nodes will be eliminated.

The improved stability of the overlay is obviously bought by the additional bandwidth needed to send the downlists. From a logical point of view, however, it does require more overhead to keep the overlay stable under higher churn rates. In this sense, the additional overhead traffic caused by sending downlists is self-organizing as it automatically adapts to the current churn rate. The more churn there is in the system, the more downlists are sent.

It should also be mentioned, that without appropriate security arrangements a sophisticated attacker could misuse the downlist algorithm to exclude a target node by claiming in its downlist that this specific node had gone offline. However, this problem can be minimized by only removing those nodes which were actually given to the searching node during a search or additionally by verifying the offline status using a ping message. One could also apply trust or reputation based mechanism to exclude malicious nodes.

**Effect on Search Efficiency** To compare the downlist modification to the standard implementation we again simulated a scenario with 20000 peers on average and calculated the 95 percent confidence intervals. Figure 3.23 proves, that the downlist modification has the desired effect on $P_r$, the number of correctly returned neighbors. Using downlists both $P_h$ and $P_r$ stay close to the desired value of 20, almost independent of the current churn rate. That is, even in times of high churn the stability of the overlay can be guaranteed.

This improved correctness of the overlay stability also has a positive influence on the search efficiency. In Figure 3.24 we plot the average duration of a search against the average online/offline time of a peer. In this context an overlay hop was modeled using an exponentially distributed random variable with a mean of 80 ms. Both curves show the same general behavior. The longer a peer stays online on average, the shorter is the duration of a search. However, especially in times of high churn, the downlist modification (lower curve) significantly outperforms the standard implementation. The main reason is that on average a peer runs into more timeouts using the standard implementation, as it queries more offline peers during a search. The effects on the maintenance overhead will be discussed in Section 3.5.4.

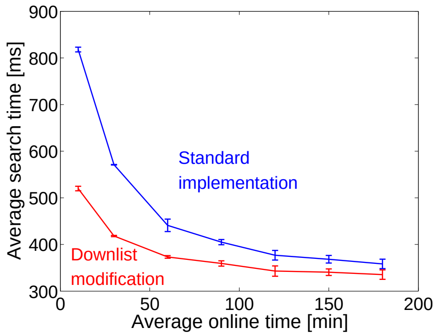


Figure 3.24: *Influence of the downlist modification on the search efficiency*

### 3.5.3 Increasing the Robustness of the Overlay

When peers join and leave the overlay network, the neighbor pointers of a peer have to be updated accordingly. As mentioned above, the downlist modification greatly improves the correctness of the $k$ closest neighbors of a peer. To understand this effect in more detail, we have a closer look at a single simulation run. We consider a mean online/offline time of 60 minutes and an average of 20000 peers for both the standard implementation and the downlist modification.

Figure 3.25 illustrates the distribution of $P_h$ and $P_r$ in both scenarios. As can be seen in the left part of the figure, almost all peers know more than 17 of their 20 closest neighbors using the standard implementation. However, the number of correctly returned peers $P_r$ is significantly smaller for most peers. This problem is greatly reduced by the downlist modification as can be seen in the right part of the figure. In this case, the number of known and the number of returned peers are almost equal to each other. Yet, there are still some peers, which do not know all of their 20 closest neighbors. This is in part due to the churn in the overlay network. However, simulations without churn produce results, which are comparable to those shown in the right part of Figure 3.25. The cause of this

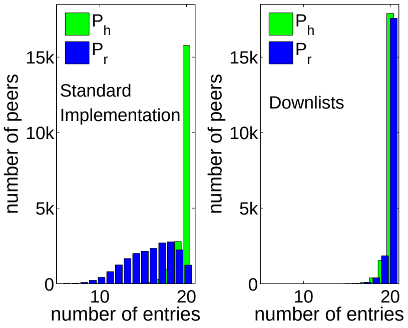


Figure 3.25: *$P_h$ and $P_r$ for the original version and the downlist modification*

problem can be summarized as follows: Let $B_p$ be the k-bucket of peer p, which includes the ID of peer p itself and $B_{\bar{p}}$ the brother of $B_p$ in the binary tree whose leaves represent the k-buckets as shown in Figure 3.26. Then according to the Kademlia algorithm bucket $B_p$ is the only bucket which will be split. However, if only $e < k$ of the actual $k$ closest contacts fall into this bucket, then $v = k - e$ of these contacts theoretically belong into its brother $B_{\bar{p}}$.

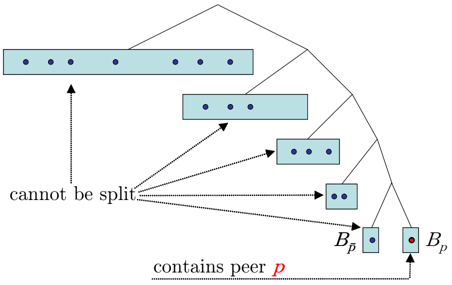


Figure 3.26: *$B_p$ and its brother $B_{\bar{p}}$ in the Kademlia routing table*

Now, if this bucket is full it cannot be split. Thus, if some of the $v$ contacts are not already in the bucket, it is very unlikely that the peer will insert them into its buckets. The reason is, that a new contact will be dropped in case the least recently seen entry of $B_{\bar{p}}$ responds to a ping message. Since in a scenario without churn all peers always answer to ping messages, new contacts will never be inserted into $B_{\bar{p}}$, even though they might be among the $k$ closest neighbors of the peer. In the original paper it is suggested to split additional buckets in which the peer's own ID does not reside in order to avoid this problem. However, this has two major drawbacks. At first, it is a very complex process, which is vulnerable to implementation errors. Secondly, it involves a great deal of additional overhead caused by bucket refreshes and other maintenance routines. In the next section, we therefore develop a simple solution, which does not require any additional overhead.

**Solution - Force-$k$** As stated above, it is possible, that a peer does not know all of its $k$ closest neighbors, even in times of no churn. To solve this problem, we need to find a way to force a peer to always accept peers belonging into $B_{\bar{p}}$ in case they are amongst its $k$ closest neighbors. Suppose a node receives a new contact, which is among its $k$ closest neighbors and which fits into the already full bucket $B_{\bar{p}}$. So far, the new contact would have been dropped in case the least recently seen entry of $B_{\bar{p}}$ responded to a ping message. Compared to this, the Force-$k$ modification ensures that such a contact will automatically be inserted into the bucket. In order to decide which of the old contacts will be replaced, one could keep sending ping messages and remove the first peer, which does not respond. However, this again involves additional overhead in terms of bandwidth. A faster and passive way is to put all entries of $B_{\bar{p}}$, which are not among the $k$ closest peers into a list $l$ and drop the peer which is the least useful. This could be the peer which is most likely to be offline or the peer which has the greatest distance according to the XOR metric.

In our implementation, we decided to consider a mixture of both factors. Each of the entries $e$ of list $l$ is assigned a specific score

$$s_e = t_e + d_e \tag{3.24}$$

and the one with the highest score will be dropped. Thereby, $t_e$ is intended to be a measure for the likelihood of peer $e$ to be offline and $d_e$ for the distance of peer $e$ to peer $p$. The exact values of $t_e$ and $d_e$ are obtained by taking the index of the position of the corresponding peer in the list, as if it was sorted ascending by the time most recently seen or by the peer's distance, respectively. That is, if $e$ is the most recently seen peer ($t_e = 1$) and has the third closest distance to peer $p$ ($d_e = 3$) it is assigned a score of $s_e = 4$.

**Effect on Stability** We investigated the impact of the Force-$k$ modification on the stability of the overlay network in various simulations. In scenarios without churn, all peers finally know and return all of their $k$ closest neighbors. The corresponding figures show lines parallel to the x-axis at a value of $k = 20$. It is therefore more interesting to regard the overlay stability during churn phases.

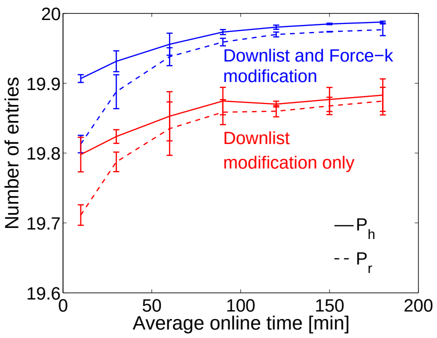


Figure 3.27: *Effect of Force-$k$ under churn*

In Figure 3.27, we plot the average online time of a peer against the number of known and returned neighbors using the same simulation scenario as before. The two lower curves correspond to our previous results using the downlist modification. The two upper curves represent the Force-$k$ modification in combination with the downlist modification. It can be seen that the Force-$k$ algorithm also improves the stability of the overlay in times of churn. While the appearance of the curves is similar, there are more neighbors known (solid lines) and returned (dashed lines) as compared to using only the downlist modification. Even if a peer stays online for only 10 minutes on average, it will know about 19.9 out of 20 neighbors and return more than 19.8 correct entries. By improving the correctness of the neighbors, the Force-$k$ modification also increases the search success rate and the redundancy of stored documents.

To investigate the overlay stability in more critical scenarios, we simulated a mass exit where 90% of all peers left the overlay at a random time within an interval of $x$ minutes. Figure 3.28 shows that even if 90% of all peers leave within 10 minutes, our modified Kademlia algorithm still knows and returns about half of its $k$ closest neighbors. Moreover, shortly after the mass exit the overlay recovers again, whereas all peers correctly know and return all of their $k$ closest neighbors.

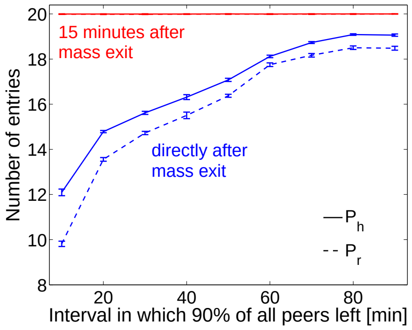


Figure 3.28: *Overlay stability after a mass exit*

So far, we showed how to improve the efficiency and the stability of the overlay, but did not yet consider the maintenance traffic caused by the Kademlia algorithm. The next section will study the bandwidth required by a peer running the standard Kademlia protocol and the additional overhead traffic caused by our modifications.

### 3.5.4 Reducing the Redundancy Overhead

The bandwidth required to maintain a stable overlay and to ensure the persistence of stored documents directly reflects the costs for a peer to participate in the network. We simulated a network with 20000 peers on average and recorded the average number of packets per second sent by a peer while it was online. Figure 3.29 illustrates the average traffic per peer in dependence of the average online time of a peer. In addition to the total traffic, the figure also shows its three main components, the join, the republish, and the downlist traffic.

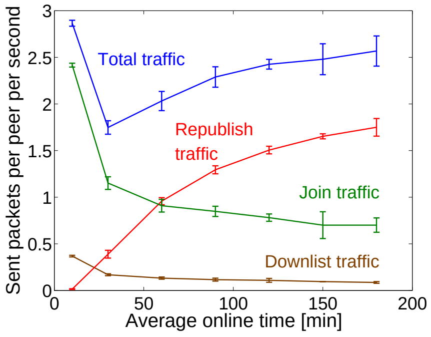


Figure 3.29: *Maintenance traffic of a peer split into its components*

Since $E_{search}$, the average time between two searches of a peer, was set to 15 minutes, the search traffic per peer per second can be neglected in this scenario and is thus not shown in the figure. The same is true for the traffic caused by bucket refreshes, since a specific bucket is only refreshed if it has not been used for an entire hour. The Force-$k$ algorithm is performed locally and does thus also not produce any additional overhead.

It can be seen in the figure that the downlist traffic automatically adapts itself to the current churn rate. The more frequently the peers join and leave the system, the more downlist traffic is produced by a peer on average. In general, the small amount of bandwidth needed to distribute the downlists is also easily compensated by the improved stability of the overlay. The major part of the traffic is caused when joining the network and republishing documents. It is obvious that the average amount of join traffic per peer per second increases if a peer stays online for a shorter period of time. The join traffic cannot and should not be avoided as it is necessary for a peer to make itself known when it joins the network. Moreover, the join traffic already shows a self-organizing behavior. The more churn there is in the system, the more joins there are in total and the more overhead is produced to compensate the problems caused by the churn.

At first, the run of the curve representing the republish traffic seems to be counter-intuitive. The less churn there is in the system, the more republish traffic is sent by a peer on average. However, the reason becomes obvious, if one takes into account that the longer a peer stays online on average, the more likely it gets that there are republish events. In fact, the probability that a peer stays online for longer than 60 minutes given the corresponding average online time $E_{on}$, resembles the run of the republish curve. The reason why the total amount of republish traffic exceeds the remaining traffic so significantly is as follows: Each document is stored at the $k$ closest nodes to its ID, the so called replication group. To compensate for nodes leaving the network, each peer sends the document to all other peers of the replication group if it has not received the document from any other peer for $T_{rep} = 60$ minutes. The idea behind this republish mechanism is that one peer republishes the document and all other peers reset their republish timers accordingly. Since the republishing peer sends the document to all peers of the replication group simultaneously, the peers reset their timers at approximately the same time. The next time the first peer starts to republish the document, it has to search for the corresponding replication group before it can redistribute the document. However, during this search the republish timers of the other peers are likely to run out and they will start to republish the document as well. For this reason, a document might get republished by up to $k$ peers instead of just one single peer, resulting in unnecessary overhead traffic. This problem of synchronization is already mentioned in the original paper. In the following section, we present a solution, which greatly reduces the republish overhead and which is also resistant against churn.

**Solution - Betarepublish** The synchronization problem of the republish process arises if all peers of a replication group have approximately the same time stamp for the next republish event. At first this seems to be unlikely. However, each time a peer republishes a document all other peers of the replication group receive this document at approximately the same time and are thus synchronized again. The main idea to avoid this problem is to assure that all peers use different time stamps. To achieve this, each peer chooses its time stamp randomly in the interval $[T_{rep} - x, T_{rep} + x]$ instead of exactly after $T_{rep} = 60$ minutes. Let $I_{rep}$ be the random variable describing the time stamp of the next republish event. Then we want $I_{rep}$ to be distributed in such a way, that only few peers start republishing at the beginning of the interval and the probability to republish increases toward the end of the interval. This can, e.g., be achieved by setting:

$$I_{rep} = (T_{rep} - x) + 2 \cdot x \cdot I_{beta} \tag{3.25}$$

where $I_{beta}$ is a random variable with density

$$i_{beta}(t) = \begin{cases} \frac{t}{\sqrt{(1-t) \cdot B(2,0.5)}} & \text{if } 0 < t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

and $B(\alpha, \beta)$ is the beta function, defined by

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} \, \mathrm{d}t \tag{3.27}$$

Thereby $2 \cdot x$, the length of the interval in which the peers will start their republish process, should be small compared to $T_{rep}$ but still significantly larger than the duration of a search. Figure 3.30 shows the probability density function of $I_{rep}$ for different values of $x$. All peers will set their time stamps somewhere

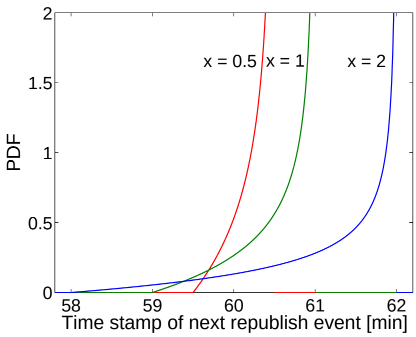


Figure 3.30: *PDF of $I_{rep}$ for different values of $x$*

in the interval $[60 - x, 60 + x]$. The probability for a peer to set its time stamp is still very low at the beginning of the interval. It then ascends significantly toward the end of the interval. In the case of $T_{rep} = 60$ minutes, $x = 2$ minutes is a reasonable choice, since it offers a long period of time with a low probability of republish events. This way, the republish traffic will be significantly reduced as it becomes very likely that only one or a few peers actually start a republish process. Again, note that a peer does only republish a document if it has not received it from another peer for $T_{rep} = 60$ minutes.

**Effect on Overhead** In this section we will have a look at the influence of the Betarepublish modification on the average amount of republish traffic sent by a peer. Figure 3.31 shows the average number of republish packets per peer per second in dependence of the average online time. We compare the results for simulations using the standard implementation, our two previous modifications, and all modifications including Betarepublish. First of all, the average republish

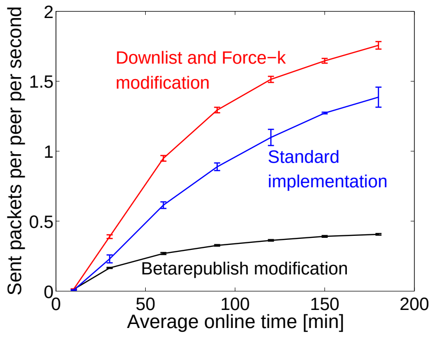


Figure 3.31: *Maintenance traffic caused by republish processes*

traffic of a peer is increased by using the downlist modification. The reason is that using the standard implementation there are more offline nodes in the $k$-buckets during times of churn. Thus, documents are republished to less peers, which reduces the republish traffic but also the redundancy in the system. The additional traffic introduced by the downlist modification is therefore used to improve the availability of documents.

The Betarepublish modification is applied in an effort to minimize the traffic necessary to achieve this availability. The figure shows that Betarepublish indeed reduces the amount of required republish traffic significantly. The Betarepublish traffic lies well beneath the standard implementation and also rises slower with an increasing average online time. Note that the Betarepublish modification does only avoid redundant traffic. It is still able to guarantee the same redundancy, stability, and functionality. Figure 3.32 shows how the reduced republish traffic influences the total traffic for the three regarded versions of Kademlia (Standard, downlists and Force-$k$, all modifications). At first, it can be seen that the use of downlists increases the total traffic as compared to the standard implementation. Again, this is desired overhead as it greatly helps to increase the robustness, the stability, and the redundancy of the overlay in an autonomous way.

