Technische Universität München
Lehrstuhl für Kommunikationsnetze

Performance Analysis and Optimized Operation of Structured Overlay Networks

Dipl.-Ing. Univ. Gerald Kunzmann

Vollständiger Abdruck der von der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr.-Ing. Hans-Georg Herzog
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Jörg Eberspächer
	2.	Univ.-Prof. Dr.-Ing. Phuoc Tran-Gia, Bayerische Julius-Maximilians-Universität Würzburg

Die Dissertation wurde am 21.11.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 08.04.2009 angenommen.

Performance Analysis and Optimized Operation of Structured Overlay Networks

Dipl.-Ing. Gerald Kunzmann

May 6, 2009


IF YOU CONTINUALLY GIVE,
YOU WILL CONTINUALLY HAVE.
如果您不斷施予，您會不斷有收獲

Abstract

The ongoing process of globalization leads to a huge demand for highly scalable applications. The Peer-to-Peer (P2P) technology enables an arbitrary large number of users to participate and contribute to distributed services like content distribution or collaboration. With P2P, the intelligence is shifted from centralized instances to the edges of the Internet. This shift is supported by the widespread availability of powerful terminals and broadband networks. In P2P systems, these end terminals create a powerful overlay network, which is highly scalable since new users automatically add new resources to the system.
The main goal of P2P overlays is to efficiently find and share distributed resources among users in the Internet. However, the functionality of a deployed system heavily depends on the maintenance of its overlay topology. A disruption of this overlay structure can cause anything from degraded performance or limited functionality up to the point of a total collapse of the system. Thus, current research tackles these problems on different level, like optimized self-organization schemes, efficient overlay maintenance and data transfer, and short lookup delays.
Structured P2P networks arrange resources in the system according to a well-defined overlay structure. Thereby, proactive routing ensures that each node knows about a certain part of the current overlay. As a result, querying arbitrary resources can be performed within a limited number of hops. The most common approach to realize the necessary overlay structure are distributed hash tables (DHTs). In this thesis, we concentrate on the scalability and robustness of these overlay structures, as well as the lookup of resources stored in the overlay.
Based on a detailed simulative and mathematical performance analysis of structured P2P protocols, we show that DHTs scale well with the number of participants, just as they are designed to address the scalability problem inherent to unstructured P2P networks. However, P2P systems operate in dynamic environments and peers frequently join and leave the overlay (*churn*). Therefore, maintaining a correct overlay structure in rapidly changing scenarios is difficult. In this context, it is important to adapt the configuration of design parameters to the actual network size and churn rate.
Based on these results, we evaluate various existing modifications and extensions to

structured P2P protocols. Resulting from our analysis, we are thus able to introduce solutions, which provide a significantly more stable overlay topology in dynamic environments. This is essential for the operation of any P2P system. In addition, a correct overlay structure results in an improved lookup time, as delays due to stale overlay connections are reduced. Moreover, we develop advanced lookup algorithms, which help to further reduce search delays without increasing the required signaling overhead.
Based on our measurements, analysis and simulations, we are able to develop a new structured P2P protocol, which we apply in a Voice-over-IP (VoIP) framework. Concluding this work, we introduce a general concept for realizing services and features with structured P2P systems.

Contents

CHAPTER 1

Introduction and motivation

Two decades ago, Peer-to-Peer (P2P) networking, a novel network architecture initially used for file sharing, was introduced. Experiencing an enormous growth in usage, various P2P systems, for applications like distributed computing and Internet telephony, nowadays attract millions of users. Much of the popularity of P2P is founded on very basic characteristics of P2P networks.

The most important characteristic is, that P2P networks are *distributed* networks meaning that resources are spread among many computers, instead of being stored in a single location. Ideally, there is no Single Point of Failure (SPoF) at all. Adding more clients in a client-server architecture requires providing more servers in order to be able to handle the increased demand on the system. In contrast to that, P2P applications exploit unused and distributed resources, like bandwidth, storage capacity and computing power, found at the edges of the network. Ideally, all participants of a P2P network (peers) equally contribute to and benefit from the system. As all peers provide additional resources, the total capacity of the system is increased with each peer. The distributed nature of P2P in combination with self-organization makes P2P systems highly scalable and resilient. However, P2P is no binary choice between centralization and decentralization. Those aspects of a system that can be better handled at the edges of the networks are decentralized, whereas other aspects, like bootstrapping or security features, may still rely on central instances. All well-established P2P applications centralize specific functionality [TSG+01].

The *distribution of information* is another characteristic in P2P networks. The information is usually distributed between a number of computers. We say, content is *shared* among the peers, and *data is inserted* by a peer. For example, in a file-sharing application, peers contribute files stored on their hard disks. Thereby, each peer acts as the source for files it provides and other peers may directly download files from it. Ideally, after successfully downloading a copy of a file, the peers also share their copies and thus become additional sources for the file. However, in order to improve the lookup for content, it is beneficial to additionally distribute *references* about the location of content

in the network. Due to means of easy publishing of content, P2P benefits from the availability of attractive content.

Self-organization *Self-organization* is the most important characteristics of P2P networks and can be observed in a number of other networks and communities. Many different branches of science, like chemistry, biology, sociology and computer science deal with self-organization. Thereby, researchers study self-organization from different viewpoints and analyze different aspects. While most share basic properties, there is no commonly accepted definition of self-organization.
Our definition of a self-organizing communication network is based on [Prehofer2005]. First, the system structure appears without explicit involvement from outside the system, i.e., the organization of the system results from the interactions among the entities. Additionally, a system will be referred to as *organized*, if it has a certain structure (all entities are arranged in a particular manner) and functionality (the overall system is able to fulfill certain tasks).
Second, there is no fundamental separation between organizing, configuring or controlling parts. All entities are equal and potentially share the same basic functionalities (*redundancy*). Thus, any entity can be removed without loss of the overall functionality, resulting in a highly available system that is *robust* against failures or damage. A good self-organizing system actually will degrade gracefully rather than break down suddenly. Nonetheless, entities might accept special roles.
Third, the interaction (communication) between entities is *localized*. Each entity bases its behavior on its local observations; an overall knowledge about the whole system is not required. All entities follow some simple but important basic rules at the microscopic level, thus creating an organization out of chaos at the macroscopic level. Without central control or management component, the control of the system lies entirely in the hands of the entities themselves. The system continuously *adapts* to changes in the system or environment. Churn (adding and removing entities) is expected and is not critical to the system except for extreme churn rates. The system tries to converge toward a desired beneficial and stable structure by reacting to internal and external triggers. However, usually constant external changes prevent the system from reaching a stable form, and continuously keep the structure in a transient phase.
Fourth, self-organizing systems are complex. Although the rules are simple and may be well understood, describing and predicting the behavior of the whole system in detail is very difficult. In technical systems, that show self-organizing characteristics, all entities are man-made, yet the resulting overall structure and behavior might not be easy to foresee.
Furthermore, most self-organizing systems are able to *scale* to extremely large numbers of entities. We can find several self-organizing systems in nature, e.g., ants building huge colonies[1] without any central control, and with each ant following only a few basic instincts.
Thus, to achieve a better understanding about self-organizing networks, we aim at finding

[1] The largest known colony spans 5760 km along Italy and Spain consisting of several millions of nests and several billions of individuals.

rules about the growth and evolution of the self-organizing structures. Based on these results the next goal is to find methods to predict the future organization, which results from changes caused by its participants or by altered external conditions.
In the context of P2P networking, peers set up connections to a few other peers. In most networks, communication is limited to these adjacent peers. The protocol describes the simple basic rules all peers follow, thus determining the resulting network structure at the macroscopic level [SBD+06, SW05]. Communication between distant peers is achieved by either flooding the network or by exploiting the network structure. The collaboration of all participants creates a new powerful system, which can accomplish tasks no single node would be able to, at the expense of increased communication of the nodes. An up-to-date computer, for example, would be capable of storing status information of all live nodes as well as a metadata description of all data items stored in the network. However, storing all content at this single node is not feasible.
Moreover, *self-organization* makes these networks robust and flexible to dynamic changes with (almost) no operator interaction. Thus, compared with client server applications, management and administrative efforts are significantly reduced. Although the control of the system is shifted from centralized IT departments to individual users, the total costs can be reduced.
Furthermore, in current P2P implementations many design parameters are set to fixed values that are dimensioned for the excepted worst case. [Bin08] introduces a further step toward a more self-organizing overlay structure that automatically adapts itself to the current state of the system. In a simple three-step concept the peer measures the current conditions in the overlay, evaluates the corresponding performance and then adapts its design parameters accordingly.
However, there are still some limitations to the self-organization of P2P systems. For example, consider the bootstrapping process (see Section 2.2.2). To be able to enter the virtual network, a new peer has to know at least one IP address of a node already participating in the overlay network. Thereby, many solutions require a centralized bootstrap server. Also, providing security in P2P networks, e.g., by using public key solutions, is not yet feasible without any central components.
Summarizing, P2P networks have clear advantages to traditional client-server architectures. Thereby, the network's self-organizing nature provides important characteristics like scalability, resilience, and ease of maintenance and configuration. P2P overlay networks also enable a lot of opportunities for novel user-oriented services and innovative applications, as well as reduced maintenance and administration costs. These features mainly account for the increasing popularity of P2P applications.

Impact The breakthrough of P2P networks can be observed by all Internet Service Providers (ISPs) in the world. The most significant and obvious impact is the enormous traffic load they generate. Measurements show that P2P applications, at present especially file-sharing, account for 40–60 percent of backbone traffic and even up to 80–90 percent of local network traffic [TTG05, Mel04, SGG02, ipo]. Furthermore, in a classical client-server architecture, like the World Wide Web (WWW), a relatively small number of servers serve many clients. As a result, the communication is asymmetric with down-

load traffic considerably exceeding upload traffic. In contrast to that, peers in a P2P architecture are content providers and requesters at the same time, thereby using a large amount of upstream bandwidth.
P2P however posses more unique attributes besides the sheer amount of traffic:

1. ISPs observe an increasing symmetry at the access level. Moreover, P2P traffic on the border of a Tier-1 backbone is nearly symmetrical [Joh08].

2. More and more people use file-sharing applications to exchange large files (e.g., videos or CD/DVD images). As a result, flow sizes among terminals are rising from kilobits to gigabits. Mori et. al [MUG05] measured mean values of 20.6 kB for web flows and up to 5.8 MB for P2P flows.

3. P2P traffic differs from non-P2P traffic in terms of duration and inter-arrival times of flows [PDGM06].

4. P2P overlays contribute the major amount of the UDP "flows". Peers periodically send ping and keep-alive UDP messages to keep their routing tables up to date. As a result, the observed flows are very short, typically carrying only 1 or 2 small sized packets.

5. P2P traffic is characterized by a large fraction of unsuccessful and non-malicious outbound connection attempts to non existing hosts [Joh08], as unreliable peers are common in current P2P applications.

6. P2P networks (besides Denial-of-Service (DoS) attacks) explain the extreme amount of several millions distinct IP addresses observed in the Internet traffic [Joh08]. In addition to maintaining large routing tables, file-sharing applications open and sustain connections to a large number of sources, as they try to fully utilize the available download bandwidth, in order to complete downloads as quickly as possible.

Due to the large amount of P2P traffic, many ISPs try to filter, block, or limit P2P traffic. As a result, P2P applications try to disguise their traffic. Thus, P2P traffic is difficult to distinguish from normal web traffic and a pure blocking of ports is no longer a feasible solution to filter P2P traffic. Furthermore, P2P file-sharing applications disguise their traffic in order to evade legal implications. As a result, ISPs and researchers encounter difficulties when trying to analyze Internet backbone traffic.
However, P2P does not only mean additional costs for ISPs. For example, Internet users cited P2P applications as one of the major reasons for upgrading their Internet access to broadband, thus resulting in an increase in revenue for ISPs [Men05]. We even think that ISPs would be able to increase their revenue if they would cooperate with "legal" P2P overlays, like P2P-based communication services. Most current P2P networks establish an overlay topology that is largely independent of the Internet routing, thus impeding the ISPs' traffic engineering capabilities. As a result, P2P traffic often is zigzagging through the physical network, thereby probably crossing network boundaries multiple times [SK03, KRP05, ABFW04]. Adapting the P2P overlay to the physical conditions would significantly reduce P2P traffic, and thus related costs.

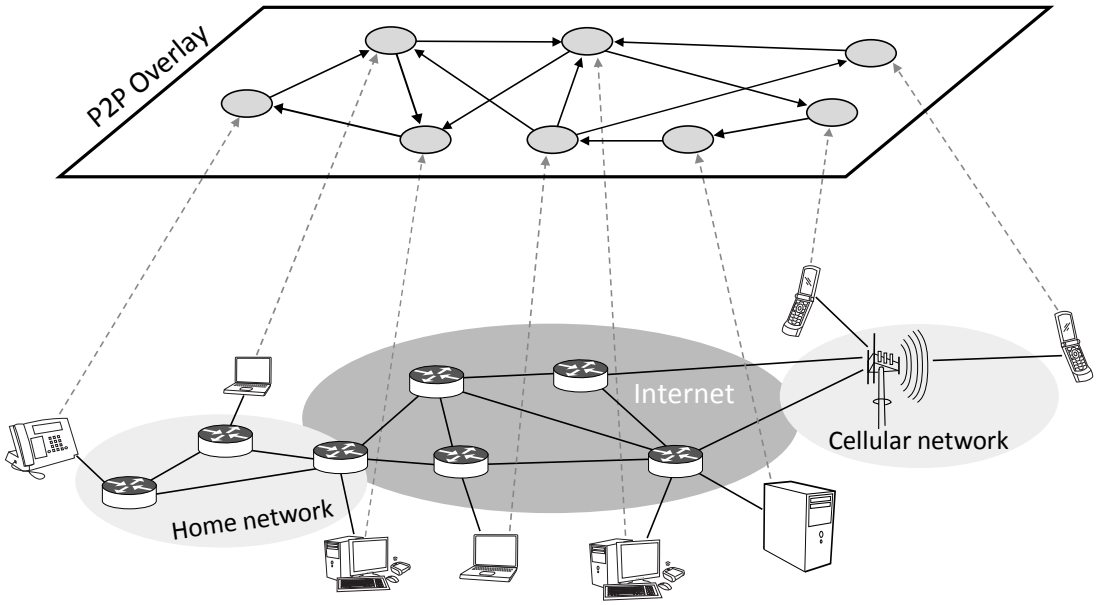


Figure 1.1.: Overlay network: Logical structure on top of an existing infrastructure

1.1. Definitions

In computer science the term **performance** refers to the amount of useful "work" accomplished by a system compared with the time and resources used. The performance is thereby determined by the manner of functioning or operation. Broadly speaking, in the context of P2P overlays, the performance is measured by the duration and the correctness of searches in that overlay. Additionally, signaling and communication overhead, necessary for the maintenance and operation of the overlay, are important performance metrics. Better performance, compared with another system, means faster searches accomplished using fewer resources.
Operation is the method by which a device performs its function. Thereby, we do not apply the term **optimized operation** as a real optimization in a mathematical sense, but use it for describing an increased performance by selecting appropriate values for design parameters, thus improving the operation of the system.
An **overlay network** is a logical network of nodes and logical links that is built on top of an existing network or infrastructure (see Figure 1.1). Usually, overlays are logical structures, i.e., no direct physical connections exist among different nodes, but logical links. Logical links correspond to a path, possibly traversing several physical links in the underlying network. The overlay is an additional topology that commonly applies its own address space, addressing and routing. In its beginnings, the Internet was such an overlay built on top of the telephone network.
Nodes will be called **neighbors**, if they are connected via a direct (virtual) link. Thereby, overlay networks establish a certain protocol-specific **geometry** describing the pattern of virtual links. In **structured** overlays a deterministic geometry of neighbor links is established. Thereby, distributed hash tables (DHTs) are the most common approach to

realize the necessary overlay structure. On the contrary, in **unstructured** overlays each node sets up connections to random nodes, resulting in a more probabilistic geometry.
In general, "**Peer-to-Peer (P2P)** is a mindset, not a particular technology or an industry" [TSG+01]. P2P networks in particular are overlay networks built on top of IP networks. As a result, these networks are independent of specific access networks. Moreover, in this communication model all parties have the same capabilities, take over similar roles, and are able to initiate a communication session to another party. The network relies primarily on the computing power and bandwidth of its participants on the edges of the Internet rather than concentrating it on a relatively low number of servers. Thereby, the main objective of a P2P overlay is to support finding and using distributed resources. Nodes in P2P networks are often referred to as **peers**, thereby emphasizing that they are equal instances (Latin *par*, 'equal'). They connect as equals and are able to share processing, control and access to data and peripherals. Sometimes also the term **servent** is used, pointing out that peers act as SERVers and cliENTs at the same time. In structured P2P overlays peers are addressed by a unique identifier (ID). In this thesis we use the same symbol n for denoting a node and its ID—the meaning can easily be deduced from the context. Similarly, content is addressed by a key k, and the same symbol k is used for denoting the respective content identifier.
The **scalability** of a system refers to its ability to keep pace with changes and growth. No general definition of scalability exists, as scalability can be measured in various dimensions. Usually, an algorithm, networking protocol, or other system is said to be scalable if it can be applied to large situations, e.g., an arbitrary large number of participants in a distributed system, without considerably impacting performance and functionality. That is why we refer to that dimension as **functional scalability**. Furthermore, we consider **stochastic scalability**, i.e., the network's ability to handle a growing rate of changes in the overlay caused by an increased number of membership changes and user activity within a certain time period [Bin08].
The frequency of peers going online and offline is called **churn rate**. A higher churn rate means more stress on the system as the overlay must be adapted to each membership change (see Section 4.1).
Another crucial property is the **stability** of the overlay. A structured overlay will be denoted as stable if all deterministic neighbor links are correct, i.e., each peer established all connections to its neighbors determined by the protocol. As we discuss later on, only a stable overlay guarantees a correct resolution of queries.
In a computer network, the transmission of a data packet between two network nodes is called **hop**. In the context of P2P, one hop is the step from one peer to the next; however, one hop in the overlay may require multiple hops in the underlying IP network. Looking up content may require several hops in the overlay. The **hop count** is the number of subsequent steps along the overlay path from source to sink.