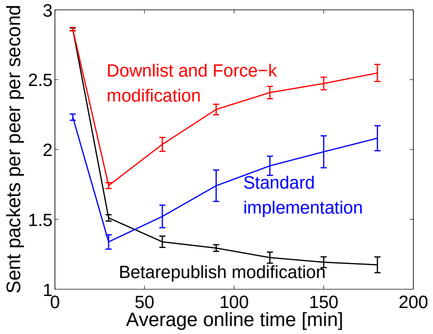


Figure 3.32: *Total maintenance traffic in dependence of the churn rate*

By adding the Betarepublish modification, the total traffic is significantly reduced and no longer dominated by the republish traffic. While the average maintenance traffic sent by a peer in the standard implementation actually increases when there is less movement in the overlay network, it finally shows a self-organizing behavior when using all modifications. The less churn there is in the system, the less maintenance traffic is generated to keep the overlay network up to date. That is, the amount of bandwidth invested to keep the overlay running automatically adapts itself to the current conditions in the overlay.

# 4 Modeling the Dynamics of P2P Overlays

In contrast to the classic client-server architecture, overlay networks have to cope with highly dynamic components in their system. In order to maintain the structure of the overlay under such conditions they have to apply appropriate countermeasures. In particular, they may adjust parameters like the number of overlay connections to other peers or the frequency at which they exchange information about the current overlay status with those peers. The optimal amount of such maintenance overhead directly depends on the current size of the overlay as well as the current online/offline behavior of the participating peers. While insufficient overhead may lead to loss of the overlay structure and ultimately to a break down of the entire system, too much overhead results in an unnecessary waste of available resources. In practice, the maintenance overhead in structured overlay networks is set to a fixed value which is dimensioned for the expected worst case.

In this chapter, we take the first step toward a self-organizing concept for overlay networks as illustrated in Figure 4.1. The main problem in this context is that to a single peer the remaining system essentially appears as a black box. We therefore introduce and discuss different models to estimate the current conditions in the overlay, like churn or the system size, based on information which is locally available to a peer. These estimates can then be used by a peer to calculate the probability of a loss of the overlay stability, as e.g. shown in Chapter 3.4. From this it can then derive optimal parameters to adapt the maintenance overhead accordingly. Furthermore, the fact that the current status as well as the performance

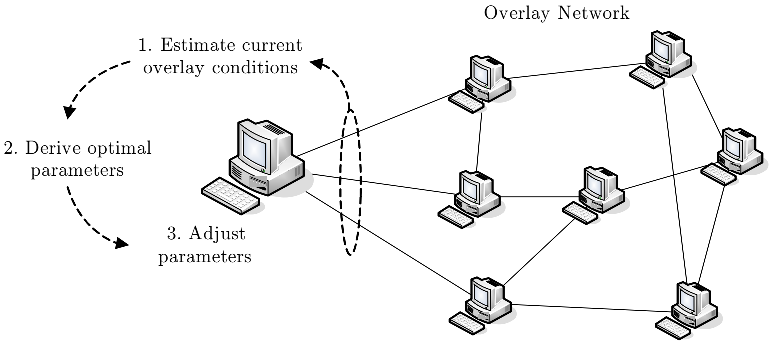

Overlay Network
1. Estimate current overlay conditions
2. Derive optimal parameters
3. Adjust parameters

Figure 4.1: *Self-organization concept for overlay networks*

and the stability of a deployed overlay system are inherently unknown, is one of the main reasons why telecommunication carriers are still hesitant to build distributed applications based on structured overlay networks. Besides our passive estimation algorithms, we therefore also present an entirely novel and scalable approach to actively create a snapshot of a deployed overlay network. Using our algorithm, a provider can then either monitor the entire system or just survey a specific part of the system.

## 4.1 Problem Formulation and Related Work

Structured p2p overlay networks have been designed to scale with the number of participating peers. The real issue concerning such overlay networks therefore lies in the management and maintenance of their stability, robustness, and redundancy. Blake and Rodrigues [37] showed that the real scalability problem can be found in the service bandwidth needed to maintain redundancy and stability in dynamic overlay networks. However, especially in these dynamic networks it is most important to know the current churn rate as well as the current size of the network to be able to adjust the maintenance cost needed to obtain redundancy and stability. Therefore Mahajan et al. [92] investigated the trade-off between high maintenance cost and poor stability in dynamic networks. The results show that it is crucial to adapt parameters dynamically. The authors introduce an estimator for the size of Pastry based networks, that can in some way be extended to other networks like CAN or Chord. However, there was no mathematical treatment on the quality of the estimator nor any confidence intervals for the obtained results. Moreover since the estimator was primarily designed for Pastry networks, it does not exploit additional characteristics that are typical for other overlay structures like Chord rings.

A very simple passive estimator for the size of butterfly based p2p overlay networks is introduced by Malkhi et al. in [105]. Most approaches to estimate the overlay size, however, rely on active probing of the network. A distributed algorithm is presented by Horowitz et al. [106], where an additional logical ring among existing nodes is maintained and nodes exchange their estimates upon arrival and departure. Bawa et al. [107] estimate the size of general overlay networks by actively sending samples to other nodes and evaluating the answer statistics. Jelasity et al. [108] solve the same problem using a method which is based on the information flow through a peer and which heavily depends on the assumption of independence between the individual samples. Finally, Kostoulas et al. [109] combine an active as well as a passive method to estimate the system size in a tool called PeerCounter. The active algorithm spreads a gossip through the network, where each peer marks its distance in terms of hops from the initiator. The passive algorithm is based on the density of the peers in the identifier space. Current size estimators either lack a mathematical description, need additional overhead to actively probe the network or do not exploit all properties of ring based overlay structures. In Chapter 4.2 we therefore present a mathematical substantiated estimator, which is well adapted to the properties of the Chord algorithm.

Besides the mere size of the system, churn is an even more complex problem in structured overlay networks [24, 110]. Stutzbach et al. [111] showed that churn plays a crucial role in the design, operation, and evaluation of p2p systems. Krishnamurthy et al. [91] further characterized churn and found that it can be of different types, all of which should be studied in detail. The concept of temporary and permanent churn was presented by Tati et al. [112], which lead to temporary unavailability and permanent loss of resources, respectively. Rhea et al. [75] evaluated different DHT implementations on an emulated network and concluded that without proper modifications current overlay structures cannot handle realistic churn rates. Subsequently, Godfrey et al. [113] showed that, while it is possible to minimize churn (e.g. by selecting a uniform-random replacement whenever an overlay neighbor fails), one cannot entirely avoid it. Zhuang et al. [114] discuss and compare different algorithms to minimize the node failure detection time in distributed overlay networks. They conclude that algorithms which share information in times of node failures improve the detection time at the cost of an increased control overhead.

The actual user behavior in a real system heavily depends on the kind of service being offered. In this context, Gummadi et al. [29] showed that p2p users behave essentially different from web users. For a typical filesharing application, they found a median session time of only 2.4 minutes and a 90th percentile of 28.25 minutes for sessions during which a peer was actively retrieving files. Their findings for large requests, however, showed that less than 10 percent are completed in an hour, 50 percent take more than a day, and nearly 20 percent of users are willing to wait a week for their downloads to complete. Additionally, Bhagwan et al. [110] argue that availability is not well-modeled by a single-parameter distribution, but instead is at least a combination of two time-varying distributions. This is supported by the observation that failure rates vary significantly with both daily and weekly patterns and that the failure rate in open systems is more than an order of magnitude higher than in a corporate environment [93]. To be able to compare the performance of different selection strategies for overlay neighbors, Godfrey et al. [113] present a definition of churn which reflects the global number of changes within a time interval $\Delta t$:

$$C = \frac{1}{\Delta t} \cdot \sum_{events\ i} \frac{|U_{i-1} \ominus U_i|}{\max\{|U_{i-1}|, |U_i|\}}.$$

Thereby $U_i$ is the set of online nodes which are in use after the $i$th change and $\ominus$ is the symmetric set difference. While the definition is very useful in simulations which possess a global view on the system, it cannot be used by an estimator which can only rely on local information. A simple method to estimate churn in a deployed system was introduced by Ghinita et al. [115] but not evaluated in detail.

In general, the problem of monitoring an overlay network from a central location is far from being solved. Sing et al. [116] give a good overview of different approaches to monitor and debug distributed systems. Other works aim at using overlays for decentralized network management [117]. Chen et al. [118] describe how to efficiently monitor all paths in a network using an overlay topology. Lim et al. [119] also use a distributed structure to monitor IP flows and end-to-end service quality. Renesse et al. [120] introduce Astrolabe, an overlay network specifically designed to monitor and report the dynamically changing state of a collection of distributed resources. This approach is further extended to a distributed information management system which aggregates information in large-scale networked systems by Yalagandula et al. [36]. Tang et al. [121] propose to cluster overlay nodes based on their geographic location for aggregating and delivering events with the minimum latency and cost. However, none of the above mechanisms implicitly monitors the status of the overlay itself. Stutzbach et al. [122] introduce a crawling-based approach to query Gnutella-like networks, which is limited to unstructured overlays. While structured overlays may be queried peer by peer from a central position, like e.g. in the CoMon project [123], the approach does clearly not scale to larger networks. In Chapter 4.4, however, we exploit the special features of structured p2p overlays and present an entirely novel and scalable approach to create a snapshot of a deployed overlay network.

## 4.2 Estimating the Current Peer Population

In a structured overlay network, each peer maintains pointers to $r$ well defined peers in the overlay in order to maintain the stability and robustness of the overlay structure. According to [45] the stability of a Chord ring can be obtained with high probability as long as $r = \Omega(\log_2(n))$, where $n$ is the current peer population of the Chord ring. In practice a peer either has to choose the parameter $r$ large enough to be able to handle the maximum possible ring size or has to adapt $r$ on the fly. Choosing a large constant value for $r$ results in high maintenance cost in the majority of cases, or insufficient stability in larger than expected overlay networks. To autonomically adapt the size of its neighborlist to $r = \Omega(\log_2(n))$ a single peer needs to know the current size of the overlay network it is participating in. In the following we therefore introduce an estimator for the current size $n$ of a Chord ring based on local information like the peer's current neighborlist [22]. A participating peer can then use this estimate to adjust the size of its neighborlist to the current size of the overlay network. This way, the peer uses the optimal amount of maintenance overhead to guarantee a stable overlay given the current size of the network.

### 4.2.1 Analytical Model

The analytical framework of our model is based on both a peer's successor- and fingerlist. At first we have a closer look at the identifier space itself. We assume that a total of $n$ peers share the identifier space of length $N = 2^m$ and that, by the hash function, the position $S(z)$ of every peer $z$ is distributed uniformly in the identifier space. Accordingly, every identifier is occupied by a peer with probability $p = n/N$. Let $I(z) = S(z+1) - S(z)$ be the random variable describing the length of the interval between peer $z$ and peer $z+1$, i.e. the distance between two neighboring peers as illustrated in Figure 4.2. We assume a collision-free

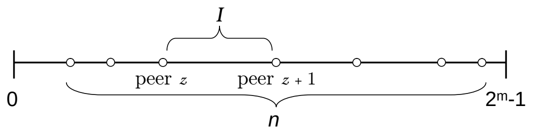


Figure 4.2: *The random variable $I$ describes the length of the interval between two peers.*

hash function, i.e. each peer has a unique identifier. Further, let us assume that without loss of generality peer $z$ has identifier 0, i.e. $S(z) = 0$. Then, the probability that another peer sits on position 1 is $(n-1)/(N-1)$ as there remain $n-1$ peers for $N-1$ free identifiers. The probability $P(z+1, i)$ that $S(z+1) = i$ is

$$P(z+1,i) = \left(1 - \frac{n-1}{N-1}\right)\left(1 - \frac{n-1}{N-2}\right)\cdots \tag{4.1}$$

$$\cdots\left(1 - \frac{n-1}{N-i+1}\right) \cdot \left(\frac{n-1}{N-i}\right) \tag{4.2}$$

$$\approx \left(1 - \frac{n}{2^m}\right)^{i-1} \cdot \frac{n}{2^m} \approx \left(1 - \frac{n}{2^m}\right)^{i} \cdot \frac{n}{2^m}. \tag{4.3}$$

The first approximation is justified as $n >> 1$ and $N >> i$. The second approximation is justified as on average $i = \frac{N}{n} >> 1$. Thus, we can conclude that the interval $I(z)$ between a peer and its direct neighbor is approximately geometric with parameter $p$:

$$I(z) \sim geom(p) \text{ where } p = \frac{n}{2^m}. \tag{4.4}$$

We validate this approximation by generating 10000 snapshots of random Chord rings with 1000, 10000, and 100000 peers in an identifier space of size $2^{160}$. Peer $z$ has identifier 0. We evaluate the distance to peer $z+1$ and refer to this distance as interval 1, which is equal to $S(z+1) - S(z)$. Figure 4.3 compares the simulated distribution to the theoretical geometric distribution. Since the curves match exactly when plotted on a linear scale we use a log-log scale. Considering the magnitude of the interval sizes and probabilities, the geometric distribution and the simulated distribution are almost identical. The dithering in the simulated curve comes from the limited amount of values that we gain from the simulations.

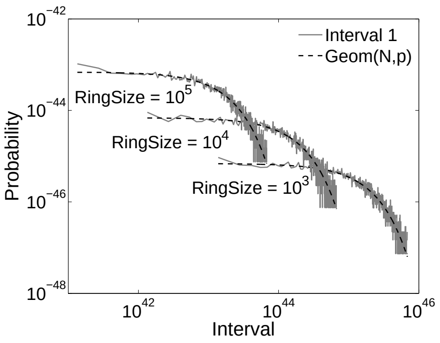


Figure 4.3: *Interval 1 is well-approximated by the geometric distribution.*

Ideally, peer $z$ does not only know its direct neighbor but the next $r = \lceil \log_2(n) \rceil$ neighbors and can calculate the distances between them. The probability that the location of peer $z + 2$ is directly after peer $z + 1$ is

$$\frac{n-2}{N-(S(z+1)-S(z))} \tag{4.5}$$

as there are $n - 2$ unknown peers and

$$(N-(S(z+1)-S(z))) \tag{4.6}$$

free identifiers remaining. Consequently, from peer $z$'s point of view interval $I(z + 1)$ depends on interval $I(z)$. However, we can argue again that due to the large size of the identifier space

$$\frac{n-2}{N-(S(z+1)-S(z))} \approx \frac{n}{N} = p. \tag{4.7}$$

Thus, the intervals between all $r$ neighbors of Peer $z$ are iid and we introduce the random variable $I$ for an arbitrary interval between two neighbored peers.

In Figure 4.4 we validate this approximation by means of the cumulative distribution function (CDF) of interval 1 and interval $r$, i.e. the interval between the last two successors. We can see that the curves for both intervals match very well with the geometric distribution independent of the ring size. The simulated curves start with a probability of $1e-4$ as we generated 10000 snapshots. Note that the distribution of 99% of the intervals (CDF$\geq 1e-2$) coincides with the geometric distribution.

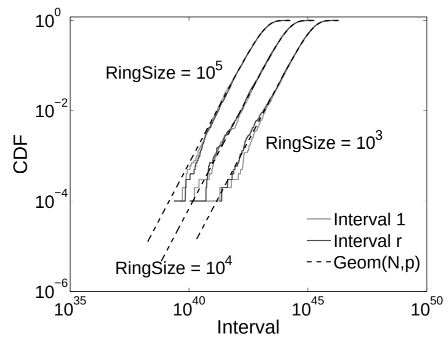


Figure 4.4: *Interval 1 and Interval r follow a geometric distribution.*

The main idea of our algorithm is to estimate the parameter $p$ of the geometric distribution of $I$. We denote the estimated value of $p$ as $\widehat{p}$. From this we can then conclude that

$$\widehat{n} = \widehat{p} \cdot 2^m. \tag{4.8}$$

To be able to estimate $p$ we need to obtain realizations of $I$, which can be gathered by looking at our neighborlist.

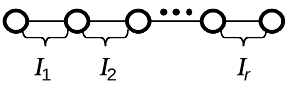


Figure 4.5: *Realizations of the random variable $I$.*

As shown in Figure 4.5 the intervals between a peer's $r$ immediate successors can be regarded as $r$ different realizations of the random variable $I$. More realizations of $I$ can be found if we have a closer look at a peer's fingerlist. As has been shown in [45] only $O(\log_2(n))$ of those $\log_2(m)$ fingers are actually different, i.e. are actually pointing to different peers. This is due to the fact, that especially the first fingers tend to coincide with a peer's successorlist. The interesting fact concerning our estimator, however, is that the actual position of the *i-th* finger on the ring is different from its theoretical position $id_z + 2^{i-1}$.

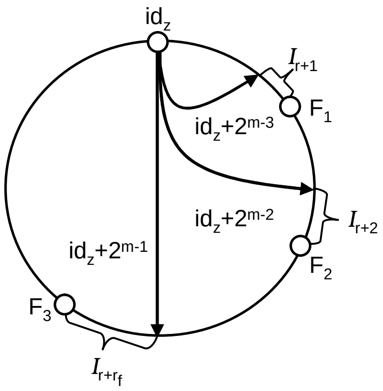


Figure 4.6: *Distance between theoretical and actual position of the* i-th *finger.*

Figure 4.6 illustrates this issue in detail. The figure shows three exemplary fingers for a peer $z$ pointing to $id_z + 2^{m-3}, id_z + 2^{m-2}$, and $id_z + 2^{m-1}$ respectively. As we can see the actual positions of the finger peers $F_1$, $F_2$, and $F_3$ are different to the fingers theoretical positions. This distance, however, can be interpreted as another realization of the geometrically distributed random variable $I$.

As stated above we already know that the length of the interval between a finger $F_i$ and the previous peer on the ring is geometrically distributed. If we now choose a random point in this interval, due to the memoryless property of the geometric distribution, the interval between the theoretical position of the finger and the actual finger is as well geometrically distributed with the same parameter $p$ as illustrated in Figure 4.7.

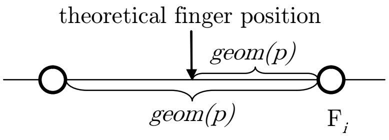


Figure 4.7: *Memoryless property of the geometric distribution.*

Again, we validate this assumption by means of the snapshots we used above. Figure 4.8 compares the distances of the theoretical and actual finger positions to the geometric distribution. We consider only those fingers that don't coincide with the successorlist. The figure shows the geometric distribution with regard to three different ring sizes. Each of these distributions is compared to the simulated distributions of each finger. Note that there are more simulated curves for the ring with 10000 peers than with 1000 peers, as there are more distinct fingers in larger rings as stated above. Again the plot is presented on a log-log scale, since the curves are effectively identical on a linear scale. By means of the geometric distribution of the finger intervals, we obtain another $r_f \approx \log_2(n)$ realizations of $I$ from a peer's fingertable, leaving us with a total of $r + r_f$ different realizations of the random variable $I$.

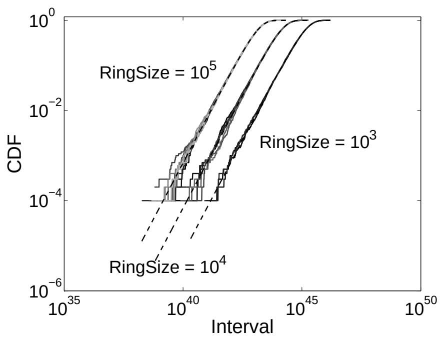


Figure 4.8: *The interval between actual finger position and theoretical finger position is geometric.*

### 4.2.2 Maximum Likelihood Estimation

The main goal of this section is to introduce an estimator $\widehat{n}$ for the current size of a Chord ring. This estimator can then be used to dynamically adjust the estimated necessary size $\widehat{r} = \log_2(\widehat{n})$ of a peer's successorlist. Since the estimator is based on a peers successor- and fingerlist and those lists in turn are adjusted according to the estimator, we assume that to get started, a peer is notified about the current size of the Chord ring by its immediate successor when first entering the network. In this section we show how to estimate the parameter $p$ of the geometric distribution of $I$ using a maximum-likelihood estimator (MLE). The MLE is used since we already know that the random variable $I$ is geometrically distributed but the parameter $p$ is still unknown. The basis for the MLE is a likelihood function $L(p)$ which is defined as follows:

$$L(p) = f_p(I_1) f_p(I_2) \cdots f_p(I_j), \tag{4.9}$$

where $f_p(I)$ is the probability mass function with parameter $p$ and $j$ is the number of observations made. The MLE $\widehat{p}$ of the unknown value of $p$ is then defined to be the value that maximizes the likelihood function $L(p)$. That is,

$$L(\widehat{p}) \geq L(p) \tag{4.10}$$

for all possible values of $p$. In our case we have

$$f_p(I) = (1-p)^I \cdot p \tag{4.11}$$

and

$$L(p) = (1-p)^{\sum_{i=1}^{r+r_f} I_i} \cdot p^{r+r_f}. \tag{4.12}$$

As has been shown in [124] in this case the MLE can be computed as

$$\widehat{p} = \frac{1}{\overline{I}(r+r_f)+1}, \tag{4.13}$$

where $\overline{I}(r+r_f)$ is the sample mean. With $\widehat{p}$ we can then estimate the current size $\widehat{n} = \widehat{p} \cdot 2^m$ of the Chord ring. Finally $\widehat{n}$ will be used to determine the number of successors the peer is going to maintain. The size of the successor-list will be set to

$$\widehat{r} = \lceil log_2(\widehat{n}) \rceil . \tag{4.14}$$

An obvious advantage of this approach is that the size of the successor-list is not as sensitive to errors as the estimated size of the Chord ring itself. That is due to the fact that the size of the successor list is logarithmically dependent on the size of the Chord ring. In practice a peer is going to use this estimator to set the size of its successor-list as follows. When first entering the Chord ring, a peer learns the current size of the Chord ring from its direct neighbors and adjusts the size of its successor-list accordingly. Afterwards it periodically uses the MLE $\widehat{p}$ to estimate the current size of the Chord ring and dynamically adapts the size of its successor-list. The disadvantage is that so far we cannot make any statement of how good the MLE $\widehat{p}$ estimates the actual size of the ring. Therefore we build confidence intervals for $\widehat{p}$. The $100(1-\alpha)$ confidence interval [124] for $\widehat{p}$ is given by

$$\widehat{p} \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{\widehat{p}^2(1-\widehat{p})}{r+r_f}}, \tag{4.15}$$

where $z_{1-\frac{\alpha}{2}}$ (for $0 < \alpha < 1$) is the upper $1 - \frac{\alpha}{2}$ critical point for a standard normal random variable.

However, the consequences of underestimating the real value of $p$ are by far more severe than the consequences of overestimating the real value of $p$. The main reason for this is that a successor-list which is too small has a negative effect on the stability of the Chord ring. A successor-list which is too large, on the other hand, only results in some additional overhead. To minimize the danger of underestimating $n$ we use the upper limit of the confidence interval to estimate $n$:

$$\widehat{n}_+ = \left(\widehat{p} + z_{1-\frac{\alpha}{2}} \sqrt{\frac{\widehat{p}^2(1-\widehat{p})}{r+r_f}}\right) 2^m. \tag{4.16}$$

This $\widehat{n}_+$ is then used to calculate the size $\widehat{r}_+$ of the successor-list as

$$\widehat{r}_+ = \lceil log_2(\widehat{n_+}) \rceil \tag{4.17}$$

Again, we round up to minimize the probability of underestimating the real value of $r$. The next section summarizes how the estimator performs in an actual Chord implementation.

## 4.2.3 Accuracy of the Estimate

In this section we show the results obtained by our simulations. If not stated otherwise, each snapshot of our simulations is done by uniformly placing $n$ peers into the identifier space of length $2^m$. Then the distances between the first $r$ consecutive peers are calculated and given as input to our estimator. We regard different ring sizes to see how the estimator scales to larger networks. Furthermore, we evaluate the difference between the upper and lower limit of the estimator and study the influence of the corresponding confidence levels. Additionally, we investigate how accurate the estimator and its upper bound are able to estimate the actually required number of successors. Finally we study the influence of churn by varying the number of successors a peer maintains.

To see how accurate our estimator $\widehat{n}$ approaches the current ring size we generated 10000 snapshots of a specific ring size $n$. We then set the number of successors to the ideal value $r = \lceil \log_2(n) \rceil$ and compared the estimated ring sizes to the actual ring size. Figure 4.9 shows the results of our simulations for a given ring size of 10000 and a successorlist of size 14. As can be seen in the figure,

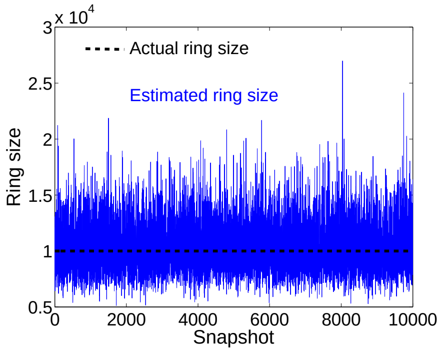


Figure 4.9: *10000 estimates of the ring size as compared to the actual ringsize*

our estimator $\widehat{n}$ is well in the right order of magnitude and roughly oscillates between $0.5n$ and $2n$. Depending on the range of application, however, under- or overestimating might be crucial to the performance of the application on top of the estimator.

In Figure 4.10, we therefore compare the lower bound $\widehat{n}_-$ and the upper bound $\widehat{n}_+$ of our estimator to the actual ring size, again using 10000 snapshots of a ring of size 10000. The confidence level in this example is set to 95%. The lower

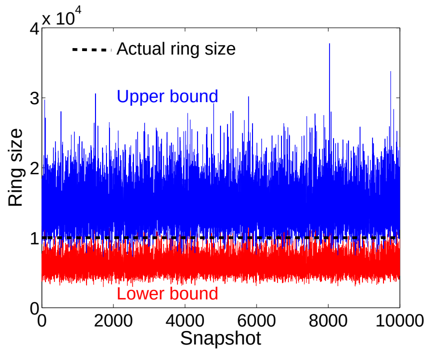


Figure 4.10: *The lower and upper bound of the estimator with a confidence level of 95% as compared to the actual ring size*

bound $\widehat{n}_-$ of the estimator stays beneath the actual size of the ring with high probability, whereas the upper bound ranges between $n$ and $2n$ to $3n$, underestimating the real value of $n$ at times.

To analyze the probability that the lower bound overestimates and the upper bound underestimates the actual ring size we plot the sorted snapshots in Figure 4.11. The Figure shows the normalized results obtained for the estimator and its lower and upper bounds for three different ring sizes. Again a confidence level of 95% is used. The part of the upper bound beneath the dotted line represents the number of times the upper bound underestimates the actual ring size, the part of the lower bound above the dotted line the number of times the lower bound overestimates the actual ring size, respectively. Note that the median of the estimator itself approximately intersects with the actual ring size as indicated by the vertical line. This justifies our assumption that the random variable $I$ is approximately geometric since the median of an estimator based on exactly geometric intervals would exactly intersect with the actual ring size.

Another important fact which can be derived from the figure is that we over-

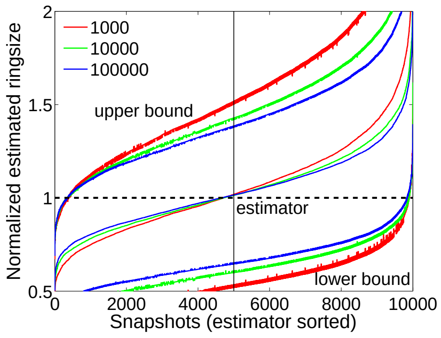


Figure 4.11: *Sorted estimates gained by the estimator.*

and underestimate the actual ring size less significantly in larger networks. This is of course due to the fact that we use more neighbors in larger networks. The primary reason, however, lies in the fact, that a peer also has more distinct fingers and thus more uncorrelated realizations of $I$ in larger networks. Note that the tiny spikes in the graphs of the lower and upper bound arise since we only sorted the estimator itself and plotted the corresponding upper and lower bounds.

As can be seen in Figure 4.12 the lower and upper bound of the estimator can be fine tuned by adjusting the confidence level. The confidence level in this example was varied between 50% and 99%. The higher we set the confidence level, the more the curves of the upper and lower bound drift away from the estimator. This means that the higher we choose the confidence level, the less frequently we will under- and overestimate the actual ring size. However, the drawback of a high confidence level is that the estimates of the upper and lower bound get less precise. The trade-off between overlay stability and maintenance overhead can thus be fine tuned by means of the confidence level.

The most obvious application of the estimator is the dynamic adaptation of a peers successorlist. Since a peer ideally maintains a list of at least $r = \lceil \log_2(n) \rceil$ neighbors the estimate in this case does only depend logarithmically on the esti-

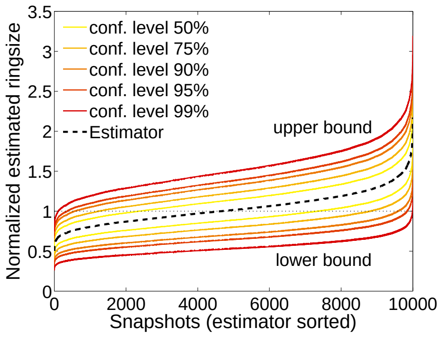


Figure 4.12: *Influence of the confidence level on the upper and lower bound.*

mate of $n$. As it is more critical to underestimate than to overestimate the required number of successors, we will concentrate on the estimator and its upper bound in the following. Since we additionally round the estimate for the upper bound

$$\widehat{r}_+ = \lceil \log_2(\widehat{n_+}) \rceil ,$$

we set the confidence level to moderate 95% in the remainder of this section. Figures 4.13 and 4.14 show the estimated number of required neighbors in a network of size $10^4$ and $10^5$. In Figure 4.13 the actually required number of neighbors is $14 = \lceil \log_2(10^4) \rceil$. The regular estimator provides the correct number of neighbors in over 80% of all cases. However, in almost 20% of the snapshots the estimator would set the size of the successorlist to 13, one peer less than needed. In order to minimize the danger of underestimating the required number of successors, one should therefore use the number of neighbors estimated by the upper bound. While the upper bound does almost never underestimate in the current example, it tends to overestimate more frequently than the regular estimator.

In a ring of size $10^5$ (see Figure 4.14) the upper bound overestimates the required number of neighbors by 1 in over 60% of all cases. In return it never

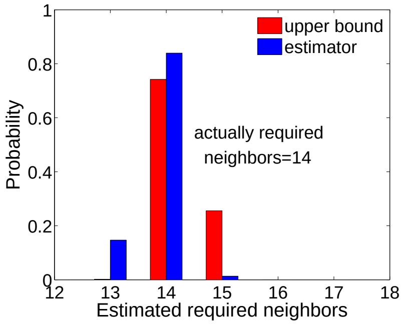


Figure 4.13: *Estimated required neighbors for upper bound and regular estimator.*

understimates the actually required number of successors. The regular estimator on the other hand again underestimates the actual value, even though only in very few cases. Note that in about 90% of all cases the regular estimator meets the actually required number of neighbors. Given the fact that the upper bound only

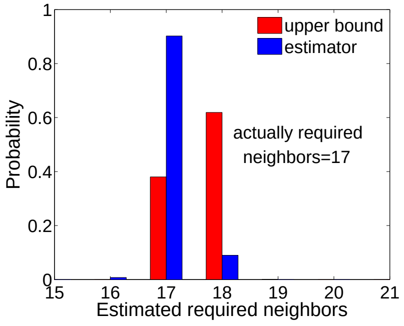


Figure 4.14: *Comparison of upper bound and regular estimator for* $10^5$ *peers.*

slightly overestimates the desired number of neighbors, we suggest to prefer the upper bound to the regular estimator in critical applications.

So far the results presented in this section were based on the ideal number of neighbors in the given networks. To see how the estimator performs when relying on an unideal number of neighbors, we again simulate 10000 snapshots for a ring of size $10^4$ and evaluate the estimator and its upper bound based on successorlists of different size. Thereby the number of successors used as input to the estimator ranges between 1 and 20 successors. The actually required number of neighbors in this example is again 14. Figure 4.15 shows the results corresponding to the regular estimator. The bars represent the results obtained by using 1 to 20 neighbors. The darker the color, the more neighbors have been used as input to the estimator. Obviously, the more neighbors the estimator can rely on, the better the

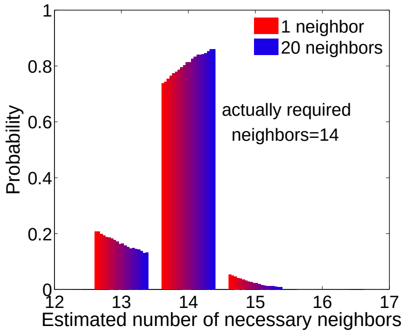


Figure 4.15: *Results obtained by using 1 to 20 neighbors for* $10^4$ *peers.*

obtained results become. That is, the more realizations of $I$ we can give as an input to the estimator, the more precisely it calculates the actually required number of neighbors and the less often it over- and underestimates this value. Still the estimator underestimates the actual value, even in the case of 20 neighbors.

For comparison, the results obtained by the upper bound are summarized in Figure 4.16. The bars increase and decrease more rapidly than the bars in the last figure. That is due to the fact that, the more realizations of $I$ we obtain, the smaller the confidence interval is going to be. Thus the upper bound will converge to the estimator. Having a closer look at the Figure, we also notice that the probability that the upper bound underestimates the required number of neighbors is negligible but not entirely zero. Obviously, this is especially noticeable

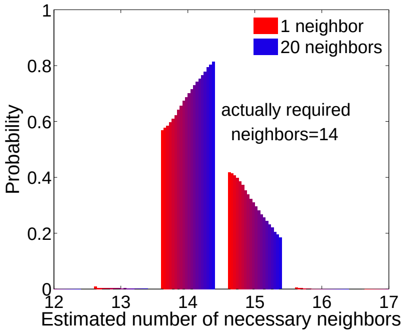


Figure 4.16: *The upper bound is more sensitive to the number of neighbors than the regular estimator*

for small successorlists, since a small successorlist also means fewer realizations of $I$. Moreover since $13 = \log_2(8192)$ all estimated values of $n < 8913$ will result in an underestimation of $r$. Thus, the estimator can not fully take advantage of the mathematical round step. Note that, independent of the size of the successorlist, the upper bound is able to rely on the realizations of $I$ gained by its fingerlist. Thus, it supplies an applicable estimate of the required number of neighbors independent of the number of successors used as input.

## 4.3 Assessing the User-Behavior

The dynamic behavior of the users causes fluctuations in the overlay network which lead to inconsistencies, lost messages, and ultimately to a decreased user-perceived quality. As a consequence, structured overlay algorithms require more maintenance traffic when the churn rate is high. However, p2p networks operate without a centralized control unit and each peer has only a limited view of the entire network, usually not being aware of the current churn rate in the network. Thus, a peer should be able to estimate the churn rate from the limited information that is available and autonomously react to high churn situations by increasing the maintenance traffic.