1.2. Contribution

The contribution of this thesis is threefold: *performance analysis, optimized operation, and application of structured overlay networks.* First, as a basis of our work, we carry out a detailed simulative and mathematical performance analysis of structured P2P protocols. In P2P applications, where an arbitrary large number of users may participate in a network, simulation is required to evaluate the performance and scalability of the protocol and to check for its correct behavior. Moreover, simulation is used to predict the behavior of a protocol in uncommon or even undesirable situations.
We show that DHT protocols scale well with the number of participants, just as they are designed to address the scalability problem inherent to unstructured P2P networks. However, maintaining a correct overlay structure is difficult in highly dynamic scenarios, with peers frequently joining and leaving the overlay [KBH05]. Similarly, an optimal configuration of design parameters hardly depends on the network size, but primary depends on the current churn rate.
Based on the evaluation of state-of-the-art modifications and extensions to Chord, we develop novel solutions to improve both the robustness and performance of the protocol. We present *Token Stabilization*, which improves the robustness of the overlay by exploiting Chord's circular structure and sending token-like stabilization messages in both directions [KNE05]. Still, the probability of a disruption of the overlay structure is not negligible and the overlay may split in multiple partitions due to, for example, an organization being disconnected from the Internet. We discuss several *recovery mechanisms* and give design choices that further reduce the probability of disruptions [KB06]. Additionally, we are able to introduce a scalable algorithm for detecting other partitions. Furthermore, we introduce *hybrid routing*, a combination of recursive and iterative routing with the advantages from both variants [Kun05]. For even faster content lookup without additional signaling overhead, we develop *Freebie Fingers* [KS06], an extension to the known Chord protocol. To further increase the lookup performance we present a *fuzzy logic-based route selection*, which combines different routing heuristics and network parameters. In particular, considering the availability of nodes, significant performance gain can be achieved in dynamic networks. Besides our implementation based on the Chord protocol, we discuss ways of translating these concepts to other structured P2P protocols.
Concluding this work, we consider the application of structured P2P for Voice-over-IP (VoIP) systems. We give an overview on supplementary services and add-ons for Public Switched Telephone Network (PSTN) and VoIP-based systems and introduce a general *framework for realizing supplementary services* in a P2P-based VoIP system [SK07]. In this context, the applied lookup protocol must support complex queries, e.g., to provide fuzzy queries for users in distributed white pages. Therefore, we introduce our solution for *prefix-based multi-attribute keyword queries.*

1.3. Outline

The remainder of this thesis is organized as follows.
Chapter 2 gives an introduction to P2P networking. We introduce several classifications of P2P and discuss basic concepts of overlay networks. Subsequently, we focus on different lookup concepts used for P2P networking.
Chapter 3 concentrates on structured P2P lookup concepts. It provides an in-depth introduction and evaluation of the concepts and technologies, which provide the basis for this dissertation. Starting with a general introduction to DHT-based protocols, we take a closer look at selected structured P2P protocols. In the following, we present additional algorithms for increasing content availability and load balancing. Concluding this chapter, we analyze non-DHT protocols, which inherently support load balancing and tree functionality.
Chapter 4 introduces a new simulation environment used throughout the course of this work. Therefore, we first present our new efficient approach to model user behavior and the physical network. Based on this model, we describe the design of our simulator and the workflow of the simulation and specify relevant design parameters.
In Chapter 5, we provide a detailed analysis of the Chord protocol, a well-known representative of DHTs. After introducing fundamental metrics, we evaluate the protocol in regard to these metrics. Based on our analysis, we conclude this chapter with a comparison to related protocols.
In Chapter 6, we analyze and evaluate state-of-the-art modifications and extensions to Chord. Resulting from this analysis, we develop new solutions to improve the robustness and performance of the protocol. In addition, we consider possibilities to translate these concepts to other structured P2P protocols.
Chapter 7 presents the application of structured P2P for VoIP. We introduce solutions for realizing supplementary services and keyword-based queries in P2P-based VoIP systems.
Chapter 8 concludes the thesis by summarizing the main findings and contributions. Based on these we name limitations, open issues, and approaches for future work.

CHAPTER 2

Peer-to-Peer (P2P) overlay networks

2.1. Classification of P2P overlays

A common classification of P2P systems is based on its different application areas (see Table 2.1). We distinguish between *information sharing*, e.g., file-sharing and location based services, *personal communication* like telephony (e.g., Skype, *community and collaboration services*, i.e., (spontaneous) forming of virtual communities, *data streaming* for video and audio, and *networking services* like a P2P-based distributed DNS (DDNS). Independent from a specific application area, *incentive mechanisms*, like 'Tit-for-Tat' or credit systems, try to motivate users for sharing more content and over a longer time period. The basic idea behind such mechanisms is that the more actively a user participates in the network, the better he will be served.
P2P networks can be applied to different layers of a communication system (see Table 2.2) [AH04]. In the preceding paragraphs we focused on the *data access* and *service* layers. The data access layer consists of the overlay network and provides basic functionalities for storing and searching of resources using application specific identifiers. Prominent examples are Gnutella [Cli00] and Freenet [CSWH01]. The service or application layer enhances the data access layer and builds an application on top of it. Various kinds of applications can be realized as mentioned above.
In this model, the lowest layer is called *networking* layer. It includes basic services to route messages over a physical network in an application independent way, like TCP/IP in the Internet. P2P technology is also used to provide a distributed monitoring service for network statistics [FKSK06]. Furthermore, several Next Generation Internet (NGI) concepts exist based on structured P2P networks, e.g., [EFK03, FDKC06, KCC+07, HSKE09].
On the contrary, the topmost layer in that model is the *user* layer. It is characterized by direct interactions of users belonging to social communities. This layer is no longer of technical nature; however, users exchange items in a Peer-to-Peer (P2P)-like manner.

Class	Application	Example system
Information sharing	File sharing	Napster [Nap], BitTorrent [Coh]
	Photo sharing	[ACMDH03]
Community and collaboration services	Collaboration	Microsoft Office Groove [Mic], Croquet project [cro]
	Spontaneous forming of virtual communities (based on location)	[Zün07, TB04]
	Gaming	Solipsis [Fra], Microsoft XNA [Mic07]
Personal communication	(Video) telephony	Skype [Skyb], PeerThings [Wim06], P2PSIP [BR]
Data streaming	Video streaming	Joost [Mac07], HotStreaming [WPL+06], SopCast [Sop]
Networking services	Distributed DNS	DDNS [Boc, CMM02]

Table 2.1.: Application areas of P2P

An example would be the online auction website eBay [eBa].
Talking about P2P technology, most people refer to the data access layer and service layer. Also, most algorithms and evaluations we deal with in this thesis correspond to the data access layer. In Section 7 we present a P2P-based telephone system that is situated in the service layer.
P2P networking can be split into two main aspects, namely *lookup* and *data delivery*. Lookup concepts aim at finding data that is provided by another peer. We say "data has been inserted into the P2P network" by that peer. In centralized and structured P2P networks, peers register their shared content with a central or distributed index database. This index can later on be used to efficiently find peers that share the requested content. We refer to this kind of strategy as *proactive querying*, as some effort for indexing the data must be applied in advance of the queries. In contrast to that, peers that participate in unstructured P2P networks do not initially publish any information about the content they share. Therefore, searching content requires asking all peers whether they provide the requested data, for example, by flooding the network with query messages. That is why we call this strategy *reactive querying*.
After a requesting peer has found potential sources for the content it is looking for, the data must be delivered to it. Data delivery concepts cope with efficiently transmitting data from one or multiple sources to the requesting peer. Depending on the application area, different requirements must be fulfilled. In the classical file-sharing application, data shall be transmitted as quickly as possible. Therefore, files are, for example, split in several parts, so-called *chunks*. Consequently, different chunks can be downloaded from multiple peers simultaneously, resulting in a higher total download rate. Media streaming applications require the packets of a stream to be transmitted in a correct order and within certain time constraints. Thus, building an efficient distribution tree is crucial for such applications. Users receiving a media stream must forward the stream to

Layer	Description	Application Domain	Service	Example System
User	Interactions of users belonging to social communities	User communication	Collaboration	eBay [eBa], Ciao [Cia]
Service	Combination and enhancement of data access layer functionalities to provide higher-level abstractions	P2P applications	Messaging, distributed processing	Napster [Nap], ICQ [Mir]
Data access	Search and update of resources using application specific identifiers in a distributed environment	Overlay networks	Resource lookup & delivery	Gnutella [KM02], Freenet [CSWH01]
Networking	Basic services to route requests over physical network in an application independent way	Internet	Routing	TCP/IP

Table 2.2.: Application of P2P concepts in different layers of a communication system

at least one other user in order to build such a tree. Massively Multiplayer Online Role-Playing Games (MMORPG) must take a special focus on security in order to prevent gamers from cheating.
For most applications, it makes sense to establish direct TCP/UDP connections between the source peers and the requesting peer. On the contrary, some applications prefer to route the content along various peers that act as proxies, for example, in order to anonymize the sources as well as the sink [CSWH01]. In this thesis, data delivery concepts are not examined in greater detail, instead we focus on lookup aspects. Hence, we refer to [Sto01, Li08] for an in-depth discussion of data delivery.

2.2. P2P lookup concepts

2.2.1. Centralized P2P overlays

Application layer P2P networking started in 1998, when Shawn Fanning finished implementing an easy method to search and exchange music files over the Internet. His *Napster* network was the first file-sharing service. Its functionality and ease of use attracted thousands of users within a few days.
The software clients connect to a central Napster server [SGG03]. Then the clients scan the local hard disks for MP3-files, and report the results to the server. The server acts as a central index database that maintains an index of all files that are shared by the peers currently logged on. It does not store the files itself, but records ⟨filename; IP address:port⟩ pairs. Queries for content are also sent to that server (see Figure 2.1 ①). For each query, the server returns a list of computers (i.e., their IP address and port number) that share the queried file ②. The querying clients then choose the "best" source, for example, based on connectivity or provided data rate, and establish a direct connection ("Peer-to-Peer") in order to transmit the music file ③. As a central index server is mandatory, we classify that kind of lookup concept as *centralized P2P*.

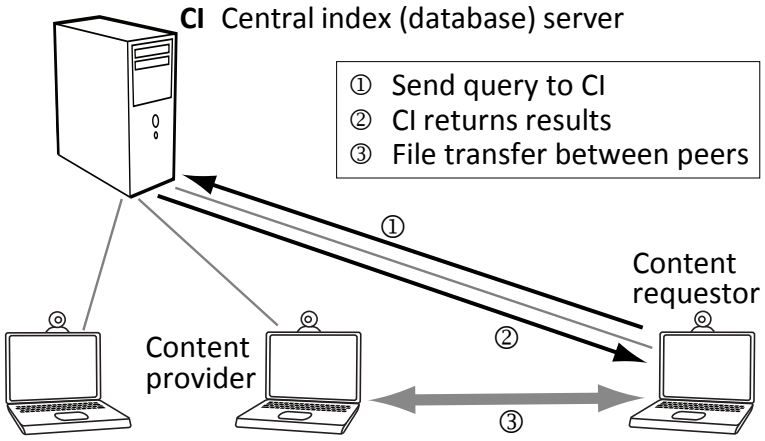


Figure 2.1.: Centralized P2P: The server acts as a central index database.

Two years later, McCaleb released the client and server versions of *eDonkey2000* [Kli]. The eDonkey network also uses a centralized index database for looking up content, and content is exchanged directly between users. Its main improvement compared with Napster was the use of *swarming*. Thereby, files are split in so-called *chunks*, allowing clients to retrieve different pieces from different peers simultaneously. Most private users possess an asynchronous Internet connection (a result of the traditional client-server architecture), where the upload data rate is considerably smaller then the download rate. Therefore, downloading different chunks of a single file from various sources can significantly accelerate the download process. Additionally, eDonkey2000 servers index file hashes. Thus, identical files that have different files names across different peers can be identified, and the number of potential sources for a certain file is increased.

Napster saw its peak of use in February 2001 with 26.4 million users worldwide [Jup01]. However, the fast rise came to an abrupt stop in 2001, when Napster was found guilty for contributory and vicarious copyright infringement. Napster was forced to shut down its service, as it was not able to block access to infringing material. Deprived of its central index server, the Napster network was no longer functional. In centralized P2P networks, the central entity is a Single Point of Failure (SPoF). Although it is easier to protect a single server from different kinds of attacks than protecting all of the clients, the Napster server was easily brought down by the U.S. District Court.

In centralized P2P networks computing and storage power, as well as the available bandwidth of the server must grow proportionally to the number of users. Thus, scalability is one problem of centralized approaches. Services like Google [Goo] prove that they are able to manage the huge number of clients. However, as Google's index of the Web is too large and would not fit on a single machine, it is distributed across many machines [Lon04]. Also, in NGI visions almost all devices will be equipped with Internet access, thereby significantly increasing the number of clients.

A benefit of centralized solutions is that the server has a complete view of the network. Thus, no routing in the overlay network is required. Also, keep-alive signals or electronic heartbeats are not necessary to organize the overlay. In contrast to that, in unstructured

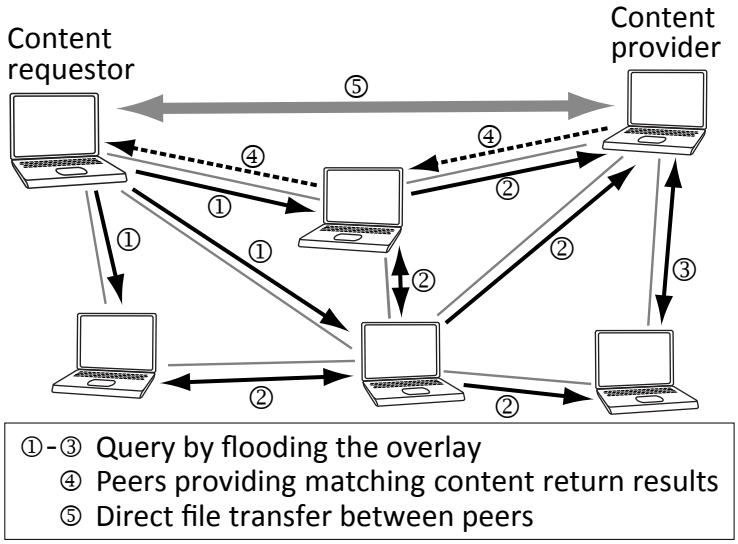


Figure 2.2.: Unstructured P2P: Peers establish random connections to each other.

and structured P2P networking, routing and signaling messages consume a huge amount of bandwidth.
Complex query functionality, such as wildcard queries, similarity queries or the detection of typing errors, can also easily be supported by centralized P2P systems, as all data is available at the server.

2.2.2. Unstructured P2P overlays

Unstructured P2P networks completely avoid any central instance and all peers possess the same functionalities. Peers organize in a partly meshed overlay network (see Figure 2.2), and they exchange service and query messages with each other [ES05]. Gnutella [Cli00] and The Free Network Project (Freenet) [CSWH01] are the most prominent examples of unstructured P2P networks.
Gnutella was released in early 2000 by Nullsoft, Inc. which belongs to AOL LLC. The very next day, AOL stopped its availability due to legal concerns. However, many reverse engineered open source clones appeared based on the Gnutella 0.4 protocol draft [Cli00].
The Freenet project exploits the unstructured overlay to prevent censorship and to ensure anonymity in order to provide freedom of speech to the users participating in the network. Therefore, all messages are encrypted and routed along many hops. Content is also encrypted and replicated across a large number of continuously-changing anonymized computers. Thus, it is extremely difficult to determine who is requesting and providing which piece of information.
In order to join such a system, a peer must know at least one other peer that is already participating in the overlay. Therefore, most clients store a list of IP addresses and port numbers of peers that the client was connected to in a previous session. For an initial setup, a list of "always-on" peers is provided in the installer or can be downloaded from certain websites. Many networks also provide a kind of beacon or bootstrap server. It

caches peers that recently have logged on to the system via this server with a temporarily IP address. Thus, the bootstrap server is able to provide several IP addresses of active participants. After a peer has established at least one connection to any other peer n, it learns about more peers from gossiping with n. Then, it tries to establish connections to known peers by sending a connection request to them. A peer that receives such a request will either accept or reject the request, depending on, for example, its current number of established connections (referred to as *node connectivity*), the availability of free connection slots, or the compatibility of the used protocol versions. Existing connections are maintained by periodically sending keep-alive messages (heartbeat) to all connected peers. If no more keep-alive messages are received from any neighbor within a certain timeout interval the neighbor will be considered as stale, the connection will be closed, and a new connection to another known peer will be established.
Thereby, peers act completely autonomous—there is no central instance that coordinates and maintains the network and its participants. In unstructured P2P networks, the protocol defines the way nodes communicate with each other. It also recommends values for certain parameters, like the number of connections that are maintained to other peers. However, compared with structured P2P protocols, it does *not* determine the overlay network structure, i.e., the connections between the nodes in the overlay topology are random and not predetermined.
Another characteristic of unstructured P2P networks is that content is not managed or indexed. Content is only stored at the nodes that provide the content. It is not shifted or replicated to other nodes in order to reach an optimal distribution. Neither an index of content like in centralized or structured P2P networks is maintained, nor are pointers to the offered content distributed in the network. Thus, searching content must be realized by asking many nodes whether they provide that particular content, much like in a bazaar or flea market. Thereby, popular content that is available at many nodes can be found quickly, whereas finding an extremely rare content is difficult, requires much effort, and a lookup success is not guaranteed. Different strategies for finding the required content may be used.
In Gnutella query messages are *flooded* through the network, i.e., incoming messages are forwarded to all neighbors in the overlay, except the neighbor the message was received from. However, in large networks it is not feasible to forward a query message to all nodes. Thus, a Time-to-Live (TTL) counter is introduced. The TTL counter is decremented by one each time a message is forwarded. Nodes will not forward a message if the TTL counter reached zero. The default TTL value in Gnutella is $h = 7$ [Sch05], i.e., a message is forwarded 7 times. If we assume a network that is free of loops and nodes have an average connectivity of $c = 3$ [SD03], each message would be flooded to $N(c, h) = 381$ nodes, with

$$N(c,h) = \sum_{i=0}^{h-1} c \cdot (c-1)^i = c \cdot \frac{1-(c-1)^h}{2-c} \tag{2.1}$$

Both, the number of queried nodes, as well as the number of messages, will increase significantly if the network has a higher connectivity or a larger initial TTL value is used. However, in real networks, our assumption of an overlay structure without loops is not realizable. As nodes have only a local view on the network and connections are

established randomly, loops cannot be prevented. A Globally Unique Identifier (GUID), included in every message, helps to identify messages that have been received multiple times due to loops. Nodes store a cache with messages they received. If any message is received twice within a certain time, the message will be discarded as it has already been forwarded by that node.
In Freenet, GUIDs are also used for returning an answer to a request along the same path the request has traveled. Thus, no information about the initiator and the content provider must be included in the message, thereby ensuring the anonymity of both parties.
However, due to the limited TTL counter it is not guaranteed that a peer finds the desired content in the network. The small-world experiment [Mil67], however, showed that members of large social networks (in this case, the population of the United States) would be connected to each other through short chains of roughly six acquaintances. A recent study on a large instant-messaging network with 240 million users confirms that the average path length between any two users in the network is 6.6 hops [LH08]. Affirmed by these results, we can estimate that social searches and thus as well searches for content can reach their targets in a median of five to seven steps, although actual success depends strongly on individual incentives. In addition, the more interesting content is for users, the more it will be shared by many participants, resulting in an even higher success probability.
Flooding the network is very resource consuming and creates a huge message overhead. Therefore, in the literature a number of improvements have been proposed, which we will briefly discuss in the following paragraphs. *Expanding ring search* implements a Gnutella like search starting with a small TTL counter value, that will be increased iteratively if there is no query success, until a certain limit is reached [LCC+02]. As a result the number of messages is reduced and fewer nodes are visited for successful queries compared with query flooding. However, the average search delay is increased as the iterative expansion of the ring is time-consuming.
A second alternative are *random walks* [LCC+02], which further reduce search overhead at the expense of increased search latency. The requesting peer is sending out the query to k neighbors only. Peers receiving the request will check back with the initiator of the request if the resource has already been found. If not, they will forward the query to one neighbor. Thus, the request is not flooded in the overlay network, but k parallel "random walks" are performed. [LCC+02] reports that random walks can further reduce the number of messages compared to expanding ring search by half.
Additional alternatives to flooding can be found in [ALPH01, SBR03]. However, a more promising solution to reduce the waste of network capacity are hierarchical P2P networks (see Section 2.2.4).
In most P2P networks, the overlay is not matched to the underlying physical layer, as nodes establish random connections to other nodes. Thus, neighboring nodes in the overlay can be located far away in the physical network. Hence, a message must be transmitted along many hops in the physical layer. This implies two consequences: The average one-hop delay in the overlay is much longer, and more network capacity is used in the physical network. As a result, the average resolution of a query takes much longer, as each overlay hop has a longer Transmission Time (TT). Even worse, many queries

are forwarded along zigzag routes. In our measurements in the Gnutella network, we found out that quite a few queries cross the Atlantic Ocean multiple times [SK03]. In order to reduce that waste of network capacities, a geographically sensitive adaption of the Gnutella protocol is proposed in [SK03].
Summarizing, the unstructured organization of the overlay bears two main disadvantages. First, as it is not feasible to query the complete network, it cannot be guaranteed that queries are resolved although the queried content exists. The more scarcely distributed content is, the more queries for that content are not resolved. A high replication rate is necessary to provide a high success rate for all content. Second, flooding results in high signaling traffic. Introducing a hierarchical overlay (see Section 2.2.4) can significantly reduce this waste of network capacity.
In return, unstructured P2P networks are extremely fault resistant. Any peer can be removed without loss of functionality, because there is no central entity and all peers provide the same functionalities. Even a sudden outage of a huge portion of peers will not completely destroy the network. Furthermore, as each peer tries to match the query to its locally available content, complex query functionality can be provided (see Section 7.3).

2.2.3. Structured P2P overlays

A third lookup concept was introduced in 2001, the so called *structured P2P* networks. Nodes and resources are organized in a deterministic structure, thus ensuring that any node will be able to efficiently route a query message to a peer that has the desired resource, even if the resource is extremely rare. Thereby, lookups benefit from knowing the location where the queried content should be located in the network. Thus, query messages can be forwarded on an almost direct path without the need of flooding the network. Arriving at its deterministic destination, a lookup can be resolved at any rate: If the resource is available, it will be returned to the initiator of the query. Otherwise, the query will be answered with a 'resource not available' message. In this context, resources can either be the files themselves or metadata describing the files. Different structures have been proposed in literature, e.g., a one-dimensional ring-shaped structure [SMK+01a], a multi-dimensional torus [RFH+01], or a graph structure based on hypercubes [SSDN02]. In order to successfully route requests, effort for building and maintaining the structure is necessary. We call that strategy *proactive querying*, as this effort must be spent prior to the lookups.
Most structured P2P networks are based on DHTs. An Identifier (ID) from an m-bit ID space is assigned to every node. The overlay structure is set up according to these IDs, hence, nodes are arranged with increasing IDs. Thereby, each peer is responsible for a certain part of the ID space. Like in a traditional hash table, each data item (value) is assigned with an ID (key). A $\langle \text{key}; \text{value} \rangle$-pair is stored at the peer responsible for the corresponding key. In general, nodes are responsible for keys that are close (according to a certain metric) to the node ID. Content IDs (keys) are generated by hashing a certain characteristic of the data item, e.g., its filename. Consequently, any peer requesting a particular file calculates the corresponding ID by using the same hash function, and routes the query directly to the peer responsible for that ID.

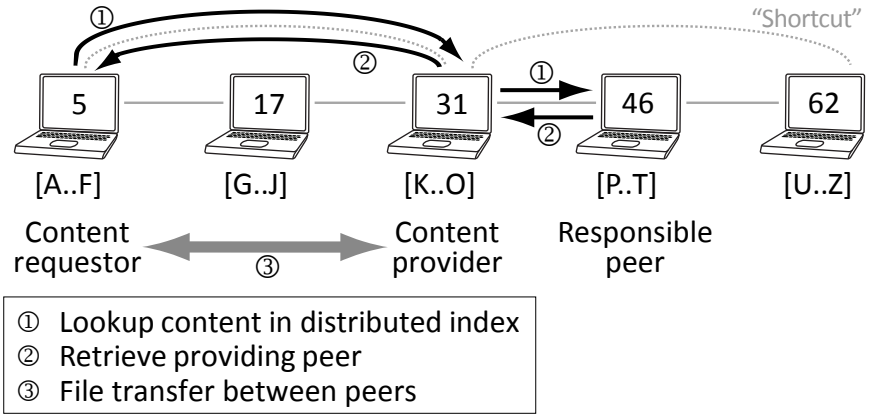


Figure 2.3.: Structured P2P: The overlay structure is determined by the node IDs.

Structured P2P protocols try to remedy the disadvantages of unstructured P2P protocols we discussed in the previous section. By maintaining the deterministic overlay structure, it is always possible to resolve a query within a bounded number of hops in a stable network. This bound is a function of the overlay size, yet it is independent of the frequency of the data item. Even if the queried content is not available in the system, an empty list of found items will be returned to the user. In contrast to that, in unstructured P2P networks, queries will only be resolved if "by chance" a node is hit, that is sharing the queried item. Also, by routing the queries instead of flooding them, the signaling overhead during queries is reduced by a considerable amount.
However, additional signaling overhead is necessary to maintain the overlay structure. Each time a peer joins or leaves the network, the structure must be adapted, and ⟨key; value⟩-pairs (references) in the affected region must be shifted. Nodes leaving the network must shift all references to adjacent nodes that are responsible for the keys from this moment on. In contrast, nodes that join the network must take over references with keys assigned to the ID range they are responsible for. Nodes joining and leaving the network can still be handled without affecting the content availability. Nodes that fail, without notifying their neighbors, for example, due to link breaks or power outages, cause considerable problems. References that are stored on these nodes are lost. This is why references must be replicated, thus significantly increasing the signaling overhead. Furthermore, in order to repair the defective structure, it is important to detect such failures as soon as possible (see Section 5.2.3). Therefore, like in unstructured P2P networks, peers have to monitor their neighbors by sending periodic keep-alive messages to each other.
We can conclude that structured P2P networks cause low signaling traffic in stable environments, because flooding is avoided. However, the higher the churn rate, i.e., the more frequently nodes join and leave the network, the more signaling messages must be sent. In the worst case, the constant changes hamper the construction of the overlay. Like unstructured P2P networks, this type of P2P networks has no central entity and presents no single-point-of-failure. Yet, the overlay structure is critical for resolving queries and thus is a potential weak spot.

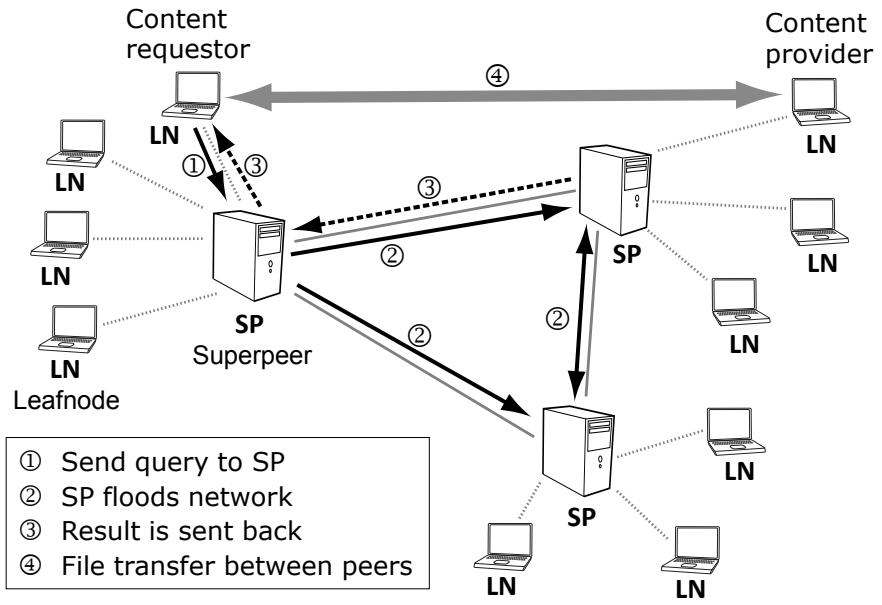


Figure 2.4.: Hierarchical P2P: A peer's capacities determine its level in the hierarchy.

Another challenge lies in the fact that peers must establish deterministic connections to other peers. In the current Internet architecture this poses to be a problem, when peers have private IP addresses and are shielded by firewalls and Network Address Translation (NAT). Then establishing a direct connection to a deterministic peer might not always be feasible.
In this section we mainly focused on DHTs. However, there are also some structured P2P networks that are not based on a DHT. Skip Graphs [AS03] and HyperCup [SSDN02], for example, are based on skip lists and hypercubes, respectively. In Chapter 3 we have a more detailed look at selected structured P2P protocols.

2.2.4. Hierarchical P2P overlays

In 2002 a hierarchical version of Gnutella (Gnutella v0.6 [KM02]) was developed, which significantly reduces the number of maintenance and lookup messages. Gnutella v0.6 establishes a two-tier hierarchy (see Figure 2.4). Thereby, exceptionally available and powerful nodes, so-called "supernodes" or "superpeers", are grouped in the top-level. The superpeers set up an unstructured P2P overlay by establishing random connections to each other. Other peers, called "ordinary peers" or "leafnodes", are directly attached to one of the superpeers. Each superpeer acts as a central index database for all leafnodes, which are connected to it, much like in a centralized P2P system. Hence, superpeers can shield their leafnodes from the signaling traffic in the top-level.
Nodes joining the system connect to one of the superpeers, for example, the one with the shortest Round-Trip Time (RTT). Consequently, they register content they provide with their superpeer. If leafnodes fulfill certain criteria, like a public IP address, a large amount of bandwidth, high processing power, and long online times, they may be promoted to superpeer status.