In this chapter, we propose a fully distributed algorithm for peers to assess the behavior of the user and estimate the churn rate by exchanging measurement observations among neighbors [7]. The overlay network itself is used as a memory for the estimate while each online peer contributes to updated measurements of the estimator. The advantage of this method is that it operates passively, i.e. there are no additional entities required to monitor online and offline periods of the peers and the generated overhead is negligible.

### 4.3.1 Algorithm to Capture the Fluctuations in the Overlay

In general, a good estimator for the churn in the system must in some way capture the fluctuations in the overlay structure and then deduce an estimate for the churn rate from these observations. Thereby, we must take into account that an individual peer does not have any global knowledge about the state of the system but has to rely on a very limited view of the network. In structured p2p networks, each peer has periodic contact to a specific number of overlay neighbors, like the *successors* in Chord, the *$k$-bucket entries* in Kademlia, or the *leafs* in Pastry. The basic principle of our estimator is to monitor the changes in this neighbor list and use them to derive the current churn rate. Thereby, we model the behavior of a

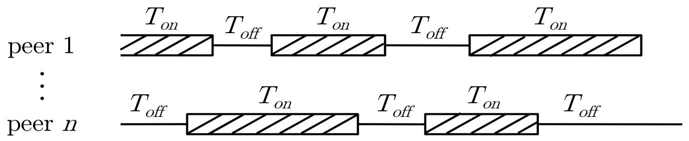


Figure 4.17: $T_{on}$ *and* $T_{off}$ *describe the online and offline times of the* $n$ *peers.*

peer using two random variables $T_{on}$ and $T_{off}$ which describe the duration of an online session and an offline session as shown in Figure 4.17.

We assume that each online peer $p$ stores pointers to $c$ well defined overlay neighbors (or contacts) which are specified by the individual DHT protocols. To deal with stale entries and to maintain the structure of the overlay, peer $p$ periodically contacts a special subset of its neighbors every $t_{stab}$ seconds and runs an appropriate stabilization algorithm. This corresponds, e.g., to bucket refreshes in Kademlia or the stabilization with the direct successor in Chord. At each of these stabilization instants the peer synchronizes its neighbor list with those of its contacts. The main idea of our estimator is to monitor the changes in the neighbor list and thereby collect different realizations of the random variables $T_{on}$ and $T_{off}$. That is, a peer $p$ observes the online and offline session times of its overlay neighbors. Thereby, $obs(i)$ is the value of the $i$th observation made by the peer and $time(i)$ is the time at which the observation was made. The observation history is stored in a list which contains up to $k_{max}$ entries. Furthermore, a peer stores the time stamps $t^p_{on}$ and $t^p_{off}$ which correspond to the time peer $p$ itself joined or departed from the overlay, respectively.

The join rate is the less important one of the two components of churn. The shorter a peer stays offline on average (i.e. the smaller $E[T_{off}]$) the higher is the join rate. To obtain realizations of $T_{off}$, a peer stores the time $t_{off}$ when it last went offline. The next time it goes online it calculates the duration of its offline session as $now - t_{off}$ and sends this value to its $c$ overlay neighbors. Figure 4.18 visualizes this concept. Note that the information can be piggybacked on other

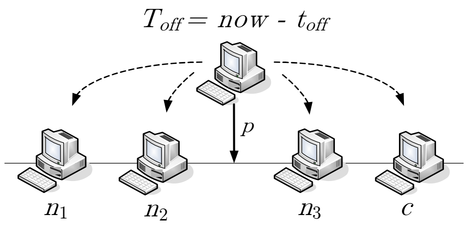


Figure 4.18: *Peer $p$ rejoins the network and sends its offline duration to its $c$ neighbors*

protocol messages to avoid unnecessary overhead.

Since failed nodes can no longer inform their overlay neighbors about their online duration, we are not in the position to directly obtain realizations of $T_{on}$. That is why we are looking for another passive way to collect realizations which is still independent of the applied DHT protocol. In a DHT system, a peer $p$ periodically contacts at least one neighbor $s$ to stabilize the overlay structure (cf. Step 1 in Figure 4.19). In Chord this would be the direct successor in a clockwise

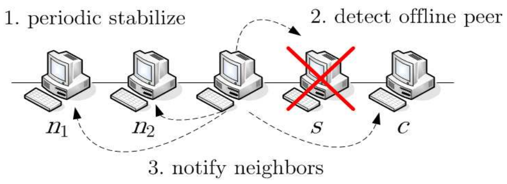


Figure 4.19: *Peer $p$ only monitors its direct neighbor $s$ but distributes its observations*

direction, in Kademlia the closest peer according to the XOR-metric. If, during one of its stabilization calls, $p$ notices that $s$ has gone offline (cf. Step 2 in Figure 4.19), it calculates the duration of the online session of peer $s$ as $now - t^s_{on}$, where $t^s_{on}$ is the time when peer $s$ went online. Peer $p$ then distributes this observation to all its overlay neighbors as shown in Step 3 in Figure 4.19. If the DHT applies some kind of peer down alert mechanism [75, 93], the information could also be piggybacked on the corresponding notify messages.

An obvious problem of this approach is that peer $p$ does not always naturally know $t^s_{on}$, the time when peer $s$ went online. This is for example true if $p$ went online after $s$ or if $s$ became the successor of $p$ due to churn in the network. For this reason each peer $s$ memorizes the time $t^s_{on}$ when it went online and sends this information to its new predecessor whenever it stabilizes with a new peer. To cope with the problem of asynchronous clocks it sends its current online time $now - t^s_{on}$. This way the error is in the order of magnitude of a network transmission and thus negligible in comparison to the online time of a peer. The advantage of this method to collect realizations of $T_{on}$ is that it only requires regular contact to one single neighbor.

When a peer joins the network, it first needs to obtain some observations before it can make a meaningful estimate of the churn rate. The problem is that the lifetime of the peer is in the same order of magnitude as the lifetime of its overlay neighbors. Thus, it is unlikely, that the peer is able to make enough observations during its lifetime. Therefore, we use the overlay network as a memory of already obtained observations to maintain them beyond the lifetime of the peer. If a new peer joins the overlay it downloads the current list of observations from its direct successor. This way the observations persist in the overlay and a new peer can already start with a useful estimate which reflects the current churn rate in the network.

Another way to maintain the persistence of the observations is to invest more overhead by periodically contacting a number of peers instead of just one. Mahajan et al. [92] present an algorithm which relies on the fact that a peer $p$ continuously observes $c$ overlay neighbors as shown in Figure 4.20. Such a peer should on average observe one failure every

$$\Delta t = \frac{1}{c} \cdot E[T_{on}] \tag{4.18}$$

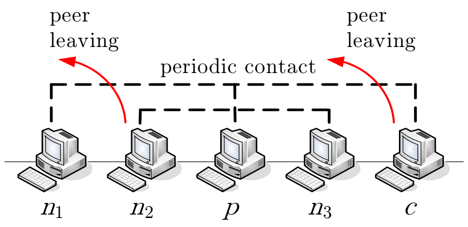


Figure 4.20: *Peer $p$ periodically monitors the changes in its overlay neighborhood*

seconds. Thus, if peer $p$ observes $k$ failures within a time period of $\Delta t$ seconds, the mean online time of a peer can be estimated as:

$$\widehat{E}[T_{on}] = \frac{c \cdot \Delta t}{k} = \frac{c \cdot (time(k) - time(1))}{k} \tag{4.19}$$

where $time(i)$ is the time of the $i$th observed node failure. In addition to the periodic contact to $c$ neighbors, the algorithm also has to struggle with the problem of obtaining enough observations during the lifetime of the peer. A possible solution to the problem is to piggyback the current estimate on protocol messages and to set the own estimate to the median of the estimates received from other nodes in the overlay [93].

### 4.3.2 Analytical Derivation of the Churn Rate

In this section we discuss the possibilities of a peer to deduce the current churn rate from the observations it has made. We will use the following notation: For a random variable $X$, we denote $x(t)$ as the probability density function, $X(t)$ as the cumulative density function, and $E[X]$ as the mean. Estimated values will be marked using a hat as in $\widehat{E}[X]$, which describes an estimate for the mean of $X$.

Once a peer has obtained a list of observations $obs(i), i = 1, \ldots, k$ of the random variables $T_{on}$ and $T_{off}$, it needs a mechanism to derive an estimate of the current churn rate based on its a priori knowledge. If it cannot make any reasonable assumptions about the distribution of the random variable, it has to rely on robust estimates like the empirical mean and the empirical standard deviation.

$$\widehat{E}[T_{on}] = \frac{1}{k} \cdot \sum_{i=1}^{k} obs(i) \tag{4.20}$$

$$\widehat{\sigma}(T_{on}) = \sqrt{\frac{1}{k-1} \sum_{i=1}^{k} \left( obs(i) - \widehat{E}[T_{on}] \right)^2} \,. \tag{4.21}$$

To evaluate the accuracy of the estimate we can construct the $100(1-\alpha)$ percent confidence interval for the estimated mean of $T_{on}$ as

$$u(k,\alpha) = \widehat{E}[T_{on}] + t_{k-1,1-\frac{\alpha}{2}} \cdot \frac{\widehat{\sigma}(T_{on})}{\sqrt{k}} \tag{4.22}$$

$$l(k,\alpha) = \widehat{E}[T_{on}] - t_{k-1,1-\frac{\alpha}{2}} \cdot \frac{\widehat{\sigma}(T_{on})}{\sqrt{k}} \,, \tag{4.23}$$

where $t_{k-1,1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$ critical point of the $t$ distribution with $k-1$ degrees of freedom. Depending on the intended purpose of the estimator, it might be crucial that the estimator does not over- or underestimate the actual value too often. In such a case, the upper and the lower bound of the confidence interval can themselves be used as estimates, as already discussed in Chapter 4.2.3.

While the mean gives a first idea about the churn in the system, the main purpose is to use the estimate to self-tune the parameters of the DHT or to calculate the probability of certain events. This usually requires knowledge of the entire distribution or at least of some important quantiles. For example, to calculate the probability that an overlay neighbor will no longer be reachable at the next stabilization instant, we need to know the probability that this contact will stay online for less than $t_{stab}$ additional seconds. In general, this probability depends on how often the peer itself already stabilized with this neighbor. In particular, the probability $p_i$ that an overlay neighbor will no longer be reachable at the $i$th stabilization instant can be calculated as follows:

$$
\begin{aligned}
p_i &= P\left(T_{on,r} < i \cdot t_{stab} | T_{on,r} > (i-1) \cdot t_{stab}\right) && (4.24)\\
&= \frac{P\left((i-1) \cdot t_{stab} < T_{on,r} < i \cdot t_{stab}\right)}{P\left(T_{on,r} > (i-1) \cdot t_{stab}\right)} \; , && (4.25)
\end{aligned}
$$

where $T_{on,r}$ describes the forward recurrence time for the random variable $T_{on}$. In case of exponentially distributed online times $T_{on,r} = T_{on}$ and the above equation can be simplified to:

$$
\begin{aligned}
p_i &= \frac{\left(1 - e^{-\lambda_{on} \cdot i \cdot t_{stab}}\right) - \left(1 - e^{-\lambda_{on} \cdot (i-1) \cdot t_{stab}}\right)}{1 - \left(1 - e^{-\lambda_{on} \cdot (i-1) \cdot t_{stab}}\right)} && (4.26)\\
&= 1 - e^{-\lambda_{on} \cdot t_{stab}} = p \; . && (4.27)
\end{aligned}
$$

Thus, for exponentially distributed online times $p_i = p$ for all $i$ and an unbiased point estimator for this probability is given by Equation 4.28.

$$
\begin{aligned}
\widehat{p} &= \widehat{P}\left(T_{on} < t_{stab}\right)\\
&= \frac{1}{k} \left| \left\{ T_{on}^i : T_{on}^i < t_{stab} \quad \text{for } i = 1, 2, ..., k \right\} \right| \; . && (4.28)
\end{aligned}
$$

The $100(1-\alpha)$ confidence interval for $\widehat{p}$ can be calculated using the following bounds:

$$
\begin{aligned}
u(k,\alpha) &= \widehat{p} + z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{p}(1-\widehat{p})}{k}} && (4.29)\\
l(k,\alpha) &= \widehat{p} - z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{p}(1-\widehat{p})}{k}} \; , && (4.30)
\end{aligned}
$$

where $z_{1-\frac{\alpha}{2}}$ is the $1 - \frac{\alpha}{2}$ critical point for a standard normal random variable. In case over- or underestimating has serious consequences for the applied application, the limits of the confidence interval can again be used as estimates themselves.

In some cases an application requires knowledge of the entire distribution function of the online time of the peers. If the type of distribution is known a priori, the peer can use the corresponding Maximum Likelihood Estimator (MLE) to estimate the corresponding parameters of the distribution. In the most often assumed case of an exponential distribution, the MLE is known to be the sample mean. For other typical distributions like the log-normal distribution Log-N($\mu, \sigma^2$) the MLE becomes more complicated, but can usually still be calculated using the information collected by the peer:

$$\widehat{\mu} = \frac{1}{k} \cdot \sum_{i=1}^{k} \ln\left(obs(i)\right) \tag{4.31}$$

$$\widehat{\sigma} = \left[ \frac{\sum_{i=1}^{k} \left(\ln\left(obs(i)\right) - \mu\right)^2}{k} \right]^{\frac{1}{2}} . \tag{4.32}$$

However, there is always the danger of assuming an incorrect distribution which would lead to correspondingly distorted results. A possibility to reduce this risk is to perform a hypothesis test [125] to verify that the type of distribution is actually the assumed one and only use an MLE if the test delivers a positive result. In general, however, the actual type of distribution is not known or a superposition of multiple distributions. In this case, a peer has to rely on an estimate of the quantiles [126] of the online distribution. Let $T_{on}^{1}, T_{on}^{2}, ..., T_{on}^{k}$ be the $k$ observations in the history of a peer and let $T_{on}^{(1)}, T_{on}^{(2)}, ..., T_{on}^{(k)}$ be the ordered statistic, in such a way that $T_{on}^{(1)} < T_{on}^{(2)} < ... < T_{on}^{(k)}$. These sorted values of $T_{on}$ can then be taken as the $\frac{0.5}{k}$, $\frac{1.5}{k}$, ..., $\frac{k-0.5}{k}$ quantiles of the distribution of the online time. Quantiles for probabilities between $\frac{0.5}{k}$ and $\frac{k-0.5}{k}$ can be computed using linear interpolation, while the minimum or maximum values of $T_{on}$ are assigned to quantiles for probabilities outside that range.

The accuracy of the estimates heavily depends on $k$. The more observations a peer maintains in its history, the more accurate the estimate is going to be.

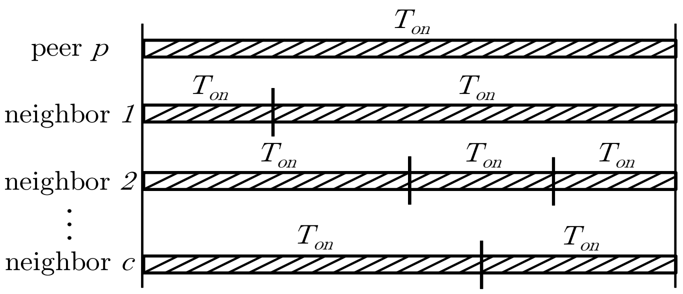


Figure 4.21: *Observations made by peer $p$ during its lifetime*

However, if the overlay network is not used as a memory for already made observations, a joining peer has to rely on its own observations. Therefore, it can either observe one specific peer and send the result to its $c$ overlay neighbors or directly observe $c$ peers itself. Figure 4.21 shows the online period of a peer $p$ and the $c$ overlay neighbors it observes during its lifetime. In the figure we assume a perfect stabilization algorithm. That is, an overlay neighbor which went offline is immediately replaced by another overlay peer.

To analyze the expected size of the history of a peer, we regard the random variable $X$ which describes the number of observations a peer makes during its lifetime. This number corresponds to the number of leave events in Figure 4.21. It can be computed as

$$P\left(X=i\right)=\int_0^\infty t_{on}(t)\cdot P\left(X=i|T_{on}=t\right)\,dt \tag{4.33}$$

where $t_{on}(t)$ is the probability density function of $T_{on}$. In the case of exponentially distributed online times, this can be written as

$$P\left(X=i\right)=\int_0^\infty \lambda e^{-\lambda t}\cdot\frac{(c\lambda t)^i}{i!}\cdot e^{-c\lambda t}\,dt \tag{4.34}$$

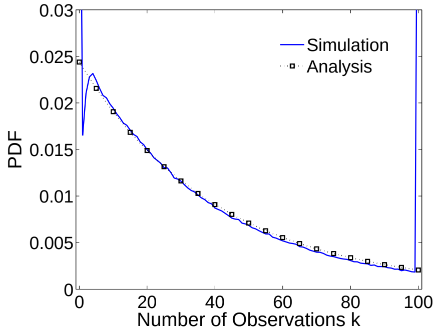


Figure 4.22: *Expected number of observations for* $c = 40$

since the number of departures in a fixed interval of length $t$ is Poisson distributed with $c \cdot \lambda$. The equation can be simplified to

$$\begin{aligned} P\left(X=i\right) &= \frac{c^i \lambda^{i+1}}{i!} \int_0^\infty t^i \cdot e^{-(c+1)\lambda t}\, dt \\ P(X=i) &= \frac{c^i \lambda^{i+1}}{i!} \frac{i!}{(\lambda \cdot (c+1))^{i+1}} \\ &= \frac{c^i}{(c+1)^{i+1}}. \end{aligned} \tag{4.35}$$

To compare this theoretical approximation to practical values, we simulated an overlay network with $T_{on} = 300$s, $t_{stab} = 30$s, and $c = 40$. The maximum size of the history was set to $k_{max} = 100$. Figure 4.22 shows the probability density function of $X$ for both the analysis and the simulation. It can be seen that the analysis matches the simulation very well except for the two peaks at the left and the right of the figure. The peak at 100 clearly results from the maximum size of the history. That is, in the simulation all probabilities for $P(X > 100)$ are added to $P(X = 100)$. The peak at 0 arises from the fact that while the analysis immediately takes offline peers into account, the first stabilization instant in the simulation occurs 30 seconds after the peer joined the network. Thus, all peers which stay online for less than 30 seconds, can never make an observation.

While so far we studied the accuracy of an estimator, it is also interesting to analyze how fast an estimator reacts to changes in the global churn rate. The more observations a peer makes per time unit, the faster it can react to such changes. This can be measured by looking at $T_{obs}^{leave}$, the time between two observed leave events, or $T_{obs}^{join}$, the time between two observed join events. In general, the collection of observations shows a self-organizing behavior. The more churn there is in the system, the more observations will be collected per time unit. If a peer monitors only its direct neighbor and shares its observations with $c$ overlay neighbors, the next observation is made as soon as one of $c+1$ monitored peers goes offline. Thus, the distribution of $T_{obs}^{leave}$ can be calculated as the minimum of $c+1$ forward recurrence times of $T_{\mathit{on}}$. Due to the memoryless property, the forward recurrence time of an exponentially distributed online time $T_{\mathit{on}}$ is also exponentially distributed with the same parameters. In this case the distribution of $T_{obs}^{leave}$ can be calculated as shown below.

$$P\left(T_{obs}^{leave} < t\right) = 1 - \prod_{1}^{c+1} P\left(T_{\mathit{on}} \geq t\right) = 1 - e^{-(c+1)\lambda t} \tag{4.36}$$

If the distribution is not known, we can still easily compute the mean of $T_{obs}^{join}$ and $T_{obs}^{leave}$. Each node which joins the network calculates its own offline time and additionally sends this observation to its $c$ contacts.

$$E\left[T_{obs}^{join}\right] = \frac{E[T_{\mathit{off}}]}{c+1} \tag{4.37}$$

The calculation is slightly more complicated for $T_{obs}^{leave}$ since the time when a peer actually observes that another peer is offline differs from the actual time the node left the overlay. Assuming that overlay neighbors are updated every $t_{stab}$

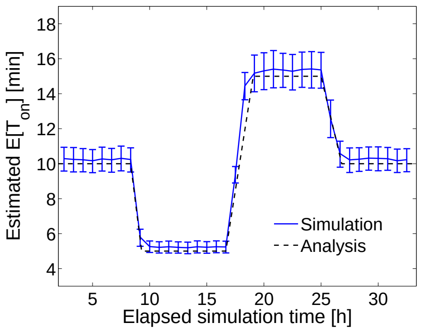


Figure 4.23: *Response time for* $c = 10$ *and* $k_{max} = 100$

seconds, the average error is

$$\epsilon_{on} = \frac{t_{stab}}{2}. \tag{4.38}$$

Thus, the mean of $T_{obs}^{leave}$ can be calculated as

$$E\left[T_{obs}^{leave}\right] = \frac{E[T_{on}] + \epsilon_{on}}{c+1}. \tag{4.39}$$

The above considerations can be used to approximate the expected time it takes the estimator to respond to a global change of the churn rate. When the mean online time of the peers changes from $E_{old}[T_{on}]$ to $E_{new}[T_{on}]$, we approximate the expected response time $E[R]$ by the time needed to collect $k_{max}$ new observations.

$$E[R] = E_{old}[T_{on,r}] + \frac{k_{max}}{c+1} \cdot (E_{new}[T_{on}] + \epsilon_{on}) \tag{4.40}$$

Figure 4.23 compares the analytical response time to that obtained from a simulation run. In the simulation we set $k_{max} = 100$, $c = 10$, $t_{stab} = 30$s and changed $E[T_{on}]$ from 10min to 5min to 15min and back to 10min after $8.33$h, $16.66$h, and $25$h of simulation time, respectively. The simulated curve shows the mean of the estimated $E[T_{on}]$ values of all peers, which were online at the corresponding time. The error bars represent the interquartile range, i.e. the difference between the third and first quartiles, as a measure of statistical dispersion. It can be seen that the estimator is able to capture the changes in the churn rate and that the time it takes to adjust to the new value complies with the analysis. Note that, due to the stabilization period of 30 seconds, the estimated values lie $\epsilon_{on} = 15$s above the actual value.

The response time can be improved by maintaining more overlay neighbors or by giving less weight to older values in the history. This can, e.g., be achieved by using an exponential weighted moving average [127], which applies weighting factors which decrease exponentially:

$$\widehat{E}_k[T_{on}] = \alpha \cdot obs(i) + (1 - \alpha) \cdot \widehat{E}_{k-1}[T_{on}] \ . \tag{4.41}$$

Thereby, the smoothing factor $\alpha$ determines the weight given to the latest observation.

## 4.3.3 Accuracy, Responsiveness, and Practicability

In the following evaluation, unless stated otherwise, we will always consider that the online and offline times of the users are exponentially distributed with mean $E[T_{on}]$ and $E[T_{off}]$, respectively. The default stabilization interval is $t_{stab} = 30$s and the size of the neighbor list is $c = 20$. We will further assume that there are 40000 initial peers and that $E[T_{on}] = E[T_{off}]$, resulting in an average of 20000 online peers at a time. Although our estimator yields results for both online and offline time, we will concentrate on estimating the online time $T_{on}$, since this is usually a more important parameter for the system performance. The estimation of $T_{\mathit{off}}$ can be calculated in an analogous way.

## Proof of Concept

The main purpose of this section is to show that the theoretic concept of the proposed estimator as described in Section 4.3.1 does work equally well in practice. We focus on Chord since it is the currently most studied structured overlay architecture. To model the stabilization algorithm, a peer synchronizes its neighbor list every $t_{stab} = 30$s with its direct successor. When a peer notices that another peer is offline, it notifies the peers in its neighbor list, piggybacking the observed online time in these messages. We furthermore consider a symmetric neighbor list, i.e. the number of peers in the successor list is the same as that of the predecessor list. This improves the stability of the Chord overlay and provides a better comparability of the result to symmetric overlays like Kademlia.

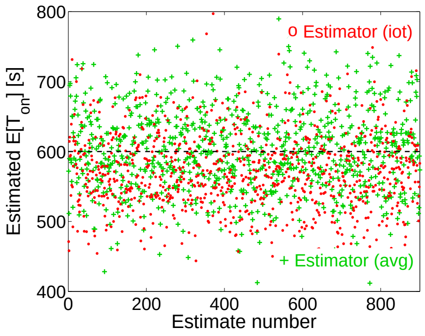


Figure 4.24: *Snapshot of mean online time obtained from avg and iot estimation*

Figure 4.24 plots the estimated values for $E[T_{\mathit{on}}]$ obtained by two different estimation methods. The green crosses, denoted by *avg*, show the mean of the observed values as given by Equation 4.20. The red circles, denoted by *iot*, are based on the mean time between two observations according to Equation 4.39.

The figure was created by picking 900 random peers leaving the overlay and calculating their estimates based on both estimation methods. We can see that both methods perform quite well, as the estimated values cluster around the actual average $T_{on}$ of 600s. In general the *avg* method always provides a robust estimate. The *iot* method, however, heavily depends on the implementation specifics of the stabilization routine. In our simulation experiments we do assume an idealized stabilization routine, neglecting packet loss and other (network) errors. In a real implementation the neighbor list can still contain wrong or offline entries after stabilization. In such a case the updates may be sent to less than $c$ contacts. If a peer, however, assumes that $c$ other peers contribute with their estimated values, it will receive too few estimates and heavily underestimate the actual churn rate using the *iot* method. Thus, while in theory it can be considered as a good method, it is in fact not very suitable in practice.

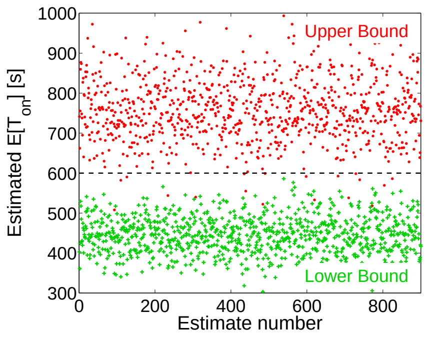


Figure 4.25: *Estimates of upper and lower 99% confidence levels*

In practice, too high or too low estimates might have critical consequences in terms of performance or even functionality, which is similar to the problematic in Chapter 4.2. In such a case it should be avoided that the estimator underestimates or overestimates the actual churn rate. This can be achieved by using the upper or lower bound of a specified confidence level instead of the estimated value itself.

Figure 4.25 shows the upper and lower bounds of the 99% confidence interval for the mean according to Equation 4.22 and Equation 4.23, respectively. As expected, the upper bound overestimates the actual value, while the lower bound underestimates it. The frequency at which the upper bound underestimates or the lower bound overestimates the actual value can be influenced by the confidence level. The higher the confidence level is chosen, the smaller is the probability for this to happen at the cost of more innacurate values.

## Accuracy of the Estimator

In this section we evaluate how the accuracy of the estimated results depends on different parameters. The key parameter we focus on is the size $k$ of the observation history, i.e., the number of samples that are used to obtain the estimate.

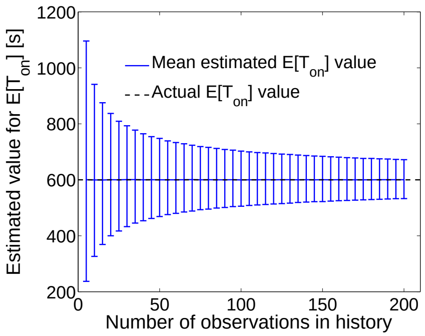


Figure 4.26: *Influence of the history size $k$ on the estimation accuracy*

In order to show the influence of $k$ on the accuracy of the estimate, we perform several simulation runs in which we vary the size $k$ of the history and for each $k$ examine the estimated $E[T_{on}]$ values from 10000 peers. The mean of these estimated values is shown in Figure 4.26. The error bars in the figure represent the sampled standard deviation obtained from the 10000 estimates. First of all, we can recognize that the method is robust for estimating the mean, as the mean estimated $E[T_{on}]$ value corresponds to the actual mean value of 600s. Furthermore, increasing the history size greatly improves the accuracy of the estimate in terms of a smaller standard deviation. The accuracy improves nearly exponentially with $k$. However, increasing the history size to a too large value above $k = 100$ does not significantly improve the accuracy while it will lead to a slower responsiveness of the estimator as shown in the next paragraph.

In the next step we take a closer look at the trade-off between accuracy and responsiveness in dependence of the size of the history. To express accuracy, we regard the deviation from the actual value in percent. In particular, we consider how much the 97.5% and 2.5% quantiles of the estimated values based on $k$ observations differ from the actual value in percent. This is plotted as the dotted blue curves in Figure 4.27 using the left $y$-axis. As in the previous figure, it can be recognized that an increase in the history size results in more accurate estimates. The quantiles confirm the exponential dependence on $k$.

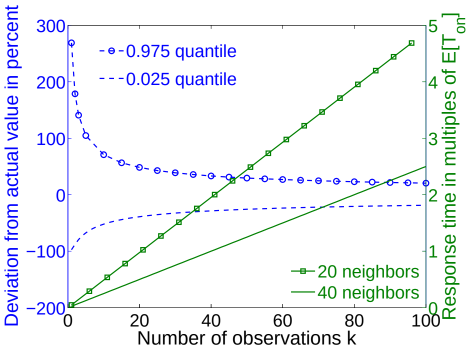


Figure 4.27: *Trade-off between accuracy and responsiveness*

An increased accuracy, however, comes at the drawback of reducing the responsiveness of the estimator. Responsiveness is defined as the time it takes to collect $k$ fresh results when there is a change in the global churn rate. It is expressed in multiples of $E[T_{on}]$ and approximated by Equation 4.40. The responsiveness increases linearly with $k$ as can be seen in the green solid curves of Figure 4.27 using the $y$-axis on the right. The slope of the curve is determined by the number of overlay neighbors. The more neighbors there are, the more results are obtained per time unit and the faster the estimator reacts to the change in the churn rate. The study shows that depending on the application requirements, a trade-off can be made between a higher accuracy and a faster responsiveness by changing the number of considered observations.

## Responsiveness of the Estimator

Responsiveness is a measure for the time it takes our estimator to react to changes of the global churn rate of the network. It mainly depends on the number of overlay contacts which share their observations, but is also influenced by the absolute churn rate itself. The higher the churn rate is, the more results can be collected within the same time period.

In order to provide a more comprehensive study of the responsiveness of the estimator and to validate our analytical approximation in Equation 4.40, we perform simulation runs with different churn rates and measure the time between two successive observations. Obviously, the smaller this inter-observation time is, the faster our method will react to changes of the churn rate. This is shown in Figure 4.28. For different churn rates of $E[T_{on}] = 300$s, 600s, and 900s, the inter-observation time is depicted as a function of the number of overlay contacts. The dashed lines are the results obtained by the approximation, cf. Equation 4.40. It can be seen that the inter-observation time decreases exponentially and that the analytical curves match well with those obtained by simulations. However, we can also recognize that a greater number than 20 neighbors is not necessarily justified due to the small improvement in responsiveness and the higher overhead in maintaining those neighbors. Note, that the responsiveness also depends on the quality of the stabilization algorithm. If a simple algorithm is used, the neighbor lists might be inaccurate, which in turn results in a loss of updates and thus a higher inter-observation time.

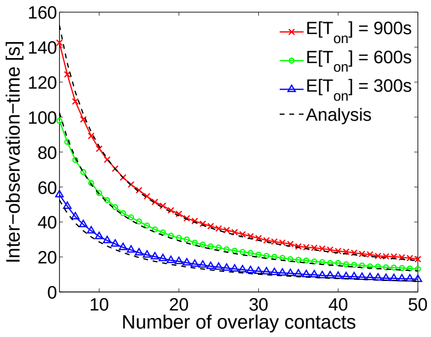


Figure 4.28: *Responsiveness to different churn rates*

To show how the inter-observation time translates into the actual response time and how the estimator behaves during these reaction phases, we simulated a network where the mean online time of all peers was globally changed from the initial value of 300s to 900s after a simulation time of 250 minutes.

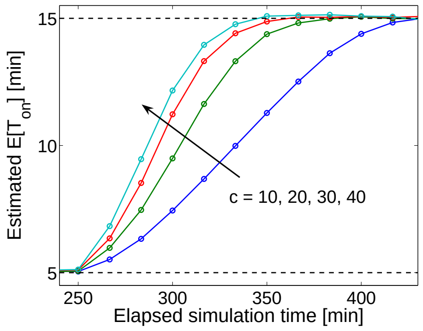


Figure 4.29: *Reaction to change in global churn rate*

In Figure 4.29 each data point shows the average of the estimated $E[T_{on}]$ values of all online peers at the same time instant. The figure visualizes that the estimates react differently to the change of the global churn rate. Again the more neighbors there are, the faster the estimator approaches the new churn rate. While the increase is nearly linear for $c = 10$, there is not much difference between the curves for $c = 20$, 30, and 40 neighbors. Thus, using a too large number of overlay contacts is not justified due to the additional overhead. Using 20 overlay neighbors, as e.g. suggested in Kademlia, is therefore a reasonable choice.

## Practicability and Implementation Aspects

In practice, the estimate of the churn rate will be used to self-adaptively tune maintenance algorithms of the p2p network, e.g. the stabilization of the overlay structure and the control of the replication of stored documents. Therefore, it would be desirable that all peers obtain equal estimate values in order to derive similar input parameters to these algorithms. Since our proposed estimation method is entirely distributed and each peer calculates its own estimate from local measurements, different peers also tend to obtain different estimation results. However, most maintenance algorithms are performed between direct overlay neighbors of the DHT.

Since these direct overlay neighbors also exchange their measured observations, their churn estimates derived from this data are expected to be highly correlated. To quantify the degree of this correlation, we took a global snapshot during the simulation and had a closer look at the estimates of 5000 consecutive peers on the Chord ring. We then investigated the correlation between these peers by applying methods from time series analysis. Figure 4.30 depicts the autocorrelation over the number of neighbors. The $x$-axis represents the different lags $x$, which in our case corresponds to the $x$-th overlay neighbor. If $x$ is positive, this corresponds to the $x$-th successor on the ring, whereas a negative value represents the $|x|$-th predecessor. The figure shows that the estimates of a peer are indeed highly correlated among neighboring peers. The curves for the different num-

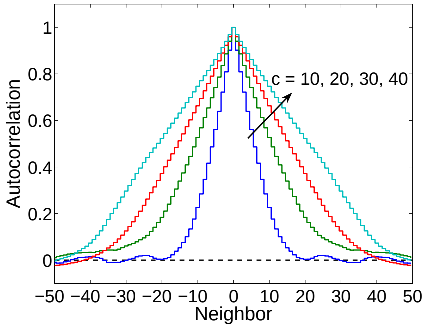


Figure 4.30: *Autocorrelation of estimates over neighboring peers*

bers $c$ of overlay neighbors among which the measurement values are exchanged show that the correlation extends to at least $c$ neighbors in both directions of the ring. Note that due to our symmetric neighbor lists a value of $c = 10$ overlay neighbors means that the peer maintains 5 neighbors in both directions of the ring.

## 4.4 Monitoring a Distributed P2P System at Runtime

In the previous chapters we discussed the possibilities of an individual peer to estimate important system parameters. While this creates a great basis for self-organizing maintenance algorithms, overlay service providers are interested in more complex information about the entire network. In particular, they do not want to lose control over their applications, but rather need to know what their system looks like right now and where problems occur at the moment. However, the decentralized nature of structured overlay networks makes it hard to find a scalable way to gather information about the running system at a central unit.