Leafnodes querying for content forward the request to their superpeer. The superpeer searches its local database and returns matching content. If no matching content is found, the superpeer will flood the query in the top-level overlay. Compared to flat unstructured P2P systems, the same query hit ratio can be obtained by flooding a considerably smaller number of peers (using a smaller TTL value), as each superpeer provides the aggregated content of multiple peers. Finally, the requesting peer connects to the leafnode that provides the queried content. If the peers are able to establish a direct connection between each other, no superpeer will be involved in transmitting the data. Otherwise, the corresponding superpeers may act as proxies, helping to establish a connection, for example, by using UDP/TCP hole punching [FKS05], or they even relay the complete communication like in Skype [BS04].
A hierarchical P2P lookup also reduces the average path length in hops, as in many cases frequent data is available at one of the local leafnodes of the superpeer. If not, following the same reasoning as above, a query hit will be more likely at a close superpeer. The lookup latency will be further reduced if leafnodes are assigned to that superpeer with the lowest RTT.
The popular file-sharing application KaZaA [Sha] is based on the FastTrack protocol. Although FastTrack is a proprietary development, reverse engineering showed that a two-level architecture similar to Gnutella v0.6 is used [giF]. Later versions of eDonkey2000 also connect servers with other servers, thereby establishing a hierarchical overlay.
Such hierarchical P2P systems exploit the fact that nodes have heterogeneous capabilities. A few powerful superpeers are able to index the content shared by their leafnodes, as well as to handle all query requests. A more general framework for hierarchical P2P lookup is proposed in [GEBR+03]. Peers are organized into groups according to some metric (e.g., topological closeness). Each group selects at least one superpeer that has special characteristics (e.g., the most reliable peer). This peer is assigned with additional responsibilities, like acting as a gateway between the groups. Similar to the Gnutella v0.6 network, lookups are first routed to the right group, and then a local query is initiated. Unlike Gnutella v0.6, superpeers do not index content available in their group. The hierarchy can also be extended to more levels.
The authors show that such a hierarchical network can significantly reduce the expected number of overlay hops, and thus the lookup latency. They also argue, that hierarchical structures can provide administrative autonomy to participating organizations, as lower-level groups may belong to different organizations that implement their own lookup protocols. The hierarchical framework does not specify any overlay structure or lookup service at any level in the hierarchy. In their work, Garcès-Erice et al. dwell on a specific two-tier hierarchy that uses Chord [SMK+01a] for the top-level and arbitrary DHTs for the bottom level overlays.
Zöls et. al [ZDK06] provide a cost model of a specific two-tier hierarchical DHT. Similar to Gnutella v0.6, the most powerful peers are called superpeers and are positioned in the top level, whereas less-performance peers are directly attached to these superpeers. Yet, in contrast to Gnutella v0.6, superpeers run the Chord protocol. The authors evaluate the costs for different superpeer ratios. In a system with N heterogeneous peers they evaluate the total network costs as well as the individual costs per peer. Thereby, they vary the number of superpeers from 1 (centralized P2P system) to N (flat

P2P system). Summarizing, they show that centralized systems possess the lowest total network costs, but the single superpeer must be able to handle the highest individual costs. In contrast to that, individual costs are small, but total network costs are very high for flat architectures. In a sample scenario with DSL, UMTS, and GPRS peers they demonstrate that a hierarchical architecture must be used, as neither a centralized system (due to an overloaded superpeer), nor a flat system (due to overloaded GPRS leafnodes) is applicable in that scenario.
Summarizing, hierarchical approaches are an efficient solution to reduce the amount of used network capacity. By positioning resource-constrained nodes in the lower level, these peers are shielded from the high traffic in the upper levels. Additionally, nodes in the upper level are shielded from the high churn rates of nodes with error-prone wireless connections. Thus, hierarchical approaches are especially suited for mobile environments.

2.2.5. Comparison

Table 2.3 summarizes similarities and differences of the presented lookup concepts. The most apparent differentiation is the network structure. In centralized overlays, peers group around a central entity acting as an index database. In contrast to that, (un-) structured overlays self-organize without the need of a central instance, thus avoiding a SPoF. In unstructured P2P networks peers set up about 3–7 connections to other randomly chosen peers, and failed peers are simply replaced by new connections to any other peers. As a result, this kind of overlay is extremely stable. Although structured overlays have no SPoF, the deterministic construction of links is sensitive to failures. Efficient stabilization algorithms must be applied to replace failed connections. However, the number of connections to other peers (usually $O(\log_2 N)$ connections are maintained) is sufficient to keep the probability of peers becoming isolated from the network very low. Private IP addresses might prove to be an even greater challenge for structured overlays. The network structure determines which peers must connect to each other. Yet, this connection could be hard to set up as some peers are situated behind NATs and Firewalls.
In P2P applications, content is usually stored at the peer providing the content and, in contrast to client-server approaches, content is directly transferred from one peer to another. However, the lookup of content is organized in different ways. In a centralized lookup, all references to content are stored in a central index database, thus providing a lookup path length of one hop. Structured lookup concepts store references at deterministic peers, that is, in combination with the structured overlay, guarantees on the mean and maximum path length can be made. In contrast to that, no references are shifted in unstructured overlays, that is why flooding or random walks must be used for looking up keys. As a result, the lookup path length in unstructured overlays depends on the popularity of the queried content. The more often an item is stored in the P2P network, the faster one of the peers providing that content will be found. Moreover, the resolution of queries is not guaranteed, as small TTL counters are used limiting lookups to a node's close neighborhood.

	Centralized	Unstructured	Structured	Hierarchical
Network structure	Centralized	Random	Determined by node IDs	Combination
Central entity	Index database	No	No	Dynamically elected (Peers with surpassing resources)
Stability	Single-Point-of-Failure	Extremely stable	Sensitive overlay structure	In between centralized and (un)structured P2P
Content references	Stored at server	Remain at providing node	Shifted to responsible node	Combination
Routing of queries	No	Flooding, Random Walks etc.	Direct by using routing tables	Combination
Lookup path length	1	$\leq TTL$, no guaranteed query resolution	Usually $f(N)$	Combination
Routing state	1	Approx. 3–7	Usually $f(N)$ or $f(d)$	Combination
Signaling overhead	Minimal	Medium to High (depending on the protocol)	Low to High (depending on user behavior)	In between centralized and (un)structured P2P
Reliability	Server returns multiple download locations	Multiple peers with data may reply to request	Replication of data on multiple peers	Combination
Complex queries	Yes	Yes	Difficult	Depending on architecture
Examples	Napster	Gnutella v0.4, Freenet	Chord, CAN	Gnutella v0.6

Table 2.3.: Comparison of P2P lookup concepts

Centralized P2P solutions result in the lowest total signaling costs, yet the central entity must handle significantly higher load than its leafnodes. In contrast to that, unstructured and structured lookups try to balance costs on all participating peers, thus resulting in higher total communication costs. Unstructured lookup concepts require low maintenance traffic, but high query traffic, whereas a structured lookup is cheap, but maintaining the underlying overlay is expensive. The costs in unstructured P2P overlays can be reduced by improving the protocol. For example, clever random walks are more efficient than a simple flooding solution. By contrast, the costs in structured P2P overlays mainly depend on the user behavior, i.e., the higher the churn rate, the higher the costs of maintaining the overlay structure. In our analysis we show that structured overlays do not scale to extreme churn rates.
The reliability of content is reasonable for all concepts. In centralized networks, the central entity returns multiple download locations. Similarly, a lookup in a structured P2P network will return multiple potential sources if available. In contrast to that, in unstructured overlays it is likely that multiple peers answer to queries for common content; however, fewer or no query responses at all are expected for unpopular content.
Complex queries (Section 7.3) are supported by centralized and unstructured overlays. By contrast, the hash functionality provided by DHTs does not support complex queries at all, thus additional mechanisms must be implemented on top of these protocols to add this feature. Other structured P2P protocols (e.g., based on trees) provide range queries, but lack efficient load balancing.
Hierarchical solutions are often combinations of two different lookup concepts, and thus also their properties and functionalities are a mixture of both concepts. Various hierarchical solutions for various operational areas exist. Usually, on the one hand powerful peers are elected for the top level and take over extra tasks. On the other hand, resource-constrained peers are arranged in the bottom layer and are shielded from most maintenance and lookup traffic. Thereby, the strengths of different lookup concepts can efficiently be combined. In the course of this work we do not examine hierarchical solutions in detail, as we concentrate on analyzing the interworking and operational behavior of pure structured overlay networks.
Summarizing, pure centralized approaches are past their peak. However, most current P2P applications still rely on a few centralized instances, for example, for storing user account information, for bootstrap and login purposes, or for providing security features like verifying certificates. Unstructured P2P networks excel other concepts in their extremely stable overlay, yet, searching the overlay is very expensive and finding existing content is not guaranteed. As a result, unstructured solutions lose ground to structured overlays. The main strength of structured P2P networks is that answers to queries are always possible. This feature is crucial for many kinds of applications. Also, the improvements presented in this thesis, as well as the ongoing research, will result in structured overlays that can also be applied in scenarios with high churn rates. As a result, we believe that structured overlays (also as part of hierarchical solutions) will be implemented in most distributed NGI applications.

CHAPTER 3

Structured P2P lookup protocols

As their name suggests, structured P2P networks arrange resources in the system according to a well-defined structure. Thereby, proactive routing ensures that information about a certain part of the current structure is known to each node. Thus, querying arbitrary resources can be performed within a limited number of hops [ESZK04, KKSZ06]. The most common approach to realize the necessary overlay structure are *distributed hash tables* (DHTs).
In this chapter, we discuss several structured P2P protocols. A protocol specifies both the overlay structure and its maintenance (referred to as *stabilization*), as well as the way lookups for keys are realized. In contrast to that, content management and efficient methods for transferring content from one peer to another are independent from the selected overlay protocol. Thus, we concentrate on the structures of the protocols and the implemented lookup algorithms. We also give a short summary of replication and load balancing techniques applied in structured P2P overlays in Sections 3.2 and 3.3.

3.1. DHT-based protocols

Most structured P2P protocols are based on distributed hash tables (DHTs), thus sometimes the term DHT is falsely used for structured P2P protocols. In general, hash functions (e.g., SHA-1 [EJ01]) map a wide set of possible input keys to a well defined identifier (ID) space. Hash tables store the corresponding data values along with these IDs. Thus, given a key (e.g., a person's name), the associated value (e.g., that person's telephone number) can be looked up efficiently.
In a DHT, the hash function maps nodes as well as objects to a common m-bit ID space. The ID space must be large enough to map a unique ID for every node and every data item with high probability. Node IDs can, for example, be computed by hashing their IP address, whereas the filename could be the value for shared objects.

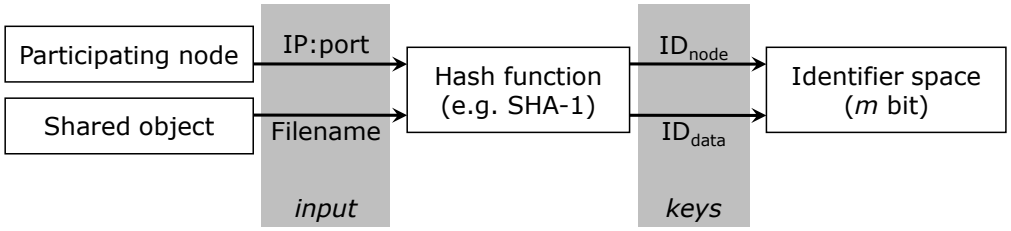
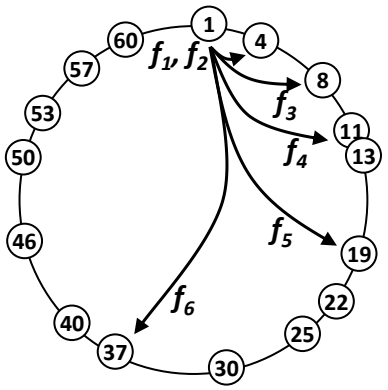


Figure 3.1.: DHT: A central hash function maps nodes and objects to a common ID space.

The network structure is then set up according to the protocol. In a DHT, nodes are positioned in the structure based on their ID. Each participating node accepts responsibility for a well defined part of the ID space. Data (or a link to the node that is hosting the data item) is mapped onto that node whose ID is the "closest" to the data ID (also called key). Thereby, the distance function $\Delta()$ is depending on the protocol, e.g., $\Delta = ID_{\text{node}} - ID_{\text{data}}$ in Chord [SMK+01a] and $\Delta = ID_{\text{node}} \otimes ID_{\text{data}}$ in Kademlia [MM02]. Finally, a routing algorithm provides an efficient method to look up data items stored in the DHT.
Structured P2P protocols specify only a few fundamental operations for users, i.e., *join* the network using a bootstrap node, *insert* and *update* a ⟨key; value⟩-pair, *lookup* a key and retrieve the corresponding value, and *leave* the network. Additional Remote Procedure Calls (RPCs) for maintaining the overlay structure, like *stabilization* or *ping*, are transparent to the user.
DHTs are a variant of consistent hashing [SMK+01a]. By using a consistent hash function it can be guaranteed that w.h.p. all nodes roughly receive the same number of keys, i.e., the load is balanced among all nodes. However, this property can only be guaranteed if IDs are unique, i.e., there is only a single value for each key. Yet, certain applications require storing multiple values per key. For example, in a file-sharing application it makes sense to store references to peers that provide a file instead of storing the file itself. However, several peers might store the same file. Thus, multiple ⟨key; value⟩-pairs are stored for specific keys. Moreover, it is likely that the popularity of the files follows a Zipfian distribution (see Section 7.3.1). Thus, the most frequent file will occur approximately twice as often as the second most frequent file, which occurs twice as often as the fourth most frequent file, etc. Additionally, popular files are requested more often. The Pareto principle or the "80-20 rule" says that 20% of the files are requested in 80% of the queries. Summarizing, a peer responsible for storing a frequent file must store significantly more ⟨key; value⟩-pairs and has to answer considerably more requests than other peers. In Section 3.3 we have a closer look at these problems and we discuss several algorithms that try to balance the load in such scenarios.
The Chord protocol is a very prominent yet simple DHT-based P2P protocol that is referred to in many publications. It establishes a ring-shaped overlay, where the position of a node on this ring is determined by the node ID. Due to its very clear structure the behavior of the protocol can easily be analyzed and evaluated. That is why in the following section the basic features of DHTs are exemplified by using the Chord protocol. In section 6, algorithms that improve the stability and efficiency of structured P2P protocols are also introduced at the example of Chord.



Index i	ID $n_1 + 2^{i-1}$	Node successor(ID)
1	2	n_4
2	3	n_4
3	5	n_8
4	9	n_{11}
5	17	n_{19}
6	33	n_{37}

Figure 3.2.: Chord fingers for node n_1 in a sample overlay network.

3.1.1. Chord

Overlay structure Chord [SMK+01a] establishes a 1-dimensional m-bit ID space. This ID space is wrapped into a ring shape by using modulo operations, i.e., all operations on IDs are performed using modulo 2^m arithmetic. Thus, participating peers are arranged in a circular structure. Each peer is responsible for the ID space between its own ID and the ID of its predecessor on the ring. In other words, a key k is assigned to that peer whose ID is equal to k or is closest following k in clockwise direction[2]. That peer is called successor of key k, and is denoted by successor(k).

Node state and scalable key location Chord establishes a ring structure where each node stores a pointer to its successor on the ring. Using these successor pointers, nodes can lookup any ID or rather the node that is responsible for the ID. However, traversing the ring hop-by-hop is not feasible for large networks, as the average routing path length would be $1/2N$ hops in a network with N participants.
In order to reduce the number of hops, the routing state maintained by each node must be increased. Chord strikes a balance between fast lookups and a relatively small routing state. By cleverly storing $F = O(\log_2 N)$ pointers, which act as shortcuts through the ring, the average lookup path length can be decreased to $1/2 \log_2 N + 1$ hops. In Chord these shortcuts are called *fingers*. Thereby, nodes maintain many fingers to close nodes and some pointers to far away nodes. The first finger is equal to the node's successor. Then, the distance to the next pointers is repeatedly doubled, i.e., the i^{th} finger of node n is the first node that succeeds n by at least 2^{i-1}:

$$f_i := \text{successor}(n + 2^{i-1}) \qquad i \in [1, m] \tag{3.1}$$

In the previous paragraph, we stated that nodes in Chord store F different fingers, although theoretically m fingers exist ($F \ll m$). This indifference is explained by the fact that, w.h.p., the first $\hat{F}$ fingers point to the same node, which we make plausible by the following approximation. The mean distance in the ID space between two nodes in a network with 2^m IDs and $N = 2^F$ is $\frac{2^m}{2^F} = 2^{m-F}$. Assume that the distance between

[2]By *(counter-)clockwise direction* we mean in the order of the ID space, and vice versa.

n and its successor is exactly this value. Thus, the theoretical fingers of n with index i' ($i' \leq m$), which are less or equal than this distance, point to the successor of n ($2^{i'} \leq 2^{m-F}$). Thereby, we deduce that $\hat{F} = m - \log_2 N$. Thus, $F = \log_2 N$ actually different fingers exist besides the successor.
In Chord, lookups for a key k (FIND_SUCCESSOR(k)) are unidirectional. Thus, it is important not to overshoot k. That is why lookups consist of two steps. First, finger entries are used to find the predecessor p of key k (FIND_PREDECESSOR(k)). In the second step, using the successor pointer of p, the successor of k is contacted. Nodes that initiate a lookup for key k, search their list of fingers for the largest entry whose ID is still preceding k (using modulo operations) and send the query to that node. Nodes receiving a lookup check whether they are the predecessor of the ID or not. In the first case they forward the query to their successor, which is also the successor of k, and thus is responsible for k. Otherwise, they also search their routing table for the node with the largest ID that does not exceed k and forward the query to it. Using this routing scheme and the given structure of the fingers, the distance to k is at least halved with every hop. Thereby, lookups can be performed recursively or iteratively. The authors of Chord propose a recursive routing, where the request is forwarded from node to node. In contrast to that, with iterative routing, each node returns information about the next hop to the initiator of the lookup. Thus, the initiator of the query is involved in each hop. At the end of a lookup, the successor of k returns the corresponding value to the initiating peer. A more detailed description of both variants, as well as a hybrid approach, will be given in Sections 6.2.1.1 and 6.2.2.
Ion Stoica et al. prove that using such a scheme, w.h.p., FIND_PREDECESSOR(k) requests will be able to be resolved with at most $2\log_2 N$ hops and $1/2 \log_2 N$ hops on average if all pointers are up to date [SMK+01b]. One additional hop is necessary to forward the request to the successor of k.
Concluding, maintaining correct successor pointers assures correct lookups, whereas fingers allow for short lookup paths. An evaluation of Chord's routing path length in stable and dynamic scenarios is given in Section 5.1.1. In Section 6.2 we present improved routing algorithms, which do not stick to such a completely deterministic finger selection, but offer the possibility to choose fingers based on different criteria (e.g., proximity).

Stabilization One of the most important tasks of structured P2P protocols is to keep up the overlay structure. This is even more important than providing an efficient search, as lookups will only be able to be resolved if routing through the overlay is possible. If the overlay structure is corrupt in one part of the network, any route through that part of the topology will fail.
A viable P2P protocol is characterized by a reliable and efficient search. For structured networks this is only achievable in a highly stable topology. Stability comprises both correctness of the neighbor entries as well as fast handling of topology changes due to joining and leaving nodes. Especially in networks with high churn rates, a fast, reliable and self-organizing stabilization algorithm is indispensable.
Chord uses a very simple stabilization scheme. Each node n stores contact information ($\langle$ID; IP address; port number$\rangle$) of its direct successor s and predecessor p on the ring.

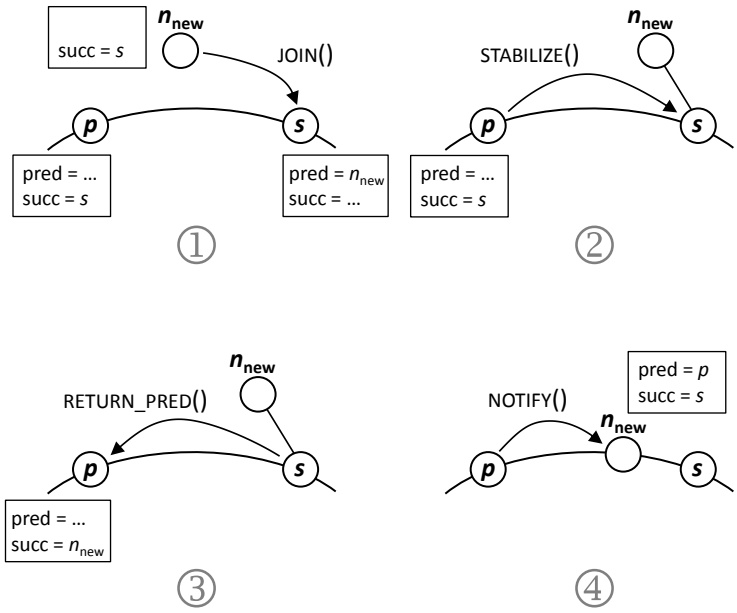


Figure 3.3.: Illustration of a *join* event.

Node n periodically sends a STABILIZATION message to its direct successor s. Receiving this message, node s returns the contact information of its predecessor p. If the overlay has not changed in that local area, the returned node p will be equal to n.
Before a node n_{new} is able to *join* the overlay, it must know at least one node $n_{\text{bootstrap}}$ already participating in the network. This process is called bootstrapping, and several common methods are presented in Section 2.2.2. Node n_{new} asks $n_{\text{bootstrap}}$ to find its immediate successor with $\text{ID}(p) < \text{ID}(n_{\text{new}}) < \text{ID}(s)$. Then, node n_{new} contacts its future successor node s and informs s that it is now participating in the network. However, the former predecessor of peer s, p, does not know about n_{new} yet (see Figure 3.3 ①).
Therefore, p keeps on sending stabilization messages to its successor s (see Figure 3.3 ②). This time, however, node s would return the contact information of node n_{new} ③. Node p stores n_{new} as its new successor and, at the same time, informs n_{new} that it is its predecessor ④. Thus, the new node is fully integrated into the overlay and the ring structure is restored. If several new nodes are attached to a single successor, one of the new nodes will be inserted in the ring in each stabilization period.
In advance of being fully integrated into the overlay, n_{new} is attached to the ring structure like a leafnode as shown in Figures 3.3 ② and 3.4. The authors of Chord call that transitional structure *pseudostar*. In that state, all successor pointers are correct, and thus correct lookups are guaranteed. Chord is able to handle concurrent joins, thus several leafnodes may be attached to a node in the ring.
Consequently, the new node n_{new} either copies an initial set of fingers from its successor or it directly queries for its fingers. Furthermore, node n_{new} starts copying the ⟨key; value⟩-pairs it is now responsible for from its successor. As long as n_{new} is not fully integrated in the ring structure all lookups for these keys are still resolved at node s. In Chord nodes do not handle the references themselves, but rather inform the higher layer software about the changes in topology. The software is then in charge of shifting the references

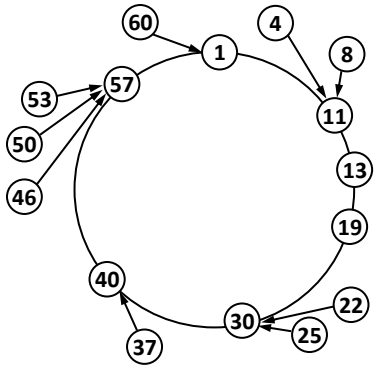


Figure 3.4.: *Pseudostar* formed by joining nodes, which are not yet fully integrated.

to the new nodes. If n_{new} receives a lookup for a key that it is responsible for, but the ⟨key; value⟩-pair is not yet transferred to it from its successor, n_{new} will be able to forward the lookup to its successor that is still able to answer the lookup. Additional replication techniques further help to resolve lookups during the transitional states.

Nodes *leaving* the network inform their neighbors, thereby avoiding any inconsistencies in the structure. In some improved variants of the Chord protocol each node n stores a list of back-pointers to the nodes that have a finger pointing to n (see Section 6.2.3). Thereby, nodes can additionally inform these nodes when they leave network. Thus, lookups are no longer forwarded to n, thereby avoiding timeouts during the lookup.

However, nodes that just *fail*, for example, due to a link break, power cut, or a discharged battery in a mobile device, are not able to send such notifications. Thus, their failure must be detected by their neighbors. In Chord, if no answers are received on successive stabilization messages, nodes will have to assume that their successor has failed. Hereby, the trade-off between fast failure detection, i.e., short timeout values, and additional stabilization overhead due to falsely detected node failures must be considered (see Section 5.2.3).

In order to replace a failed successor, each node maintains a list of several successors. Each node n transmits its successor list $\mathcal{L}$ in its STABILIZATION responses to its predecessor p. Node p adds node n to the front of $\mathcal{L}$, thereby deleting the last element, and replaces its own successor list with $\mathcal{L}$. The authors of Chord state that their protocol can cope with a simultaneous failure of half of the nodes if a list of $|\mathcal{L}| = 2\log_2 N$ successors is maintained.

Beyond, each node must periodically verify its fingers to make sure that all finger table entries are correct. In Chord, every t_{ff} seconds the procedure FIX_FINGERS selects a random finger i and runs FIND_SUCCESSOR$(n + 2^{i-1})$, thus updating this finger.

Concluding, joins and leaves are not very critical to resolving lookups, and the success rate as well as lookup time is hardly increased. In contrast to that, failed nodes must be detected by their neighbors. However, this stabilization mechanism is not able to repair a Chord system that has split into multiple disjoint overlays. In Section 6.1.4 we present solutions to avoid and repair such partitioned overlays.

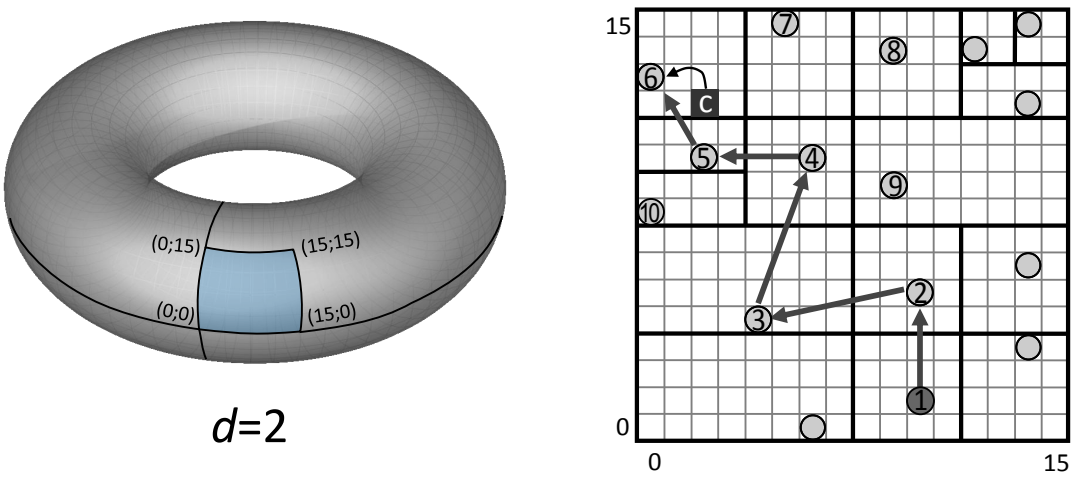


Figure 3.5.: *CAN*: A d-dimensional ID space is partitioned between all nodes.

Summary Using the Chord protocol as example we described the basic functionalities of structured P2P protocols in detail. In order to provide a fast and efficient routing, shortcuts, so-called *fingers*, are established in the ring-shaped overlay. Using these fingers the average lookup length can be reduced to $O(1/2 \log_2 N)$ hops. A stabilization mechanism is performed periodically at each peer to repair changes in the overlay structure.
Chord's stabilization is slow. Each stabilization call can join one node. However, the higher the Churn rate, the higher the probability that more than one node wants to join between two adjacent nodes. Repairing the structure after node failures takes even longer. Also, updating the successor lists requires several stabilization rounds, resulting in erroneous lists. Therefore, it is likely that the next node in the successor list has already failed as well in highly dynamic overlays.
In Chord's stabilization scheme some design parameters can be adjusted: the stabilization period, the timeout value, and the size of the maintained successor list. However, modifying the stabilization algorithm itself is most promising to improve the robustness of the overlay structure. An analysis of the Chord protocol can be found in Section 5.2, and various improvements to the protocol are presented in Chapter 6.

3.1.2. Content Addressable Network (CAN)

Overlay structure A more complex protocol is CAN [RFH+01]. CAN uses a multi-dimensional ID space that is partitioned between the nodes participating in the system. Thereby, lookups can be resolved faster compared with the Chord protocol.
The abbreviation CAN stands for "scalable content-addressable network". A hash function with a d-dimensional output is used to set up a d-dimensional ID space on a d-torus. In CAN, IDs are referred to as coordinates. This coordinate space is partitioned among all nodes participating in the system (see Figure 3.5). Each node is responsible for maintaining its sub-space, called *zone*, and for storing all documents with IDs positioned in this sub-space.

Node state and scalable key location Nodes maintain connections to their immediate neighbors along all dimensions. Node ④ (ID (6; 10)) in Figure 3.5 maintains connections to nodes ③, ⑤, ⑦, ⑨, and ⑩. Nodes ②, ⑥, and ⑧ are no neighbors as they share no common edge with node ④. In contrast to Chord, no finger-like connections to other more distant peers are maintained.
A query for an ID is always routed to that neighbor whose zone is closest to the destination coordinate. In the figure, node ① (ID (10; 1)) initiates a query for a content c (with ID (2; 12)). Node ① sends the query to node ② as its zone is closest to c. In the next hops, the query is forwarded to nodes ③, ④, and ⑤. Finally, the query reaches node ⑥ that is responsible for c. It can be shown that the average lookup path length in a perfectly partitioned coordinate space with z zones (that corresponds to $N = z$ participants) is $(d/4)(z^{1/d})$ hops, and nodes maintain $2d$ neighbors [RFH+01].

Stabilization In the beginning, peers starting to *join* the network choose a random point P in the coordinate space. As in Chord, they must know at least one peer $n_{\text{bootstrap}}$ that is already part of the network. Using $n_{\text{bootstrap}}$ as first hop, a joining peer n_{new} can route a JOIN request to the peer that currently is responsible for the point P. If the current occupant of the zone accepts the request, it will split its zone in half and assigns one half to the n_{new}. Thereby, the coordinates of a peer do not need to be inside the zone it is assigned to. Actually, in many cases both peers have coordinates that are within the same half and one of the peers must be assigned to the other half. After successfully splitting the zone, n_{new} receives the IP addresses and zone ranges of its neighbors from the previous occupant of the zone. Also, in the routing table of the previous occupant, the peers that are no longer neighbors are eliminated and the n_{new} is added. Finally, all neighbors must be informed about the split. Therefore, all peers periodically send a STABILIZATION message with their currently assigned zone ranges to all adjacent peers.
When nodes *leave* the network, they hand their zone to one of their neighbors. Ideally, the zone can be merged with a neighbor's zone so that a valid new zone is created. If not, the zone will be assigned to the neighbor with the smallest zone, and this neighbor must temporarily handle both zones. As in most DHTs, nodes monitor their neighbors with timers. If a stabilization message is received from a neighbor, the timer is reset. If one of the timers expires, the peer must assume that this neighbor has *failed*. Then, all peers adjacent to the failed peer will exchange TAKEOVER messages with each other, in order to find out the best neighbor to take over that zone. A background zone-reassignment algorithm is run to reduce the resulting fragmentation of the space.
Concluding, node joins, leaves and failures only affect adjacent nodes. Hence, the number of messages sent during such operations is not depending on the size of the network. Also, the path length scales as $O(d \cdot (z^{1/d}))$ in a perfectly partitioned coordinate space. Thus, the system is highly scalable in the number of nodes. The parameter d regulates the tradeoff between path length and overhead. On the one hand, if the dimensionality d is set to a small value, the number of neighbors will be small, i.e., the overhead introduced by service messages is low. Also, the routing paths will be long if d is small. On the other hand, increasing d leads to shorter routing paths, but a higher service overhead.