In this chapter, we therefore exploit the special features of structured p2p overlays and present a novel and scalable approach [4, 15] to create a snapshot of a running Chord-based network. On basis of the gathered information, it is, e.g. possible to take appropriate action to relieve a hotspot or to pinpoint the cause of a loss of the overlay ring structure. The overhead involved in creating the snapshot is evenly distributed to the participating peers so that each peer only has to contribute a negligible amount of bandwidth. Most importantly, the snapshot algorithm is very easy to use for a provider. It only takes one parameter to adjust the trade-off between duration of the snapshot and bandwidth needed at the central unit which collects the measurements.

## 4.4.1 A Divide and Conquer Approach

Our algorithm to create a snapshot of a running Chord system is based on a very simple two step approach. In step one, the overlay is recursively divided into subparts of a predefined size. In step two, the desired measurement is done for each of these subparts and sent back to a central collecting point ($CP$). In the following, we describe both steps in detail.

### Step 1: Divide Overlay into Subparts

The algorithm *snapshot($R_s, R_e, S_{min}, CP$)* divides a specific region of the overlay into subparts. This function is called at an arbitrary peer $p$ with $id_p$. The peer then tries to divide the region from $R_s = id_p$ to $R_e$ into contiguous subparts using its fingers. The exact procedure is illustrated in Figure 4.31. In this example peer $p$ has four fingers $f_1$ to $f_4$. It sends a request to the finger closest to $R_e$ within $[R_s; R_e]$. At first, finger $f_4$ is disregarded since it does not fall into the region between $R_s$ and $R_e$ (cf. a). This makes $f_3$ the closest finger to $R_e$ in our example. If this finger does not respond to the request, as illustrated by the bolt (cf. b), it is removed from the peer's finger list and the peer tries to contact the

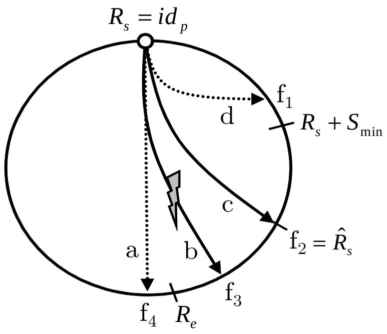


Figure 4.31: *Visualization of the algorithm.*

next closest finger $f_2$ (cf. c). If this finger acknowledges the request, peer $p$ recursively tries to divide the region from $R_s = id_p$ to $\widehat{R}_e = id_{f_2} - 1$ into contiguous subparts. Finger $f_2$ partitions the region from $\widehat{R}_s = id_{f_2}$ to $R_e$ accordingly.

As soon as a peer does not know any more fingers in the region between the current $R_s$ and the current $R_e$, the recursion is stopped. The peer changes into step two of the algorithm and starts a measurement of this specific region. In this context, the parameter $S_{min}$ can be used to determine the minimum size of the regions, which will be measured in step two. Taking into account $S_{min}$, a peer will already start the measurement if it does not know any more fingers in the region between the current $R_s + S_{min}$ and the current $R_e$. In this case finger $f_1$ would be disregarded, as illustrated by the dotted line (cf. d in Figure 4.31), since it points into the minimum measurement region. Parameter $S_{min}$ is designed to adjust the trade off between the duration of the snapshot and the bandwidth needed at the collecting point. The larger the regions, the longer the measurement will take. The smaller the regions, the more results are sent back to the CP.

A detailed technical description of the procedure is given in Algorithm 1. Peer $p$ will contact the closest finger to $R_e$ until it does not know any more fingers in between $R_s + S_{min}$ and $R_e$. If so, it changes into step two and starts a measurement of this region calling countingtoken($id_p$, $R_e$, $S_{min}$, $CP$, $result$) at the local peer.

**Algorithm 1**
The snapshot algorithm (first call $R_s = id_p$)

```
snapshot($R_s$, $R_e$, $S_{min}$, $CP$)
send acknowledgment to the sender of the request
$id_{fm} = max(\{id_f | id_f \in \text{fingerlist} \wedge id_f < R_e\})$
while $id_{fm} > R_s + S_{min}$ do
    send snapshot($id_{fm}$, $R_e$, $S_{min}$, $CP$) request to peer $id_{fm}$
    if acknowledgment from $id_{fm}$ then
        call snapshot($id_p$, $id_{fm} - 1$, $S_{min}$, $CP$) at local peer
        return //exit the function
    else
        remove $id_{fm}$ from fingerlist
        $id_{fm} = max(\{id_f | id_f \in \text{fingerlist} \wedge id_f < R_e\})$
    end if
end while
$\widehat{S} = \frac{R_e - R_s}{\lceil \frac{R_e - R_s}{S_{min}} \rceil}$ //explanation see step two
call countingtoken($id_p$, $R_e$, $S_{min}$, $CP$, $\emptyset$) at local peer
```

### Step 2: Measure a Specific Subpart

The basic idea behind the measurement of a specific subpart from $R_s$ to $R_e$ is very simple. The first peer creates a token, adds its local statistics, and passes the token to its immediate successor. The successor proceeds recursively until the first peer with an $id > R_e$ is reached. This peer sends the token back to the collecting point, whose IP is given in the parameter CP.

Ideally, each of the regions measured in step two would be of size $S_{min}$ as specified by the user. The problem, however, is that the region from $R_s$ to $R_e$ is slightly larger than $S_{min}$ according to step one of the algorithm. In fact, if the responsible peer did not know enough fingers, the region might even be significantly larger than $S_{min}$. The solution to this problem is to introduce checkpoints

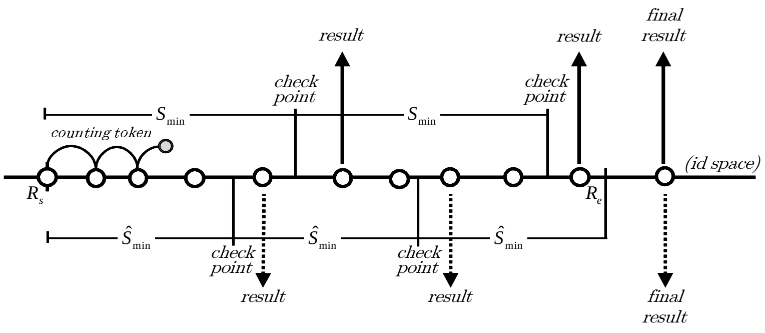


Figure 4.32: *Results are sent back to the $CP$ after each checkpoint*

with a distance of $S_{min}$ in the corresponding region. Results are sent to the $CP$ every time the token passes a checkpoint instead of sending only one answer at the end of the region. This is illustrated in the upper part of Figure 4.32. The counting token is started at $R_s$. The first peer behind each checkpoint sends a $result$ back to the $CP$ as illustrated by the large solid arrows. The final $result$ is still sent by the first peer with $id > R_e$.

A drawback of this solution is that the checkpoints might not be equally distributed in the region. In particular, the last two checkpoints might be very close to each other. We therefore recalculate the positions of the checkpoints according to the following equation:

$$\widehat{S}_{min} = \frac{R_e - R_s}{\left\lceil \frac{R_e - R_s}{S_{min}} \right\rceil}. \tag{4.42}$$

The new checkpoints can be seen in the lower part of Figure 4.32. The number of checkpoints remains the same, while their positions are moved in such a way, that the results are now sent at equal distance.

As can be seen at the end of Algorithm 1, the recalculation of $S_{min}$ is already

**Algorithm 2**
The countingtoken algorithm (first call $R_s = id_p$)

```
countingtoken($R_s$, $R_e$, $S_{min}$, $CP$, $result$)
send acknowledgment to the sender of the request
**if** $R_s \leq id_p \leq R_e$ **then**
   **if** $id_p > R_s + S_{min}$ **then**
      send $result$ to $CP$
      $result = 0$
      $R_s = R_s + S_{min}$
   **end if**
   add local measurement to $result$
   $id_s$ = $id$ of direct successor
   **while** true **do**
      send countingtoken($R_s$, $R_e$, $S_{min}$, $CP$, $result$) request to direct succes-
      sor $id_s$
      **if** acknowledgment **then**
         break
      **else**
         remove $id_s$ from successor list
         $id_s$ = $id$ of new direct successor
      **end if**
   **end while**
**else**
   send $result$ to $CP$
**end if**
```

done in the first step, just before the counting token is started. A detailed description of the counting token mechanism is given in Algorithm 2. If a peer $p$ receives a counting token it makes sure that its identifier is still within the measured region, i.e. $R_s \leq id_p \leq R_e$ . If not, it sends a $result$ back to the $CP$ and stops the token. Otherwise it adds its local measurement to the token and tries to pass the token to its immediate successor. If it is the first peer behind one of the checkpoints, it sends an intermediate result back to the $CP$ and resets the token.

## Collect Statistics

Generally speaking, there are two different kinds of statistics, which can be collected using the counting tokens. Either a simple mean value or a more detailed histogram. In the first case the counting token memorizes two variables, $V_a$ for the accumulated value and $V_n$ for the number of values. Each peer receiving the counting token adds its measured value to $V_a$ and increases $V_n$ by one. The sample mean can then be calculated at the $CP$ as $\frac{\sum V_a}{\sum V_n}$. In case of a histogram, the counting token maintains a specific number of bins and their corresponding limits. Each peer simply increases the bin matching its measured value by one. If the measured value is outside the limits of the bins it simply increases the first or the last bin respectively.

There are numerous things that can be measured using the above mentioned methods. Table 4.1 summarizes some exemplary statistics and the kind of information which can be gained from them. The most obvious application is to count the number of hops for each counting token. On the one hand, this is a direct measure for the size of the overlay network. On the other hand, it also shows the distribution of the identifiers in the identifier space. To gain information about the

Table 4.1: *Possible statistics gathered during a snapshot*

| **Statistic** | **Information gained** |
|---|---|
| Number of hops per token | Size of the network<br>Distribution of the identifiers |
| Mean search delay | Performance of the algorithm |
| Sender $==$ predecessor? | Overlay stability |
| Number of timeouts per token | Churn rate |
| Number of resources per peer | Fairness of the algorithm |
| Number of searches answered | User behavior |
| Bandwidth used per time unit | Maintenance overhead |
| Missing resources | Data integrity |

performance of the Chord algorithm, the mean search delay or a histogram for the search time distribution can be calculated and compared to expected values.

Furthermore, Chord's stability can only be guaranteed as long as the successor and predecessor pointers of the individual peers match each other correspondingly. This invariant can be checked by counting the percentage of hops, where the sender of the counting token did not match the predecessor of the receiving peer. Additionally, the number of timeouts per token can be used to measure the current churn rate in the overlay network. The more churn there is, the more timeouts are going to occur due to outdated successor pointers. Similarly, the number of resources stored at each peer is a sign of the fairness of the Chord algorithm. The number of searches answered at each peer can likewise be used to get an idea of the search behavior of the end users. Finally, a peer can keep track of the number of missing resources to verify the integrity of the stored data. This can, e.g., be done counting the number of search requests which could not be answered by the peer.

All of the above statistics can be collected periodically to survey the time dependent status of the overlay. Note, that it is also possible to monitor a specific part of the overlay network by setting $R_s$ and $R_e$ accordingly. This can, e.g., be helpful if there are problems in a certain region of the overlay network and the operator needs to verify that his countermeasures have been successful.

### 4.4.2 Analytical Evaluation of the Algorithm

To analyze the performance of our algorithm, we have a closer look at different performance measures. At first, we will regard the overhead produced during the generation of a snapshot. Then, we will estimate the duration of a snapshot as well as the temporal distribution of the token arrival times at the $CP$. Finally, we will discuss the probability to lose tokens while creating a snapshot of the running system.

## Required Bandwidth at the Monitor Station

The snapshot algorithm takes only one single parameter, $S_{min}$. It directly determines the number of areas $N_r$ into which the Chord ring is divided during a snapshot:

$$N_r = \left\lceil \frac{S_{id}}{S_{min}} \right\rceil \ . \tag{4.43}$$

Independent of the current size of the overlay network at least one result per region is sent back to the $CP$. Step one of the snapshot algorithm yields the following bounds for $N_c$, the number of counting tokens sent to the $CP$:

$$2 \cdot \left\lceil \frac{S_{id}}{S_{min}} \right\rceil \geq N_c \geq \left\lceil \frac{S_{id}}{S_{min}} \right\rceil \ . \tag{4.44}$$

The inequation can be explained as follows: According to the second step of the algorithm, a counting token sends an intermediate result every $\widehat{S}_{min}$ and an additional result at the end of the region. Obviously, this way, at least on result is sent per region. In the worst case, however, the region is slightly larger than the original $S_{min}$ in which case an intermediate checkpoint is created and the number of tokens is thus doubled.

As can be seen in Equation 4.43, $S_{min}$ can be used to adjust the trade-off between the duration of a snapshot and the number of tokens, which arrive at the $CP$. The larger $S_{min}$, the more hops per counting token are needed and the longer the snapshot will take. The smaller $S_{min}$, the less hops per counting token are needed but the more tokens arrive at the $CP$ in total.

## Duration of a Snapshot

The duration of a snapshot does not only depend on $S_{min}$, but also on the current churn rate and the exact implementation of the Chord algorithm. To be able to calculate an estimate of the duration of a snapshot, we assume a scenario without any peers joining or leaving the network. In this case, the duration of step one, the signaling step, can be estimated as follows. The last region, in which a counting token will be started is the one directly following the initiating peer. This is due to the fact, that the initiating peer will start its counting token no sooner than it divided the ring into separate regions. Before it starts the counting token, it contacts its fingers until the first finger is closer to itself than $S_{min}$. The initiating peer has at most $\log_2(n)$ fingers and each of the fingers sends an acknowledgment, before the peer can go on with the algorithm. If $T_O$ is the random variable describing the duration of one overlay hop, then the duration of step one of the algorithm is at most

$$D_{step1} = 2 \cdot \log_2(n) \cdot E[T_O] \ . \tag{4.45}$$

In the worst the initiating peer does not have any fingers and directly sends the counting token in step two. This would take $n \cdot E[T_O]$, but is very unlikely to happen. An easy solution to this problem would be to pass the responsibility of dividing the ring to the direct successor in case the counting token region becomes too large. In general, if there are $n$ peers in the overlay, there are roughly

$$P_r = \frac{n \cdot S_{min}}{S_{id}} \tag{4.46}$$

peers per region. Furthermore, in the worst case $S_{min}$ is slightly larger than a power of two and the region covered by a counting token may become almost twice as large as $S_{min}$. Therefore a good estimate for the duration of the counting step of the algorithm is:

$$D_{step2} = 2 \cdot P_r \cdot E[T_O]. \tag{4.47}$$

This results in the following total duration of a snapshot:

$$D = \left(\log_2(n) + \frac{n \cdot S_{min}}{S_{id}}\right) \cdot 2 \cdot E[T_O]. \tag{4.48}$$

## Token Arrival Time Distribution

To get a rough estimate for the distribution of the arrival times of the counting tokens at the $CP$, we consider the special case where the size of the overlay $n = 2^g$ is a power of two and $S_{min}$ is such that $N_r = 2^h$ with $h < g$. Furthermore, we assume that the individual peers are placed at equal distances on the ring.

In this case $h = \log_2(N_r)$ is the number of overlay hops it takes until the first counting token is started during a snapshot [4]. Similarly, it takes $2 \cdot h$ hops until the last counting token is started according to our assumptions. The probability $p_i$ that a counting token is started after exactly $i$ hops for $i = h, h+1, ..., 2 \cdot h$ can be calculated as:

$$p_i = \frac{\binom{h}{i-h}}{\sum_{x=h}^{2 \cdot h} \binom{h}{x-h}} \; . \tag{4.49}$$

The signaling step thus takes $i$ overlay hops with probability $p_i$ and is followed by $P_r$ hops of the counting token and the final hop to report the result back to the $CP$. To validate this analytical result, we simulated a Chord ring of size $n = 2^{15}$ with $S_{min} = 2^9$ assuming exponentially distributed overlay hops with a mean of 80 ms. Figure 4.33 shows the numerically approximated PDF of the token arrival times at the $CP$. Obviously, the simulation and the analysis match very well and the binomial distribution of the duration of step one can be recognized. So far, in the analysis each peer has a finger at an exact distance of $S_{min}$. In reality, however, the finger would sit at a slightly different position, which again would result in an additional checkpoint at the middle of the region. If we consider this in our analysis we obtain an intermediate result in the middle of the measurement region (c.f. checkpoints in the figure). The first rise of the PDF represents the intermediate results sent back to the $CP$ at the checkpoint. The second rise still represents the regular results at the end of the region.

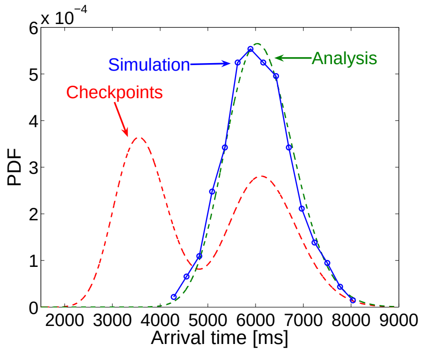


Figure 4.33: *Probability density function of the token arrival time*

## Lost Tokens

As in all token based algorithms, there is a certain probability of losing a token or a signaling message. In our case, this is especially true during high churn phases. However, the loss of a token only results in a loss of the measurements of the corresponding region. Additionally, the probability to lose a token is very small, since there are only two scenarios which result in the loss of a token. First, if a peer goes offline while it still waits for the timeout of a signaling messages. Second, if a peer goes offline while it still waits for the timeout of a counting token message. Again both scenarios only involve the loss of the measurements of the corresponding region of the ring. In both cases the probability for the loss of the region is

$$p_{loss} = P(A_r \leq timeout) \tag{4.50}$$

where $A_r$ is the random variable describing the forward recurrence time of the session duration of a peer and $timeout$ is the value of a timeout in the overlay network.