Improvements The authors of CAN suggest many improvements to their basic system design. First, they introduce *realities*, being r multiple, independent coordinate spaces. Each peer joins and maintains a zone in each reality. Content is replicated in each reality. Thus, w.h.p., it is stored at r different peers. Using realities, the average path length can be significantly reduced, as each peer has the latitude to forward a query in that reality where the distance to the destination is smallest. Also, in the case of a link failure, queries can be routed in another reality. Thus, the system is more tolerant to failures. Again, the trade-off between short and failure tolerant routes and the overhead of maintaining multiple realities must be considered. Note that increasing the number of realities r has less influence on the path length than increasing the dimensionality d. However, multiple realities provide improved content availability and fault-tolerance. The concept of multiple realities can easily be applied to other DHTs.
Second, Proximity Route Selection (PRS) (cf. Section 6.2.1.2) is introduced. Hereby, the selection of the next hop does not solely consider the Cartesian distance to the destination, but also takes Round-Trip Times (RTTs) into account. Thus, the average lookup latency was reduced by 25-40% in the authors' simulations. The basic idea of considering multiple parameters for selecting the next hop is generalized in an improvement we present in Section 6.2.4.
Third, the zones are *overloaded*, i.e., several nodes share the same zone. Each peer must know all peers in its own zone, but it still stores only one peer of each of its neighboring zones. Thereby, peers prefer neighbors that have a low RTT (*proximity neighbor selection*, see also Section 6.2.1.2). Again, taking RTTs into account, the average per-hop latency can be significantly reduced. Besides, the path length in terms of number of hops is reduced, as the number of zones is reduced for a fixed number of peers. Also, the system is more fault-tolerant, because a zone will be vacant only if all nodes in the zone crash simultaneously. Content can either be replicated or distributed among all peers in a zone.
Due to its d dimensions, CAN is suitable for a *topologically-sensitive construction* of the overlay. This is essential to avoid *zigzag routes* (see Section 2.2.2). One idea would be to use a 2-dimensional torus, i.e., the surface of a ball. We would be able to map geographic coordinates to that ID space, if each peer knew its geographical position. As this is not feasible, the authors propose a higher dimensional ID space. All nodes are put into that space according to their distance in RTTs to some fixed landmark nodes. If a node measures a short RTT to a landmark, it will be positioned close to that landmark, whereas it will be positioned far from a landmark if the measured RTT is large. A similar technique is used in Global Network Positioning (GNP) [NZ01] (see Section 4.2.1). The rationale behind this scheme is that topologically close nodes in the Internet are likely to have short RTTs. Thus, these nodes are put in the same portion of the coordinate space, and zigzag routes are avoided almost completely, resulting in lower average path latency. However, due to that mapping, the coordinate space is no longer uniformly populated, whereas content is still uniformly assigned to the coordinate space. Therefore, some nodes must handle large zones with a lot of ⟨key; value⟩ pairs. This is why load balancing techniques (see Section 3.3) must be applied to reduce the load on those nodes.

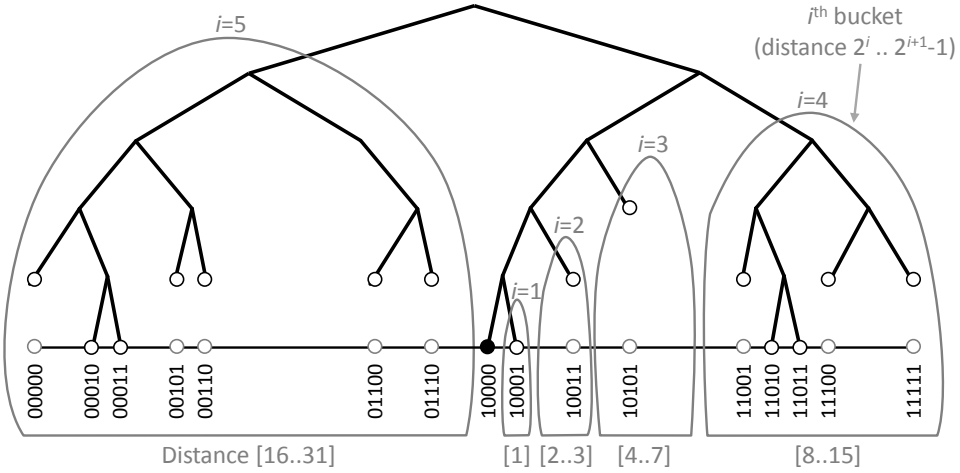


Figure 3.6.: Kademlia sets up a binary tree.

A *more uniform partitioning* of the overlay is achieved with the following proposal. Nodes receiving a JOIN request do not per se split their own zone, but compare the volume of their zone with the volumes of adjacent zones. Then, the largest zone will be split, and one half is handed to the new node. As a result, the partitions are more equal in volume. However, zones are more often handled by peers with coordinates not inside the zone.
In order to achieve an even more uniform partitioning, we propose to forward the JOIN request up to maxhops times. Thereby, each peer compares the volume of its zone with the volume of its neighbors and forwards the request to the largest zone. When a peer is reached that is responsible for a zone larger than the zones of all of its neighbors, it splits its own zone. Using this forwarding scheme, it is more likely that an extremely large zone is reached and split. A similar approach is successfully used to assign leaf-nodes to super-peers in a hierarchical DHT [ZDK07].
Finally, the authors of CAN propose caching and replicating techniques for solving the problem of very popular keys (*"hot spot" management*). Among others, these techniques are discussed in Section 3.2.

3.1.3. Kademlia

Another protocol worth mentioning is Kademlia [MM02]. So far, it is the only structured P2P protocol that is applied in large-scale real-world applications, like BitTorrent [Coh].

Overlay structure Kademlia sets up a binary tree, with the first branch above the root as the most significant bit of the node. As shown in Figure 3.6, the branches of the tree spread out until a single node is at each end of the tree. Thereby, the prefix of the ID of each node corresponds to its position in the tree and the branches may have different lengths. In contrast to Chord, the Kademlia protocol suggests to select random $m = 160$ bit node IDs instead of hashing the IP addresses of the nodes.
In Kademlia, the exclusive or (XOR) function is used as the metric to determine the distance $\Delta()$ between two points in the ID space. The XOR function is applied to

both identifiers and its result is interpreted as an integer distance. Thus, nodes will be considered to be "close" if their position in the binary tree is close.
Using the XOR function as distance metric offers several advantages. First, the function is symmetrical and offers the triangle inequality $\Delta(x,y) \oplus \Delta(y,z) \geq \Delta(x,z)$. More important, XOR is unidirectional; therefore all lookups for the same key converge along the same path. Thus, like in Chord, it is feasible to cache ⟨key; value⟩-pairs along the lookup path in order to reduce or even avoid hot spots.

Node state Kademlia does not divide its pointers to other nodes in neighbors and shortcuts (fingers). Instead, each node n groups its pointers to other nodes according to their distance to n in so called *buckets*. Like in Chord, the sizes of the intervals grow exponentially with the distance to n, i.e., the i^{th} interval stretches from 2^{i-1} to $2^i - 1$, with $1 \leq i \leq 160$.
Thereby, each node may store several pointers in each bucket, whereas in Chord only one deterministic pointer per interval is stored. Each bucket is sorted by Time Last Seen (TLS) and can store up to κ ⟨IP address; port; ID_{node}⟩-triples. The size of the buckets is chosen in such a way that it is very unlikely that all nodes fail within one hour. Following measurements from the Gnutella network [SGG02], the authors of Kademlia suggest using $\kappa = 20$. Similar to Chord for small values of i, the buckets will generally be empty. That is why the authors of Kademlia suggest allocating buckets dynamically as needed.

Scalable key location Kademlia implements an iterative lookup for IDs. The initiator n of the lookup selects the β closest nodes to the queried key k from its buckets and sends β parallel lookup requests to these nodes. The recipients also search their buckets for the β closest peers to k and return the matching triples to node n. The initiator again selects the β closest nodes from all results that have not yet been queried and sends the lookup request to these nodes. This process is repeated until a node is contacted that has stored a value for k, or all nodes close to k have been queried without a query hit.
Thereby, if all buckets are up-to-date, the distance to the target will be *roughly* halved with each step. In contrast to that, in Chord, the distance is *at least* halved in each step and the resulting lookup path length is slightly shorter. However, Chord is limited to using deterministic fingers, whereas Kademlia is more flexible in selecting the nodes that are contacted in the next step. Thus, routes can be selected based on latency. Using parallel queries (see Section 6.2.1.3) timeout delays from failed nodes can be avoided. This results in noticeably shorter lookup delays despite slightly longer lookup paths. In Section 6.2.1.2 an improvement to Chord is presented that allows storing several fingers per finger interval, thereby enabling parallel queries and Proximity Neighbor Selection (PNS).

Stabilization and Bucket refreshes Stabilizing the overlay and refreshing the buckets coheres in Kademlia, as both neighbors and shortcuts are stored in the same logical structure. In contrast to that, neighbors and fingers in Chord are strictly separated. Thus, the frequency of the active stabilization of neighbors and fingers might be tuned

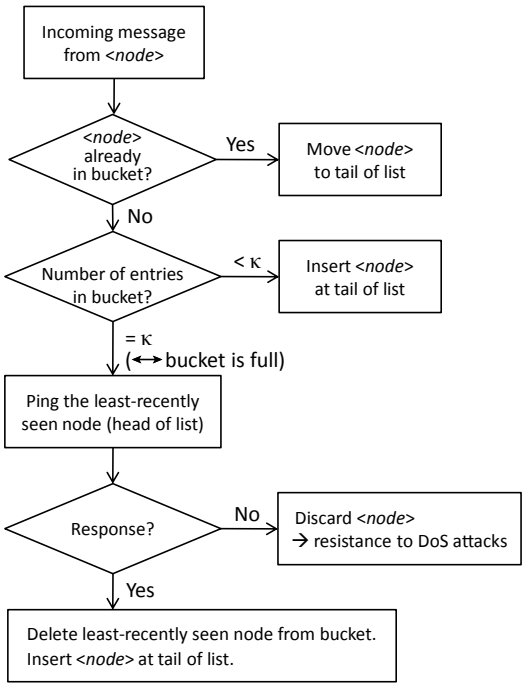


Figure 3.7.: Kademlia's buckets are updated by incoming messages.

independently of each other. Neighbors that assure a correct routing are updated more often than fingers that provide fast routing.
Kademlia refreshes its buckets in a passive manner. That is why no separation of pointers is necessary. Every message exchanged between two nodes includes the node ID of the sender. Thus, the recipient can either reset the TLS of the sender or add the sender to one of its bucket. This is feasible, as routing is symmetrical in Kademlia, and nodes receive lookups from precisely the same distribution of nodes contained in their own routing tables. Moreover, due to the iterative routing process nodes learn about various live nodes from the whole key space. In contrast to that, nodes in Chord receive only feedback from their fingers due to the recursive routing. Also, neighbors in Chord are actively verified by the stabilization algorithm.
If one of the buckets of node n is not refreshed by receiving a message for one hour, node n will pick a random ID in the range of the bucket and perform a lookup for that ID, thereby refreshing the bucket. If a node n_1 sends a RPC to another node n_2, but no response is received from n_2, for example, due to packet loss or the failure of n_2, the lookup will not be affected, as it is unlikely that all β parallel requests are not answered. However, if n_2 fails to respond to 5 RPCs in a row, node n_2 will be considered as stale.
Figure 3.7 depicts Kademlia's bucket refresh algorithm: New live nodes replace the least-recently seen entry in the corresponding bucket. However, live nodes are never removed from the buckets as measurements showed that the probability of a node remaining online for another hour increases with the current uptime of that node [SGG03, Sch05]. Additionally, this policy provides a certain resistance to Denial-of-Service (DoS) attacks. An attacker flooding the system with new nodes is not able to flush the routing states of the nodes as existing live entries are not removed from the buckets.

Improvements Similar to other publications the authors of Kademlia propose several improvements to their basic routing algorithm. First, the efficiency of the lookup might be increased by introducing a replacement cache for full buckets. If a node learns about another node, but the corresponding bucket is full, the new node will be stored in the cache. The cache will be used to immediately fill up the bucket if another node in that bucket is removed. Thereby, the least recently seen entry of the cache is moved to the bucket.
Second, the absence of an answer to a message must not imply a failed node. Instead, the UDP packet could either be lost or the message is congested. That is why, similarly to TCP, this contact should not be used for exponentially increasing backoff intervals. This introduces no significant restraint, as Kademlia is flexible in selecting the lookup paths. Thus, the impact of packet loss could be reduced. If there is still no answer after 5 RPCs, the contact will be considered as stale. However, the reason for the failure could be a temporarily failure of the own network connection. In that case, valuable routing information might be lost if the node would blindly remove all of its pointers as no answers are received from them. Thus, pointers should merely be flagged stale if the replacement cache is empty. Then, up to κ entries per bucket are preserved and might be re-used if the connection is restored.
Finally, the authors suggest to piggy-back configuration information as well as ping request on other messages, thus reducing the overhead of the protocol.

3.1.4. OneHop

Another class of P2P protocols tries to resolve lookups with a single hop. Therefore, each node must store the complete system membership. This requirement is no problem for nowadays personal computers. Assuming a size of 50 Byte per entry in the routing table (160 bit ID, 128 bit IPv6 address, 16 bit port number, 64 bit TLS counter, ...), the total routing state in a network with 1 million nodes is around 50 MB.
However, all changes in the overlay must be broadcasted to all nodes in system. The main task of these protocols is to minimize the increased communication costs. Referring to a study of Gnutella and Napster [SGG02], we assume a mean session time of $\mathrm{E}[T_{\mathrm{on}}] = 1$ h. Thus, a system with 10^5 nodes shows around $2 \cdot 10^5$ membership changes per hour, in other words, a churn rate of 2 events per peer and hour.
Similar to other P2P protocols, adjacent nodes monitor each other by sending periodic keep-alive messages. If no message is received from a neighbor within a time-out period, it will be assumed that the neighbor has failed and take appropriate actions. In a one hop protocol, the node must inform all other live nodes by sending a broadcast.

Overlay structure and scalable key location One of the protocols, which apply such a routing strategy, is OneHop [GLR04]. In order to reduce the overhead introduced by the broadcast, it uses a hierarchical approach to forward a message to each node in the system. OneHop is similar to Chord. Nodes are arranged in a 1-dimensional ring-shaped ID space with the successor of a key k being responsible for k. OneHop also adopts the concept of neighbors and fingers. Neighbors are required to precisely determine which interval a node is responsible for, whereas fingers provide fast lookups. Storing

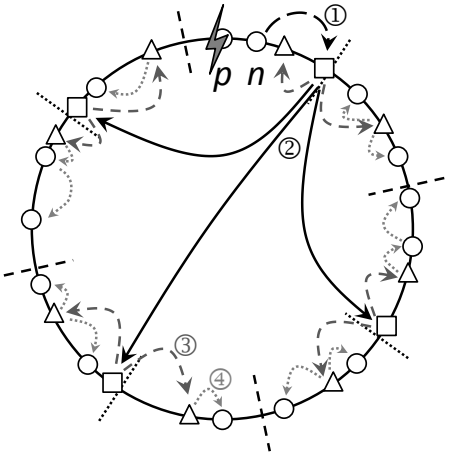


Figure 3.8.: OneHop applies a hierarchical event propagation scheme.

the complete system membership is comparable with maintaining a finger pointer to all live nodes. Assuming that all pointers are up-to-date, all queries can be resolved on the first attempt. Otherwise, queries can still be rerouted as long as the neighbor pointers are correct. Thus, maintaining local information is more important than updating finger entries. Yet, a fast finger update is necessary for complying with the intention of a one hop query resolution.

Stabilization OneHop tries to broadcast membership changes in a hierarchical way that has low delay, yet still reasonable bandwidth. Therefore, the ring is divided in intervals of equal size referred to as *slices*, and each slice is subdivided in several intervals, called *units*. We use the symbol u for the total number of units.
The logical mid-point of the successor of a slice or unit is selected as slice-leader ($\square$) or unit-leader ($\triangle$), respectively. Due to the deterministic partitioning of the ID space and the deterministic selection of interval leaders, each node is able to estimate its corresponding slice and unit leader.
The way events are propagated with OneHop is shown in Figure 3.8. The interval borders of slices and units are marked by dashed and dotted lines, respectively. In the figure node n_0 fails, and its failure is detected by its neighbor node n_1. Then, node n_1 initiates a broadcast by notifying the corresponding slice leader ①. In order to reduce the amount of messages, slices leaders aggregate events from their own slice for some seconds before reporting the changes to all other slice leaders ②. Slice leaders also aggregate incoming messages from other slices for some seconds before forwarding the aggregated message to all unit leaders in their slice ③. Finally, the message is piggybacked on keep-alive messages starting from the unit leaders to the boundaries of the units ④.
However, an additional stabilization mechanism must be implemented in order to detect errors in the membership tables due to lost event notifications. The accumulation of such errors would lead to a steady degradation of the OneHop lookup success rate. Thus, if any node detects a wrong entry during any communication attempt, it will initiate a standard propagation of that change, as described above.

Improvements In the bootstrapping process, a new node copies the current system membership information from any other node. However, in large networks the size of this information is several megabytes, thus the download takes several minutes for nodes with low bandwidth (e.g., mobile nodes). In order to be able to immediately participate in the network, we suggest using another node as a relay in the meantime.
The authors of OneHop also suggest two improvements to their protocol. First, they try to prevent nodes with low resources from becoming slice or unit leaders, as nodes in higher levels must be capable of handling more traffic than other nodes. Therefore, they suggest keeping supernodes in an extra ring and selecting slice (and unit) leaders from that subset of most capable nodes. However, by doing so additional overhead for maintaining the second ring of supernodes is required.
Second, they introduce a two hop lookup, where only a fraction of the total routing state is stored at each node. Nodes maintain information about all nodes in the same slice, but keep only one finger to any other slice. Lookups for keys in the same slice can still be answered by one hop. Lookups for other keys are sent to the appropriate slice using the corresponding finger entry. This node is aware of all nodes in its slice and can forward the request to the successor of the key. Therefore, lookups will be resolved in at most two hops if all pointers are up-to-date.

Summary Events are aggregated and broadcasted along a three-level hierarchy, thereby striking a balance between large delays in propagation and large load at the nodes in the utmost level. Also, there is no redundancy in communications, as the hierarchy provides a well-defined dissemination tree. Yet, an additional stabilization is required to repair erroneous membership information.
The authors show that the OneHop protocol works well in failure-free scenarios, i.e., there is no packet loss, no message delay and no failure of slice and unit leaders. However, no analysis of the influence of high churn rates or lost broadcast messages is provided. We assume that the protocol is not suited for such scenarios. Thus, we recommend only applying such a one hop scheme in scenarios with largely static nodes. The scheme might also be applied to hierarchical protocols, with the most capable and static nodes running the OneHop protocol and other nodes being attached to these nodes as leafnodes.
Another one hop routing protocol D1HT is presented in [MA06]. Its main difference to OneHop is the way events are propagated. Messages are distributed by recursively splitting the ID space in several intervals and forwarding the event to the first node in each interval (see Figure 3.9). The recursion is repeated until each node received a message. This procedure is similar to the way a snapshot of the system may be created [BKH07]. The authors of D1HT state that the bandwidth requirement of D1HT nodes is less than OneHop's slice/units leaders and ordinary nodes, and that their protocol can be applied to P2P systems with reasonable churn rates.

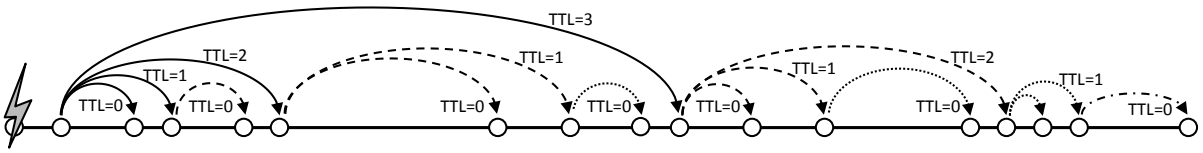


Figure 3.9.: D1HT applies a recursive event propagation.

3.1.5. Pastry, Tapestry

Pastry Pastry [RD01] arranges participating peers on a circular m-bit ID space with base 2^b, i.e., an ID consists of $\frac{m}{b}$ *digits*. Nodes maintain three different kinds of neighbors. The *routing table* is a matrix with $\lfloor \frac{m}{b} \rfloor$ rows and $2^b - 1$ columns. The node in cell (r, c) shares the first r digits with the local node, but differs in the remaining digits. The routing table is structured as a Plaxton tree. A lookup for a key k is forwarded to the node with the longest matching prefix with k. Thereby, the table is used to traverse large hops to peers with distant IDs, equivalent to fingers in Chord. Neighbors with small physical delay are preferred in the routing tables, thus reducing the overall lookup latency (see PNS in Section 6.2.1.2).
The *leaf set* is a list of predecessors and successors on the circular ID space, and is used to route a message to the correct node in the final short hops of the lookup. Finally, the *neighborhood set* is a collection of the physically closest peers. However, this set is usually not used for routing. Using the routing tables and leaf sets, the average path length of Pastry is in $O(\log_b N)$.

Tapestry Tapestry [ZHS+04] is very similar to Pastry, and differs mainly in its mapping of keys to nodes. Here the node with the maximum number of matching prefix digits with a key is responsible for the key. Furthermore, each entry in the routing table may contain several nodes. The entry with the smallest network delay is used for routing (primary neighbor); other entries are stored as backup. The primary neighbor is pinged every t_{stab} seconds. If several successive pings fail it will be replaced by the next entry in the list. A *nearest neighbor* algorithm is used to learn about new neighbors with short latencies (see PNS in Section 6.2.1.2). Moreover, Tapestry does not maintain a leaf set and neighborhood set. This is why only a Plaxton tree, but no ring structure is maintained. The expected number of routing hops in Tapestry is $\log_{2^b} N$.

3.2. Replication, Republishing, and Caching

In DHTs, ⟨key; value⟩-pairs are stored at deterministic nodes. Lookups for keys are also routed toward these nodes. However, due to node failures and new nodes joining the overlay pairs get lost or responsibilities change, respectively. Consequently, the ⟨key; value⟩-pair is still stored in the DHT, but lookups for that key terminate at the node currently responsible for the key, making additional algorithms necessary for increasing the content availability. Most of these algorithms are implemented on top of the DHT, i.e., by the application itself. We distinguish between three different types of algorithms: replication, republish and caching.

Replication techniques store multiple copies of the ⟨key; value⟩-pair. Thus, the probability is increased that any copy of a pair is accessed although responsibilities for the key k change. Assume a Chord network where node n is responsible for k and node s is the successor of node n. If n fails, node s will be the new successor of k. That is why the authors of Chord suggest exploiting the successor list $\mathcal{L}$ and replicating content on R peers succeeding k ($R \leq L$). We call those peers *replication group* $\mathcal{R}$ of k. They also state, that at least one replica is available in the case of a simultaneous failure of half of the nodes if $R \geq O(\log_2 N)$. Additionally, Chord nodes keep track of their successors. Thus, they may inform the higher layer application that changes occurred and new replicas should be propagated. Similarly, other DHTs might store replicas on adjacent peers.
We suggest another variant of replication where nodes in $\mathcal{R}$ are responsible for updating the replicas. Two situations may occur. First, one node in $\mathcal{R}$ fails and the node succeeding $\mathcal{R}$ becomes part of the group. Then, this peer must get a replica from any other peer in $\mathcal{R}$. Second, a new node n_1 joins and becomes part of $\mathcal{R}$. Then, the last node in $\mathcal{R}$ (n_2) is no longer part of $\mathcal{R}$. In this situation, either node n_1 tries to obtain a replica from any node in $\mathcal{R}$, or n_2 moves its replica to n_1. In this context, each node should be aware of all replication groups it is a member of. Storing a symmetrical list of neighbors (see Section 6.1.2) would provide Chord nodes with enough information to determine all nodes in the groups. As the information about close neighbors is more accurate, we recommend to set L noticeably larger than R, e.g., $L = 2R - 1$ as we did in our simulations.
Other replication algorithms store copies on other deterministic peers. The peers can be determined by, e.g., using R different hash functions (see Section 3.1.2) or by adding integer values of a function $f(i)$ ($i \in [1..R]$) to the key k. A simple function $f(i)$ should be selected to reduce computational overhead, e.g., $f(i) = c \cdot (i - 1)$. The constant c could be any integer. Setting $c = {}^{N}/_{R}$ evenly distributes replicas on the ID space.
Replicating data on multiple nodes provides additional benefits. The replicas might be queried in parallel, thus the content availability as well as the mean search duration are improved. If large values are stored in the DHT, *erasure coding* will help to reduce the overhead introduced by replication. Hereby, content is coded into R fragments and each fragment is stored at one node of the replication group. Finally, any R' ($R' < R$) fragments are sufficient to recover the original value.
In Kademlia, ⟨key; value⟩-pairs are replicated among the R closest nodes to the key, with $R = b$, and b being the size of Kademlia's buckets. Thereby, monitoring the replication group is more complex than in Chord. That is why Kademlia implements a more bandwidth consuming update. Nodes in $\mathcal{R}$ periodically *republish* the corresponding pair to the R closest nodes of the key to ensure high content availability. However, if a node in $\mathcal{R}$ receives such a pair, it will skip its own republish event as other nodes in $\mathcal{R}$ are also expected to have received this message. When publishing a ⟨key; value⟩-pair, a timestamp is assigned to it. If the timestamp is older than a specified time (e.g., 24 h) the pair will be deleted to limit stale index information in the DHT. Thus, the content provider must additionally *republish* the pair before its expiration in order to keep it alive.
Caching also improves the content availability. However, its main purposes are reducing the load on nodes that are responsible for a popular key, and reducing the search delay

for common keys. In most proposals, content is cached along the lookup path. Kademlia implements a different solution. After each successful lookup, the requesting peer stores the ⟨key; value⟩-pair at the last hop of the lookup path, which did not return the value. As lookup paths converge towards the end of the lookup, future queries are likely to hit cached entries. The authors of Kademlia suggest setting the expiration time of cached entries exponentially inversely proportional to the number of hops between the caching node and the node whose ID is closest to the key.
In Chord we can exploit the fact that nodes know their neighbors on the ring. Thus, content may be actively cached on nodes that are *preceding* a key k. These nodes are likely to be hit in queries and the successor of k is released from the queries. Instead of maintaining R replicas on succeeding nodes and caching the pairs on C preceding nodes, we suggest to keep $C \geq R$ replicas on preceding nodes only.

3.3. Load balancing

Load balancing algorithms try to balance storage as well as traffic load more evenly among the participating nodes.

Zipf-like distributed keywords Zipf formulated an empirical law that was originally observed in a linguistic context [Zip32]. It states that a few occurrences are very common, whereas a large number of instances are extremely rare. If items are ordered from most popular to least popular, the position in this list will be called the *rank* of the occurrence. Zipf's law also states that the popularity of an item tends to be inversely proportional to its rank i, i.e., $P(i) = ci^{-1}$ for some positive constant c. This kind of distribution is also called power-law or Pareto, yet it will be referred to as Zipf if we plot the occurrence's rank against its frequency [Ada00].
Researchers found Zipf-like distributions in many kinds of phenomena, amongst others, in search keywords in P2P networks [KLVW04]. Also, the popularity distribution for files on Web servers has been shown to commonly follow Zipf's law, with some files being extremely popular while most files receiving relatively few requests [BC98]. Furthermore, last and first names, as well as town and street names, follow a Zipf-like distribution. Thereby, we observe that ranks and common keywords do not change significantly over time. The most popular first names, for example, change within a few years, whereas the most common last names hardly change at all.
In order to obtain realistic input for our simulations, we extracted data from a German phone directory available on CD [Top]. It contains about 38 million entries consisting of seven attributes: name, street, zip code, city, type (e.g., cellular phone, fax), prefix, and telephone number. We separated names into first name and last name using the first blank as separator. Afterward, we removed incomplete entries resulting in about 27.5 million reasonable entries.
We start with an evaluation of the frequency distribution of the keywords. Figure 3.10 confirms that German first names, last names, and town names follow a Zipf-like distribution. Other researchers found similar results for other phone directories (e.g. [FJ92]).

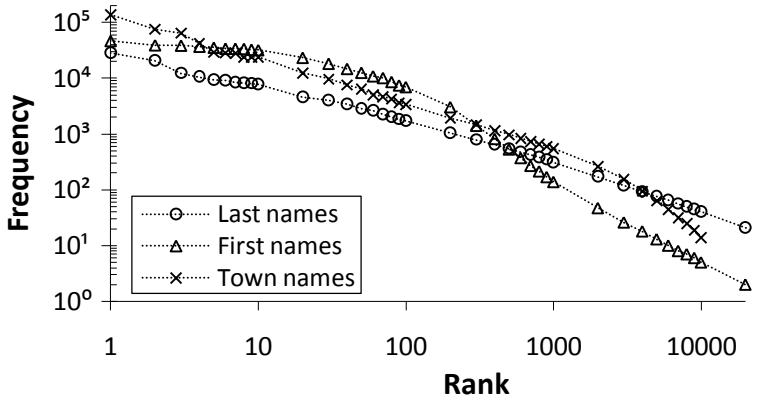


Figure 3.10.: Rank-frequency plot for German names and sample IDs.

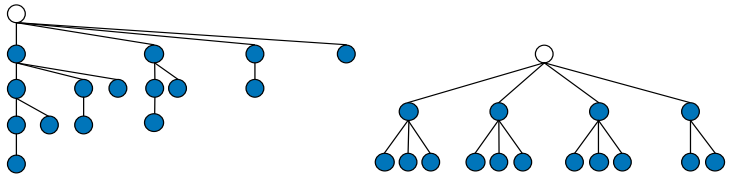


Figure 3.11.: Binomial lookup tree (left) and balanced lookup tree (right)

Load Balancing In the context of load balancing we need to consider two aspects: storage load and traffic load. Without any load balancing mechanisms some nodes in a DHT with N nodes and K keys will be responsible for up to $(1+\epsilon)\frac{K}{N}$ keys ($\epsilon = O(\log_2 N)$) [KLL+97, SMK+01a], even if the keys are distributed uniformly in the ID space.

However, the more nodes participate in the network, the smoother the distribution of keys over all nodes will be. Therefore, the probability that a node has to store a noticeable larger percentage of content or content descriptions is reduced. This is important, as being responsible for more keys implies having to resolve more lookups. In order to increase the number of nodes in the system, many load balancing approaches introduce *virtual nodes*. Virtual nodes are parallel and independent instances of the protocol running on a single machine. Thereby, the imbalance mentioned above will be able to be reduced to almost $\frac{K}{N}$ keys per node (with $\epsilon \to 0$) if each physical node runs $O(\log_2 N)$ virtual nodes.

A non-uniform distribution of keys due to the use of locality-preserving hash functions increases the need for such algorithms. In particular, mapping keywords linearly from an ASCII or Unicode character set to the ID space leads to vast gaps since keywords like names in a telephone book usually do not start with special characters. Caching ⟨key; value⟩-pairs along the lookup path might alleviate such hot spots on single nodes. Amongst others, [ZH05, RLS+03, RPW04, KR04] propose efficient *storage load balancing* mechanisms.

However, many approaches neglect the resulting *traffic load* distribution. Protocols like Chord, which are based on a binomial lookup tree, have an intrinsic imbalance caused by their recursive structure [CHHC06]. Figure 3.11 visualizes the difference between balanced and binomial lookup trees. The root of the tree is the node n that is looked up

and the leaves are all other nodes that could initiate a query for a key stored on node n. Each edge between two levels represents one routing hop, i.e., the longer a branch the more hops a query has to take. In an ideal balanced lookup tree, all branches have the same height and therefore all nodes on a certain level receive the same number of lookups.
In contrast to this, some lookups in Chord reach the key with a single hop, whereas other lookups need to travel up to $O(\log_2 N)$ hops. Also, some nodes in level 1 will be passed by many routing paths whereas others are not passed by queries. Regarding the lookup trees for all nodes at the same time, each node will be frequently used on the lookup path of some queries as long as all nodes are distributed uniformly in the ID space. In this case, the overall lookup traffic is equally distributed to all nodes. On the contrary, if nodes are shifted to ID ranges with a high key concentration, this overall balance will no longer be given. Fingers in Chord point to nodes in exponentially increasing distances. As a consequence, fingers will more often point to nodes responsible for a huge (but almost empty) range of identifiers, than to nodes hosting a small part of the ID space. More fingers pointing to a node equals being part of more lookup paths, resulting in higher traffic load.
The solution is constructing balanced lookup trees to avoid such hotspots. Examples are the DHT protocol SCALLOP [CHHC06] or the non-DHT approach Skip Graphs [AS03].

3.4. Non-DHT protocols

Finally, we present structured P2P protocols not based on DHTs.