### 4.4.3 Interpretation of the Collected Statistics

The results in this section were obtained using our ANSI-C simulator, which incorporates a detailed yet slightly modified Chord implementation. A good description of the general simulation model can be found in [21]. In this work an overlay hop is modeled using an exponentially distributed random variable with a mean of 80ms. The results considering churn are generated using peers which stay online and offline for an exponentially distributed period of time with a mean as indicated in the corresponding description of the figures. The parameter

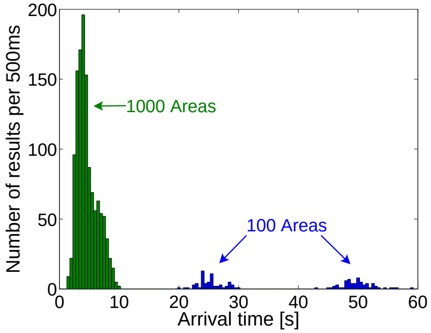


Figure 4.34: *Arrival times of the results.*

$N_r = \left\lceil \frac{S_{id}}{S_{min}} \right\rceil$ influences the duration of the snapshot as well as the number of results arriving at the central collecting point. Figure 4.34 shows the distribution of the arrival times of the results in an overlay of 40000 peers using $N_r = 1000$ and $N_r = 100$ areas in times of no churn. Obviously, the more areas the overlay is divided into, the faster the snapshot is completed. While the snapshot using 1000 areas was finished after about ten seconds, the snapshot with 100 areas took about one minute. In exchange the latter snapshot produces significantly smaller bandwidth spikes at the CP. The two elevations of the second histogram correspond to the intermediate results (first elevation) and the final results at the end of the measured subpart (second elevation).

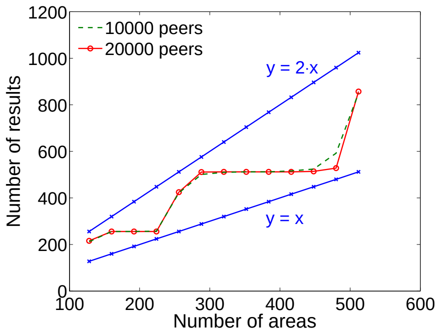


Figure 4.35: *Number of results received at the $CP$.*

A more detailed analysis of the influence of $N_r$ can be found in Figure 4.35, which shows the number of results received at the $CP$ in dependence of $N_r$. According to Equation 4.44, the number of counting tokens sent to the $CP$, is limited by $2 \cdot N_r > N_c \geq N_r$. The straight lines in the figure show the corresponding limits. The solid and dotted curves represent the results obtained for 20000 and 10000 peers, respectively. The number of results sent to the $CP$ is within the calculated limits and independent of the overlay size. The curves roughly resemble the shape of a staircase, whereas the steps are located at powers of two. This is due to the fact that the closer $N_r$ gets to a power of two, the smaller is the region covered by the average counting token and the more results arrive at the $CP$.

The distribution of the arrival times of the results is also influenced by the current size of the network. The larger the network, the more peers are within one region. However, the more peers are within one region, the more hops each counting token has to make, before it can send its results back to the CP. Figure 4.36 shows the token arrival time distribution for three different overlay sizes of 10000, 20000, and 40000 peers, respectively. We did not generate any churn in this scenario and set $N_r = 512$ areas. As expected, the larger the overlay network, the longer the snapshot is going to take. However, the curves are not only shifted to the right, but also differ in shape. This can again be explained by the increasing number of hops per counting token.

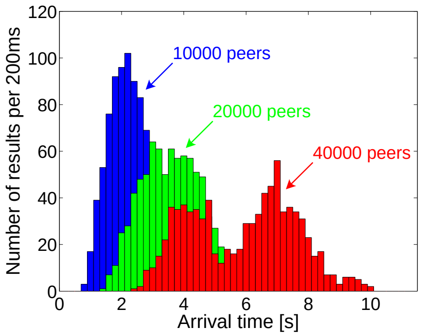


Figure 4.36: *Arrival times of the results at the $CP$.*

As mentioned above, the average counting token sends two results back to the CP, whereas the checkpoints are equally spaced. Thus, the final result takes twice as many hops as the intermediate result. In a network of 10000 peers there are approximately 20 peers in each of the 512 regions. The intermediate results are therefore sent after about 10 hops, the final results after about 20 hops, respectively. The two corresponding elevations in the histogram overlap in such a way, that they build a single elevation. In a network of 40000 peers, however, there are approximately 78 peers in each of the 512 regions. The intermediate results are therefore sent after about 39 hops, the final results after about 78 hops, respectively. The difference between these two numbers is large enough to account for the two elevations of the histogram in the foreground of Figure 4.36.

To measure the influence of churn on the stability of the overlay network, we regard the number of timeouts which occur during a snapshot as well as the frequency at which the predecessor pointer of a peer's successor does not match the peer itself. Figure 4.37 plots the relative frequency of timeouts and pointer fail-

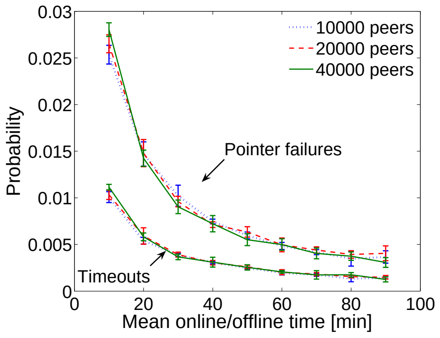


Figure 4.37: *Relative frequency of timeouts and pointer failures.*

ures against the mean online/offline time of a peer. The smaller the online/offline time of a peer, the more churn is in the system.

The results represent the mean of several simulation runs, whereas the error bars show the 95 percent confidence intervals. The relatively small percentage of both timeouts and failures is to some extent implementation specific. More interesting, however, is the exponential rise of the curves under higher churn rates. The shape of both curves is independent of the size of the overlay and only affected by the current churn rate. The curve can therefore be used to map the frequency of timeouts or failures measured in a running system to a specific churn rate.

# 5 Conclusion

The size and complexity of current computer networks call for new functionality which was not a part of their initial design. Structured overlay networks are a means to provide the necessary corrections on top of the actual network. Such overlay mechanisms are highly flexible and designed to scale with the system size. However, little is known about the performance of structured overlay networks in a productive environment.

In this monograph, we analyzed such systems in more detail. In particular, we computed the entire distribution function of the search delay as seen from a user issuing a search query. We provided numerical results to illustrate the dependence of the search duration on the variation of the network transmission delay. In this context, the analysis provides an insight into the quantiles of the search delay which can be used for dimensioning purposes. Both the mean search duration as well as the search delay quantiles were shown to scale logarithmically with the size of the system. Thus, we observed that the the real issue of structured overlay networks is the dynamic behavior of the participating peers. We therefore analyzed the stability of ring-based overlay networks in more detail. In contrast to previous work we showed that the probability to lose the overlay structure is not negligible in all scenarios. For realistic use cases we derived an equation to calculate failure probabilities in dependence of the average online time of a peer and showed that in such scenarios stability can be guaranteed with very high probability. From this we concluded that maintenance overhead should be adapted dynamically. The more movement there is in the overlay, the more maintenance overhead should be applied.

To understand the performance of the originally proposed structured overlay networks in more detail, we developed a discrete event simulator and pinpointed the weak points in their algorithms. Using Kademlia as an example, we proposed different modifications to the original algorithm to enhance the performance, the redundancy, and the robustness of structured overlay networks. In particular, we developed improved mechanisms to maintain the structure of the overlay as well as the redundancy of stored documents. Our new implementation requires less maintenance traffic while offering an improved overall performance.

In great parts our performance analysis as well as many modifications proposed in literature rely on knowledge of system parameters like the current overlay size or the distribution of the peer session duration. Additionally, an operator of a deployed overlay network needs to be able to continuously monitor and evaluate these parameters. Such system characteristics, however, are inherently unknown in a heavily distributed overlay network. We therefore introduced and evaluated different methods to capture system parameters at runtime. In particular, we developed a passive estimator for the current size of the system which exploits the special features of structured overlay networks. The resulting estimates are in the right order of magnitude which is sufficient in practical scenarios where usually the logarithm of the system size is needed. Additionally, we showed how to calculate confidence intervals whose endpoints can be used as independent estimates depending on whether it is more critical to overestimate or to underestimate the actual value.

To be able to assess and to quantify the user behavior, we provided a definition of churn, the continuous process of peers joining and leaving the overlay network. From this we derived an algorithm to estimate the current churn in the system based on the changes a peer observes in its list of overlay neighbors. Both analytical and simulation results showed that the estimator is able to capture the current churn in the system. The accuracy, the required overhead, and the responsiveness to changes can be adjusted by the number of observations considered in the estimation process and by the number of overlay neighbors which share the results. In parameter studies, we investigated the corresponding trade-offs and deduced values which are suitable for practical purposes.

For an operator such passive estimates are not sufficient to understand the system as a whole. He rather depends on the possibility to perform active measurements of all values which he is interested in. We therefore also introduced an entirely distributed and scalable algorithm to monitor a structured overlay network at runtime. Thereby, the load generated to create the snapshot of the system is evenly distributed among the peers of the overlay. The algorithm itself is easy to configure as it only requires one single parameter to determine the trade-off between the duration of the snapshot and the bandwidth required at the central collection point. We performed a mathematical analysis of the basic algorithm and provided a simulative study considering realistic user behavior. The algorithm is resistant to dynamic user behavior and provides a very accurate picture of the system.

In the course of this monograph we showed that structured overlay networks are a powerful tool to overcome the problems which are inherent to todays computer networks. They offer an easy way to add new functionality to a network which itself would be hard to modify. Thereby, the results of this work can be seen as a first step towards a self-organizing overlay system. Using our estimation methods a peer is able to automatically derive the input necessary to evaluate the current system performance. Based on the results the peer can then dynamically adapt important system parameters like the number of overlay connections it maintains to other peers. Based on our findings we come to the conclusion that overlay networks are by far not limited to best effort file-sharing or content distribution services. In fact they are already successfully being used in applications like distributed VoIP telephone services or distributed data storages. Another very promising approach is distributed network management, where structured overlay networks can be set up to connect, coordinate, and manage distributed measurement points as we have shown in [18, 25]. Those networks can then be used to pinpoint the root causes of network problems [16] or to perform distributed end-to-end measurements [9].

# Bibliography of the Author

**— Book Chapters —**

[1] A. Binzenhöfer and T. Hoßfeld, “Warum Panini Fußballalben auch Informatikern Spaß machen,” in *Fußball eine Wissenschaft für sich* (H.-G. Weigand, ed.), pp. 181–192, Verlag Königshausen & Neumann, April 2006.

**— Journals —**

[2] T. Hoßfeld and A. Binzenhöfer, “Analysis of Skype VoIP Traffic in UMTS: End-to-End QoS and QoE Measurements,” *Computer Networks*, vol. 51, no. 19, 2007.

[3] A. Binzenhöfer, H. Schnabel, and P. Tran-Gia, “Methods for Performance Improvement of Kademlia-based Overlay Networks,” *it - Information Technology (Methods and Applications of Informatics and Information Technology)*, vol. 49, pp. 280–288, May 2007.

[4] A. Binzenhöfer, G. Kunzmann, and R. Henjes, “Design and Analysis of a Scalable Algorithm to Monitor Chord-based P2P Systems at Runtime,” *Concurrency and Computation: Practice and Experience - Special Issue on HotP2P 06*, August 2007.

**— Conference Papers —**

[5] D. Schlosser, A. Binzenhöfer, and B. Staehle, "Performance Comparison of Windows-based Thin-Client Architectures," in *Australasian Telecommunication Networks and Applications Conference 2007*, (Christchurch, New Zealand), December 2007.

[6] B. Emmert, A. Binzenhöfer, D. Schlosser, and M. Weiß, "Source Traffic Characterization for Thin-Client Based Office Applications," in *13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, (University of Twente, the Netherlands), July 2007.

[7] A. Binzenhöfer and K. Leibnitz, "Estimating Churn in Structured P2P Networks," in *20th International Teletraffic Congress (ITC20)*, (Ottawa, Canada), June 2007.

[8] K. Eger, T. Hoßfeld, A. Binzenhöfer, and G. Kunzmann, "Efficient Simulation of Large-Scale P2P Networks: Packet-level vs. Flow-level Simulations," in *2nd Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks (UPGRADE-CN'07) in conjunction with the 16th IEEE HPDC*, (Monterey Bay, California, USA), May 2007.

[9] A. Binzenhöfer, D. Schlosser, K. Tutschku, and M. Fiedler, "An Autonomic Approach to Verify End-to-End Communication Quality," in *10th IFIP-IEEE International Symposium on Integrated Network Management (IM 2007)*, (Munich, Germany), May 2007.

[10] I. Norros, V. Pehkonen, H. Reittu, A. Binzenhöfer, and K. Tutschku, "Relying on Randomness - PlanetLab Experiments with Distributed File-Sharing Protocols," in *3rd EURO-NGI Conference on Next Generation Internet Networks (NGI 2007)*, (Trondheim, Norway), May 2007.

[11] A. Binzenhöfer, T. Hoßfeld, G. Kunzmann, and K. Eger, “Efficient Simulation of Large-Scale P2P Networks: Compact Data Structures,” in *Workshop on Modeling, Simulation, and Optimization of Peer-to-peer environments (MSOP2P) in conjunction with Euromicro (PDP 2007)*, (Naples, Italy), February 2007.

[12] G. Kunzmann, R. Nagel, T. Hoßfeld, A. Binzenhöfer, and K. Eger, “Efficient Simulation of Large-Scale P2P Networks: Modeling Network Transmission Times,” in *Workshop on Modeling, Simulation, and Optimization of Peer-to-peer environments (MSOP2P) in conjunction with Euromicro (PDP 2007)*, (Naples, Italy), February 2007.

[13] A. Binzenhöfer and H. Schnabel, “Improving the Performance and Robustness of Kademlia-based Overlay Networks,” in *Kommunikation in Verteilten Systemen (KiVS) 2007,* (Bern, Switzerland), February 2007.

[14] G. Kunzmann and A. Binzenhöfer, “Autonomically Improving the Security and Robustness of Structured P2P Overlays,” in *International Conference on Systems and Networks Communications. ICSNC 2006,* (Tahiti, French Polynesia), November 2006.

[15] A. Binzenhöfer, G. Kunzmann, and R. Henjes, “A Scalable Algorithm to Monitor Chord-based P2P Systems at Runtime,” in *Third International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P) in conjunction with the IEEE International Parallel & Distributed Processing Symposium ( IPDPS 2006),* (Rhodes Island, Greece), April 2006.

[16] B. Emmert and A. Binzenhöfer, “Efficient Link Failure Detection and Localization using P2P-Overlay Networks,” in *The First International Workshop on Dependable and Sustainable Peer-to-Peer Systems in conjunction with The First International Conference on Availability, Reliability and Security*, (Vienna University of Technology, Austria), April 2006.

[17] T. Hoßfeld, A. Binzenhöfer, M. Fiedler, and K. Tutschku, "Measurement and Analysis of Skype VoIP Traffic in 3G UMTS Systems," in *4th International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe 2006)*, (Salzburg, Austria), February 2006.

[18] A. Binzenhöfer, K. Tutschku, B. auf dem Graben, M. Fiedler, and P. Arlos, "A P2P-based Framework for Distributed Network Management," in *New Trends in Network Architectures and Services, LNCS 3883*, (Loveno di Menaggio, Como, Italy), May 2006.

[19] M. Fiedler, K. Tutschku, S. Chevul, L. Isaksson, and A. Binzenhöfer, "The Throughput Utility Function: Assessing Network Impact on Mobile Services," in *New Trends in Network Architectures and Services, LNCS 3883*, (Loveno di Menaggio, Como, Italy), May 2006.

[20] A. Binzenhöfer, D. Staehle, and R. Henjes, "On the Stability of Chord-based P2P Systems," in *GLOBECOM 2005*, (St. Louis, MO, USA), November 2005.

[21] G. Kunzmann, A. Binzenhöfer, and R. Henjes, "Analyzing and Modifying Chord's Stabilization Algorithm to Handle High Churn Rates," in *MICC & ICON 2005*, (Kuala Lumpur, Malaysia), November 2005.

[22] A. Binzenhöfer, D. Staehle, and R. Henjes, "On the Fly Estimation of the Peer Population in a Chord-based P2P System," in *19th International Teletraffic Congress (ITC19)*, (Beijing, China), September 2005.

[23] M. Fiedler, S. Chevul, O. Radtke, K. Tutschku, and A. Binzenhöfer, "The Network Utility Function: A Practicable Concept for Assessing Network Impact on Distributed Services," in *19th International Teletraffic Congress (ITC19)*, (Beijing, China), September 2005.

[24] P. Tran-Gia and A. Binzenhöfer, "On the Stochastic Scalability of Information Sharing Platforms," in *2005 Tyrrhenian International Workshop on Digital Communications*, (Sorrento, Italy), July 2005.

[25] A. Binzenhöfer, K. Tutschku, and B. auf dem Graben, "DNA: A p2p-based Framework for Distributed Network Management," in *Peer-to-Peer-Systeme und -Anwendungen, GI/ITG-Workshop in Kooperation mit KiVS 2005*, (Kaiserslautern, Germany), March 2005.

[26] A. Binzenhöfer and P. Tran-Gia, "Delay Analysis of a Chord-based Peer-to-Peer File-Sharing System," in *ATNAC 2004*, (Sydney, Australia), December 2004.

[27] A. Binzenhöfer and K. Tutschku, "Auswirkungen der Virtualisierung auf Transparenz und Fehlerdiagnose in lokalen Netzen," in *15. GI - Fachtagung der Fachgruppe BIK*, (Jülich, Germany), November 2003.

[28] S. Köhler and A. Binzenhöfer, "MPLS Traffic Engineering in OSPF Networks - A combined Approach," in *18th International Teletraffic Congress (ITC18)*, (Berlin, Germany), September 2003.

## General References

[29] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," in *19th ACM Symposium on Operating Systems Principles (SOSP '03)*, (Bolton Landing, NY, USA), October 2003.

[30] N. Christin and J. Chuang, "A Cost-based Analysis of Overlay Routing Geometries," in *IEEE INFOCOM'05*, (Miami, FL, USA), March 2005.

[31] Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper, "Performance of Full Text Search in Structured and Unstructured Peer-to-Peer Systems," in *25th Conference on Computer Communications (IEEE INFOCOM 2006)*, (Barcelona, Spain), April 2006.

[32] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-scale Persistent Storage," *SIGPLAN Not.*, vol. 35, no. 11, pp. 190–201, 2000.

[33] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures," in *Proceedings of the 2ndt USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, (Boston, MA, USA), May 2005.

[34] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management," in *1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, (San Francisco, CA, USA), March 2004.

[35] "The Skype Application." URL: http://www.skype.com/.

[36] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *SIGCOMM '04*, (Portland, USA), August 2004.

[37] C. Blake and R. Rodrigues, "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two," in *9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, (Lihue, Hawaii), May 2003.

[38] K. Leibnitz, T. Hoßfeld, N. Wakamiya, and M. Murata, "Peer-to-Peer vs. Client/Server: Reliability and Efficiency of a Content Distribution Service," in *20th International Teletraffic Congress (ITC20)*, (Ottawa, Canada), June 2007.

[39] "The Gnutella Website." URL: http://www.gnutella.com.

[40] K. Aberer and M. Hauswirth, "An Overview on Peer-to-Peer Information Systems," in *Workshop on Distributed Data and Structures (WDAS)*, (Paris, France), March 2002.

[41] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A Distributed Anonymous Information Storage and Retrieval System,” in *International Workshop on Designing Privacy Enhancing Technologies*, (Berkeley, CA, USA), January 2001.

[42] “The Freenet Project.” URL: http://freenetproject.org/.

[43] O. Sandberg, “Distributed Routing in Small-World Networks,” in *8th Workshop on Algorithm Engineering and Experiments (ALENEX06)*, (Miami, FL, USA), January 2006.

[44] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, “OpenDHT: A Public DHT Service and its Uses,” *ACM SIGCOMM Computer Communications Review*, vol. 35, pp. 73–84, October 2005.

[45] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” in *ACM SIGCOMM 2001*, (San Diego, CA, USA), August 2001.

[46] V. Mesaros, B. Carton, and P. Van Roy, “S-Chord: Using Symmetry to Improve Lookup Efficiency in Chord,” in *International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA’03*, (Las Vegas, NV, USA), June 2003.

[47] S. Ramabhadran and J. Pasquale, “Analysis of Long-Running Replicated Systems,” in *25th Annual Conference on Computer Communications (INFOCOM)*, (Barcelona, Spain), April 2006.

[48] A. Datta and K. Aberer, “Internet-Scale Storage Systems under Churn - A Study of the Steady-State using Markov Models,” in *6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006)*, (Cambridge, UK), September 2006.

[49] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput," in *1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, (San Francisco, CA, USA), March 2004.

[50] G. Kunzmann, "Iterative or Recursive Routing? Hybrid!," in *KIVS, Kurzbeiträge und Workshop der 14. GI/ITG- Fachtagung in Kaiserslautern*, vol. P-61 of *Lecture Notes in Informatics (LNI)*, March 2005.

[51] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, (MIT Faculty Club, Cambridge, MA, USA), March 2002.

[52] "The Azureus Client." URL: http://azureus.sourceforge.net/.

[53] M. Steiner, E. W. Biersack, and T. Ennajjary, "Actively Monitoring Peers in KAD," in *6th International Workshop on Peer-to-Peer Systems, IPTPS07*, (Bellevue, USA), February 2007.

[54] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient Replica Maintenance for Distributed Storage Systems," in *3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, (San Jose, CA, USA), May 2006.

[55] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, (Heidelberg, Germany), November 2001.

[56] P. Druschel and A. Rowstron, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," in *Eighth Workshop on Hot Topics in Operating Systems, HOTOS*, (Elmenau, Germany), May 2001.

[57] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, “Scribe: A Large-Scale and Decentralized Application-level Multicast Infrastructure,” *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, pp. 1489–1499, October 2002.

[58] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network,” in *ACM SIGCOMM 2001*, (San Diego, CA, USA), August 2001.

[59] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, “Topologically-Aware Overlay Construction and Server Selection,” in *INFOCOM*, (New York, NY, USA), June 2002.

[60] S. Oechsner, T. Hoßfeld, K. Tutschku, and F.-U. Andersen, “Supporting Vertical Handover by a Self-Organizing Multi-Dimensional P2P Overlay,” in *IEEE 63rd Vehicular Technology Conference*, (Melbourne, Australia), May 2006.

[61] “Napster.” URL: http://www.napster.com.

[62] O. Heckmann and A. Bock, “The eDonkey 2000 Protocol,” Tech. Rep. KOM-TR-08-2002, Multimedia Communications Lab, Darmstadt University of Technology, December 2002.

[63] T. Hoßfeld, K. Leibnitz, R. Pries, K. Tutschku, P. Tran-Gia, and K. Pawlikowski, “Information Diffusion in eDonkey Filesharing Networks,” in *ATNAC*, (Sydney, Australia), December 2004.

[64] “The Emule Project.” URL: http://www.emule-project.net/.

[65] R. Brunner, “A Performance Evaluation of the Kad-Protocol,” Master’s thesis, University of Mannheim, Germany and Institut Eurecom, Sophia-Antipolis, France, November 2006.

[66] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, (Berkeley, CA, USA), June 2003.

[67] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *ACM/SIGCOMM Internet Measurement Conference (IMC-05)*, (Berkeley, CA, USA), October 2005.

[68] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P Filesharing System: Measurements and Analysis," in *4th International Workshop on Peer-to-Peer Systems (IPTPS 05)*, (Ithaca, NY, USA), February 2005.

[69] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive View of a Live Network Coding P2P System," in *6th ACM SIGCOMM on Internet measurement (IMC06)*, (Rio de Janeriro, Brazil), October 2006.

[70] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *IPTPS 2006*, (Santa Barbara, CA, USA), February 2006.

[71] C. Fragouli, J.-Y. L. Boudec, and J. Widmer, "Network Coding: An Instant Primer," *ACM SIGCOMM Computer Communications Review*, vol. 36, pp. 63–68, January 2006.

[72] J. Liang, R. Kumar, and K. W. Ross, "The FastTrack Overlay: A Measurement Study," *Computer Networks Journal*, vol. 50, pp. 842–858, April 2006.

[73] "The KaZaA Client." URL: http://www.kazaa.com/.

[74] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," in *19th International Conference on Data Engineering*, (Bangalore, India), March 2003.

[75] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling Churn in a DHT,” in *2004 USENIX Annual Technical Conference*, (Boston, MA, USA), June 2004.

[76] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Observations on the Dynamic Evolution of Peer-to-Peer Networks,” in *1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, (Cambridge, MA, USA), March 2002.

[77] S. Ratnasamy, I. Stoica, and S. Shenker, “Routing Algorithms for DHTs: Some Open Questions,” in *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, LNCS 2429*, (Cambridge, MA, USA), May 2002.

[78] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, “Graph-theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience,” in *SIGCOMM 03*, (Karlsruhe, Germany), August 2003.

[79] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, “A Survey of Peer-to-Peer Network Simulators,” in *Seventh Annual Postgraduate Symposium*, (Liverpool, UK), June 2006.

[80] T. Hoßfeld, A. Binzenhöfer, D. Schlosser, K. Eger, J. Oberender, I. Dedinski, and G. Kunzmann, “Towards Efficient Simulation of Large Scale P2P Networks,” Tech. Rep. 371, University of Würzburg, October 2005.

[81] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “PlanetLab: An Overlay Testbed for Broad-Coverage Services,” *ACM SIGCOMM Computer Communication Review*, vol. 33, July 2003.

[82] S. Jain, R. Mahajan, and D. Wetherall, “A Study of the Performance Potential of DHT-based Overlays,” in *USENIX Symposium on Internet Technologies and Systems*, (Seattle, WA, USA), March 2003.

[83] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The Impact of DHT Routing Geometry on Resilience and Proximity,” in *ACM SIGCOMM 2003*, (Karlsruhe, Germany), August 2003.

[84] B.-G. Chun, J. Kubiatowicz, and B. Y. Zhao, “Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks,” in *4th International Workshop on Peer-to-Peer Systems (IPTPS 05)*, (Ithaca, NY, USA), February 2005.

[85] Y. Zhu and X. Yang, “Implications of Neighbor Selection on DHT Overlays,” in *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS’06)*, (Monterey, CA, USA), September 2006.

[86] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the Evolution of Peer-to-Peer Systems,” in *ACM PODC*, (Monterey, CA, USA), July 2002.

[87] D. Leonard, V. Rai, and D. Loguinov, “On Lifetime-based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks,” in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems(SIGMETRICS ’05)*, (Banff, Alberta, Canada), June 2005.

[88] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil, “A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn,” in *24th Infocom*, (Miami, FL, USA), March 2005.

[89] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek, “Comparing the Performance of Distributed Hash Tables Under Churn,” in *3rd International Workshop on Peer-to-Peer Systems (IPTPS 04)*, (San Diego, CA, USA), February 2004.

[90] J. S. Kong, J. S. A. Bridgewater, and V. P. Roychowdhury, “A General Framework for Scalability and Performance Analysis of DHT Routing Systems,” in *International Conference on Dependable Systems and Networks (DSN’06)*, (Philadelphia, PA, USA), June 2006.

[91] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi, “A Statistical Theory of Chord under Churn,” in *4th International Workshop on Peer-to-Peer Systems (IPTPS 05)*, (Ithaca, NY, USA), February 2005.

[92] R. Mahajan, M. Castro, and A. Rowstron, “Controlling the Cost of Reliability in Peer-to-Peer Overlays,” in *2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, (Berkeley, CA, USA), February 2003.

[93] M. Castro, M. Costa, and A. Rowstron, “Performance and Dependability of Structured Peer-to-Peer Overlays,” in *International Conference on Dependable Systems and Networks (DSN’04)*, (Washington, DC, USA), IEEE Computer Society, 2004.

[94] S. S. Lam and H. Liu, “Failure Recovery for Structured P2P Networks: Protocol Design and Performance under Churn,” *Computer Networks*, vol. 50, pp. 3083–3104, November 2006.

[95] S. Wang, D. Xuan, and W. Zhao, “On Resilience of Structured Peer-to-Peer Systems,” in *IEEE Globecom*, (San Francisco, CA, USA), December 2003.

[96] S. Wang, D. Xuan, and W. Zhao, “Analyzing and Enhancing the Resilience of Structured Peer-to-Peer Systems,” *Journal of Parallel and Distributed Computing*, vol. 65, pp. 207–219, February 2005.

[97] D. Wu, Y. Tian, and K.-W. Ng, “Analytical Study on Improving DHT Lookup Performance under Churn,” in *6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006)*, (Cambridge, UK), September 2006.

[98] S. Zoels, Z. Despotovic, and W. Kellerer, “Cost-Based Analysis of Hierarchical DHT Design,” in *6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006)*, (Cambridge, UK), September 2006.

[99] R. Rodrigues and B. Liskov, “High Availability in DHTs: Erasure Coding vs. Replication,” in *4th International Workshop on Peer-to-Peer Systems (IPTPS 05)*, (Ithaca, New York, USA), February 2005.

[100] R. Bhagwan, D. Moore, S. Savage, and G. Voelker, “Replication Strategies for Highly Available Peer-to-Peer Storage,” in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, (Bertinoro, Italy), June 2002.

[101] E. Sit, A. Haeberlen, F. Dabek, B.-G. Chun, H. Weatherspoon, F. Kaashoek, R. Morris, and J. Kubiatowicz, “Proactive Replication for Data Durability,” in *5th International Workshop on Peer-to-Peer Systems (IPTPS 06)*, (Santa Barbara, CA, USA), February 2006.

[102] Y. Kadobayashi, “Achieving Heterogeneity and Fairness in Kademlia,” in *IEEE/IPSJ International Workshop on Peer-to-Peer Internetworking co-located with Symposium on Applications and the Internet (SAINT2004)*, (Tokyo, Japan), January 2004.

[103] D. Stutzbach and R. Rejaie, “Improving Lookup Performance over a Widely-Deployed DHT,” in *IEEE INFOCOM 2006*, (Barcelona, Spain), April 2006.

[104] K. Pawlikowski, H.-D. Jeong, and J.-S. R. Lee, “On Credibility of Simulation Studies Of Telecommunication Networks,” *IEEE Communications Magazine*, pp. 132–139, January 2002.

[105] D. Malkhi, M. Naor, and D. Ratajczak, “Viceroy: A Scalable and Dynamic Emulation of the Butterfly,” in *Twenty-first Annual Symposium on Principles of Distributed Computing (PODC 02)*, (Monterey, CA, USA), July 2002.

[106] K. Horowitz and D. Malkhi, “Estimating Network Size from Local Information,” *The Information Processing Letters Journal*, vol. 88, pp. 237–243, December 2003.

[107] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, “Estimating Aggregates on a Peer-to-Peer Network.” Technical Report, Dept. of Computer Science, Stanford University, April 2003.

[108] M. Jelasity and M. Preuss, “On Obtaining Global Information in a Peer-to-Peer Fully Distributed Environment,” in *8th International Euro-Par Conference on Parallel Processing*, (London, UK), August 2002.

[109] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, “Decentralized Schemes for Size Estimation in Large and Dynamic Groups,” in *4th IEEE International Symposium on Network Computing and Applications (NCA ’05)*, (Washington, DC, USA), July 2005.

[110] R. Bhagwan, S. Savage, and G. Voelker, “Understanding Availability,” in *2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, (Berkeley, CA, USA), February 2003.

[111] D. Stutzbach and R. Rejaie, “Understanding Churn in Peer-to-Peer Networks,” in *6th ACM SIGCOMM on Internet Measurement (IMC 06)*, (Rio de Janeriro, Brazil), October 2006.

[112] K. Tati and G. M. Voelker, “On Object Maintenance in Peer-to-Peer Systems,” in *IPTPS06*, (Santa Barbara, CA, USA), February 2006.

[113] P. B. Goedfrey, S. Shenker, and I. Stoica, “Minimizing Churn in Distributed Systems,” in *ACM SIGCOMM*, (Pisa, Italy), September 2006.

[114] S. Q. Zhuang, D. Geels, I. Stoica, and R. H. Katz, “On Failure Detection Algorithms in Overlay Networks,” in *IEEE INFOCOM*, (Miami, FL, USA), March 2005.

[115] G. Ghinita and Y. Teo, “An Adaptive Stabilization Framework for Distributed Hash Tables,” in *21st International Parallel & Distributed Processing Symposium (IPDPS)*, (Rhodes Island, Greece), April 2006.

[116] A. Singh, P. Maniatis, T. Roscoe, and P. Druschel, “Using Queries for Distributed Monitoring and Forensics,” *ACM SIGOPS Operating Systems Review*, vol. 40, pp. 389–402, October 2006.

[117] M. Zhang, C. Zhang, V. S. Pai, L. L. Peterson, and R. Y. Wang, “PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services,” in *6th Symposium on Operating System Design and Implementation (OSDI 2004)*, (San Francisco, CA, USA), December 2004.

[118] Y. Chen, D. Bindel, H. Song, and R. H. Katz, “An Algebraic Approach to Practical and Scalable Overlay Network Monitoring,” in *SIGCOMM ’04*, (Portland, Oregon, USA), August 2004.

[119] K.-S. Lim and R. Stadler, “Real-time Views of Network Traffic Using Decentralized Management,” in *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, (Nice, France), May 2005.

[120] R. V. Renesse, K. P. Birman, and W. Vogels, “Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining,” *ACM Transactions on Computer Systems (TOCS)*, vol. 21, pp. 164–206, May 2003.

[121] Y. Tang, E. Al-Shaer, and B. Zhang, “Toward Globally Optimal Event Monitoring & Aggregation for Large-scale Overlay Networks,” in *IEEE/IFIP Integrated Management (IM’2007)*, (Munich, Germany), May 2007.

[122] D. Stutzbach and R. Rejaie, “Capturing Accurate Snapshots of the Gnutella Network.,” in *INFOCOM 2005*, (Miami, FL, USA), March 2005.

[123] “The CoMon Project.” URL: http://codeen.cs.princeton.edu/comon/.

[124] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd ed., 1999.

[125] M. A. Stephens, “EDF Statistics for Goodness of Fit and Some Comparisons,” in *Journal of the American Statistical Association*, vol. 69, pp. 730–739, 1974.

[126] E. J. Chen and W. D. Kelton, “Quantile and Histogram Estimation,” in *33nd Conference on Winter Simulation (WSC ’01)*, (Arlington, Virginia, USA), December 2001.

[127] J. F. Kurose and K. W. Ross, *Computer Networking A Top-Down Approach Featuring the Internet*. Addison Wesley Longman Inc., 2005.

ISSN 1432-8801