3.4.1. Skip Graphs

Skip Graphs are based on distributed skip lists. The main advantages compared to DHTs are, that they establish a balanced lookup tree and give tree functionality instead of only hash table functionality.
A *skip list* [Pug90] is a data structure that is based on linked lists with probabilistic shortcuts. The lowest L_0 is a simple sorted linked list of all elements. Each element in a certain level L_i appears in the next higher level L_{i+1} with a predefined probability p, thus creating a randomized balanced tree (see Figure 3.12). Thereby, lists at higher levels act as shortcuts, as they *skip* parts of underlying lists.
All searches for a key k start at the root element of the tree. Each level is traversed horizontally until the last element in it being less or equal to the target k has been reached. If the element is equal to k, the sought-after element has been found. If the last element that is less than the target (PREDECESSOR(K)) is found, the search will be handed over to the next lower list, and the procedure will be repeated. On average, there are $\log_{1/p} N$ lists, $1/p$ hops in each linked list, and a total number of $1/p \cdot \log_{1/p} N$ hops to resolve a query.
Due to this routing strategy, skip lists establish a balanced lookup tree (Figure 3.11 (right)), resulting in a more evenly distributed load. However, the top elements in this data structure are hotspots and Single Point of Failures (SPoFs), because they are

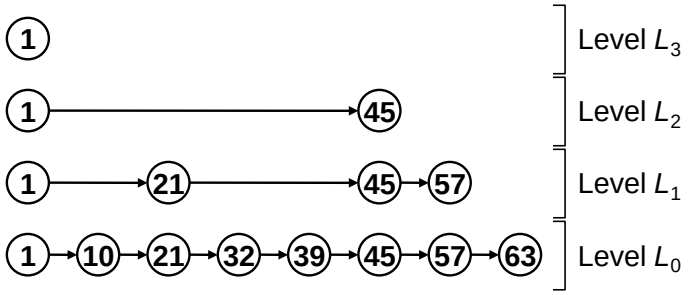


Figure 3.12.: A *skip list* is a linked list with probabilistic shortcuts.

traversed in almost all lookups. Also, the structure might easily break apart if multiple nodes fail. Therefore, skip lists by themselves are not suited for a distributed system.
Skip Graphs [AS03] set up multiple skip lists with each node participating in a list in all levels. Therefore, each node is assigned with a membership vector $m(x)$. The lowest level L_0 is a doubly-linked list of all elements. In higher levels L_i all nodes that have vectors with a common prefix of i digits belong to the same doubly-linked list. In Figure 3.13, a sample skip graph with 8 nodes is shown. There are two lists in level L_1 with prefixes 0 and 1, and four lists in level L_2 with prefixes 00, 01, 10 and 11. Node 10 is the head element of the skip list encircled with the dashed line (prefix 00), nodes 21 and 57 are the head elements of the skip list marked with the dotted shading (prefix 10), and the other two skip lists are not marked.

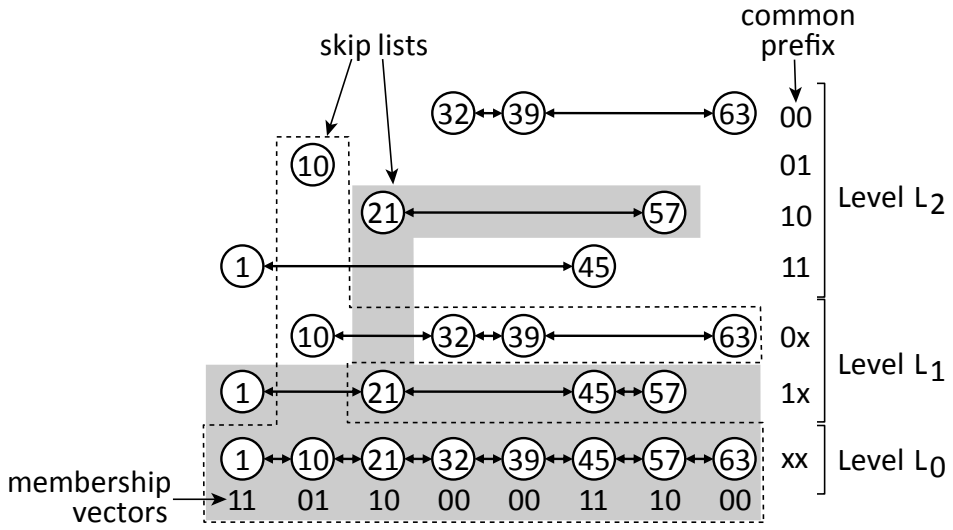


Figure 3.13.: *Skip graphs* set up multiple skip lists.

Searching skip graphs is similar to searching skip lists with the same average lookup path length of $O(\log_2 N)$. The doubly-linked lists allow for horizontally traversing the lists in both directions. On average, each node must store $O(\log_2 N)$ neighbors. By maintaining multiple lists at each level, skip graphs can handle a simultaneous failure of an $O(\frac{1}{\log_2 N})$ fraction of the nodes in the system. Moreover, skip graphs eliminate the hotspot and SPoF of skip lists, as there are many nodes in all levels.

Skip graphs also support complex queries such as range queries. Therefore, the first node in the range is searched. The query is consequently passed from node to node in level L_0 until the last node in the range is reached.
Comparable to other structured overlay networks, nodes join and leave a skip graph with $O(\log_2 N)$ messages. Thereby, several constraints must be satisfied that define the order of the elements in the lists and how the lists at different levels are related to each other. Additionally, a very complex repair algorithm is necessary that heals disruptions due to failed peers and thus prevents accumulating errors. The authors state that skip graphs are highly resilient and tolerate a large fraction of failed nodes, but they also admit that the current repair mechanisms are not very efficient and may not repair a defective skip graph. However, with uncorrelated and independent node failures, Skip Graphs behave comparably to other peer-to-peer systems.

3.4.2. SkipNet

Another non-DHT protocol is SkipNet [HJS+03]. Fundamentally, SkipNet applies the same basic data structure as Skip Graphs, i.e., distributed skip lists (with $p = 0.5$); however, lists are singly-linked and circular. Thus, each node in SkipNet has approximately $2 \log_2 N$ neighbors. Using the neighbor at level i, roughly 2^i nodes may be skipped on average, resulting in a lookup path length of $O(\log_2 N)$.
In SkipNet the authors emphasize content and path locality that is not feasible in pure DHTs. Like in DHTs, hashes of node names (e.g., IP address or host name) and content identifiers are mapped to the numeric ID space. Moreover, SkipNet employs an additional string name ID space, where node names and content identifier strings are directly mapped into. Thereby, content locality can be achieved by using the node name as prefix for content names. As an example, to store a document `ABC.txt` on the node `gerald.tum.de`, naming it `de.tum.gerald/ABC.txt` is sufficient. Note that by reversing the host name, path locality is given for all nodes within an organization (here `tum.de`). As a result, SkipNet will "gracefully" partition into two fully functional segments when one organization loses connectivity to the rest of the network. Furthermore, using the string name ID space, range queries are supported.

3.5. Conclusion

In this section we presented various structured P2P protocols. These protocols specify the respective overlay structure, its maintenance, and methods for efficient key location. Thereby, the stabilization of the structure is most crucial, as only valid neighbor information assures that content is stored and queried at the correct node. In addition, providing short routes (in terms of number of hops and transmission time) is very important, as finding the responsible peer for a certain key is required in several algorithms. These algorithms include inserting, searching, and replicating content, as well as joining new nodes, updating fingers, and repairing disrupted structures.
Table 3.1 summarizes the presented structured P2P protocols. Although the protocols apply various different overlay structures, like rings or trees, they perform similarly.

Most protocols are able to lookup keys in $O(\log N)$ and peers maintain a routing state of around $O(\log N)$. Also, the number of signaling messages required for node joins and correct leaves scales with the number of live nodes N. Protocols like OneHop are an exception to that rule. They trade scalable routing tables against a complete routing state, thus being able to resolve lookups in 1-2 hops.
Another difference is the used distance metric. Protocols like Chord and CAN forward a message for key k to the node n with the smallest Euclidean distance Δ to key k ($\Delta = k - n$). In contrast to that, protocols like Kademlia and Pastry establish routes that diminish the Hamming distance, i.e., the number of ones in $k \oplus n$. Anyhow, they show similar routing performance. Further comparisons of structured P2P protocols can be found in [LCP+05, Li06, RM06, MKL+02, EA05].
In the following, we will concentrate on Chord, as its simple ring structure can be used to intuitively explain the basic functionalities of the presented algorithms and improvements. Moreover, the ring geometry is highly flexible and its performance in terms of lookup path length, search success, and signaling overhead is similar to other geometries. The simple and clear structure, in particular, shows excellent resilience. In their research on the impact of DHT routing geometry on resilience and proximity, the authors of [GGG+03] revise their initial inclination to favor more complicate structures and ask, "Why not use ring geometries?" Nonetheless, additional improvements like, proximity based routing and improved stabilization mechanisms, should be applied to the basic protocols. We present related work and introduce novel maintenance and lookup algorithms in Chapter 6. Note that all results and improvements presented in this thesis are compatible with the Chord protocol. However, most results are also valid for DHTs in general and many algorithms may easily be translated to other overlay protocols.

Protocol	**Chord**	**CAN**	**Kademlia**	**Pastry, Tapestry**	**OneHop**	**SkipNet, SkipGraphs**
Structure	1-dimensional circular ID space	d-dimensional ID space	Binary tree	Plaxton-style global mesh network	Fully meshed ring	Skip graphs/ lists
Mapping	Successor of key k	Owner of zone containing k	Node with closest ID (XOR metric)	Node with numerically closest ID	Successor of key k	Successor of key k
Distance	Clockwise numeric distance on the ring	Number of bits in which the IDs differ	Numeric value of the XOR of the IDs	Height of smallest common subtree	Numeric distance	Numeric distance
System parameters	N	N, d	N, bucket size κ	N, base b of chosen identifier	N, total number of units u	N, p
Node state	$O(\log_2 N)$ neighbors + m fingers	$2d$	$O(\log_2 N)$	$b \log_b N$ $(+b)$	N	$O(\log_{1/p} N)$
Mean lookup path length	$1/2 \log_2 N + 1$	$\frac{d}{4} N^{1/d}$	$O(\log_2 N)$	$O(\log_b N)$	1	$1/p \log_{1/p} N$
Peers join/leave	$(\log_2 N)^2$	$2d$	$O(\log_2 N)$	$\log N$	$\frac{N}{u}$	$O(\log_{1/p} N)$
Stabilization	Periodically ping successor, active finger update	Send keep-alive, split and takeover zones	Exploit existing traffic, active refresh as fall-back	Periodically ping neighbors	Hierarchical broadcast	Periodically ping neighbors, check constraints

Table 3.1.: Comparison of structured P2P protocols

CHAPTER 4

Simulation models and environment

Simulation, emulation and analytical approaches are the main methods in the process of developing and benchmarking new networking protocols and applications. A purely analytical approach is often not feasible, because the applied models are very complex, the scenarios are too large, the behavior is influenced by many parameters, and many random events are decisive issues. Especially in P2P applications, where an arbitrary large number of users may participate in the network, simulation is required to evaluate the performance and scalability of the protocol and to observe its correct behavior.
Simulations are also used to evaluate the behavior of an application in uncommon or even undesirable situations, like the functionality of an ad-hoc network which is used for coordinating an action force after a natural disaster or the simultaneous failure of many peers participating in a P2P network. We believe that discrete event simulation is a powerful tool to gain insight into complex processes at the desired level of abstraction. In order to achieve realistic results, the models applied to the simulation must reflect the real world as close as possible. However, the greater the level of detail, the more complex and resource consuming the simulation gets. Due to limited processing power, memory and available time, sensible spending of limited resources has to be performed.
This chapter investigates different possibilities for modeling user behavior, followed by a detailed look at the network layer. We compare the most commonly-used network models and present a very efficient model for applying real-world network transmission times in large scale simulations. Finally, we introduce our simulation environment consisting of a traffic generator, the actual simulator, and a Graphical User Interface (GUI).

4.1. Modeling the user behavior

The lifetime of a node consists of one or more sessions, in which the node is participating in the network. Each session can be divided into active parts, where searches are performed, and passive parts. A session starts with a join event, and ends either with a

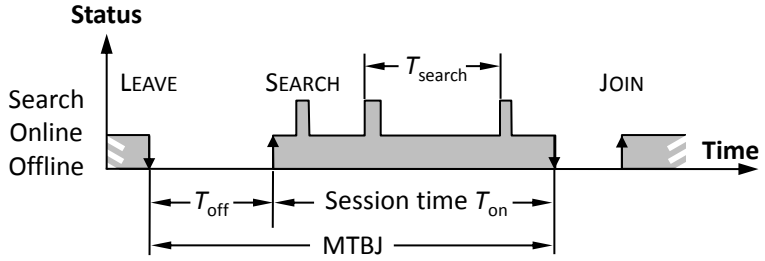


Figure 4.1.: A node's lifetime consists of one or more sessions.

graceful leave or a node failure (see Figure 4.1). After nodes have finished their session they will stay offline for a certain period before joining the overlay again.
To be able to provide an efficient proactive routing, all routing tables have to be updated regularly. Each change in the overlay topology leads to erroneous entries in fixed routing tables. A node that joins the network, for example, has to be announced to all of its new neighbors in the Chord ring. Additionally, finger tables might provide a more efficient routing if the new node was inserted into them. Nodes leaving the network have to send notification messages to all of their neighbors, whereas failed peers have to be detected by their former neighbors, which in turn have to make sure, that all outdated references are removed. Each node causes its new neighbors on the ring to update their successor and predecessor lists when it joins the network, as well when it leaves or fails a certain time later. Thus, in each session a node sets off two events that change the overlay topology.
Churn is defined as the number of changes within a certain unit of time, e.g., 1 h. Usually, it is modeled by a mean session time $\mathrm{E}[T_{\mathrm{on}}]$ plus either a join rate λ or a mean offline time $\mathrm{E}[T_{\mathrm{off}}]$. Thereby, T_{on} and T_{off} are random variables that describe the duration of the online and offline periods, respectively. In the following paragraphs we present and discuss both models.

Session time $\mathrm{E}[T_{\mathrm{on}}]$ and Arrival rate λ In the first model, a global join (arrival) rate λ and a global leave (death) rate μ are defined. These rates are usually modeled by Poisson processes, and express the average number of arrivals and leaves during a unit of time, or:

$$\text{join rate } r_{\mathrm{join}} = \frac{\text{number of joins}}{\text{time}}. \tag{4.1}$$

Then, according to Little's Law, the long-term average size of the overlay N is equal to the long-term average join rate λ multiplied by the long-term average session duration $\mathrm{E}[T_{\mathrm{on}}]$, or:

$$N = \lambda \cdot \mathrm{E}[T_{\mathrm{on}}]. \tag{4.2}$$

The network size N can either be risen by increasing the join rate λ or by extending the users' session times. Yet, this model considers churn in relation to the *whole network*. In a tiny network a churn rate of, e.g., 10 joins per minute might be a serious challenge for the protocol, whereas the same join rate is almost not noticeable in very large overlays. That is why we define the churn rate by using the following model.

Mean online $\mathrm{E}[T_{\mathrm{on}}]$ and offline $\mathrm{E}[T_{\mathrm{off}}]$ times Here a mean online time $\mathrm{E}[T_{\mathrm{on}}]$ and a mean offline time $\mathrm{E}[T_{\mathrm{off}}]$ are specified. If we know the total number of nodes that installed the P2P client N_{total}, we will be able to calculate the average size of the overlay:

$$\mathrm{E}[N] = N_{\mathrm{total}} \cdot \frac{\mathrm{E}[T_{\mathrm{on}}]}{\mathrm{E}[T_{\mathrm{on}}]] + \mathrm{E}[T_{\mathrm{off}}]} = N_{\mathrm{total}} \cdot \frac{\mathrm{E}[T_{\mathrm{on}}]}{MTBJ}, \tag{4.3}$$

with *Mean Time Between two Joins* ($MTBJ$) being the sum of the mean online and offline times.
In this model, the overlay size N is risen by increasing the mean online time $\mathrm{E}[T_{\mathrm{on}}]$ or reducing the mean offline time $\mathrm{E}[T_{\mathrm{off}}]$, and vice versa. Also, the rate is specified in relation to the *behavior of the peers*. Thus, the stress on nodes is independent of the network size N, which is why we use this second model in our simulations. The churn rate per node is defined as the average number of join and leave events per time interval per node (4.4). Thereby, each peer sets off two events in the period between two join events (see Figure 4.1). This results in a churn rate of:

$$\text{churn rate} = \frac{\text{number of events}}{\text{time} \cdot N_{\mathrm{total}}} = \frac{2}{MTBJ} = \frac{2}{\mathrm{E}[T_{\mathrm{on}}] + \mathrm{E}[T_{\mathrm{off}}]} \tag{4.4}$$

If, in a sample network, nodes are participating on average once a day, i.e., $MTBJ = 1$ d, the churn rate per node will be $2/24\ \mathrm{h}^{-1}$. If there are one million different users ($N_{\mathrm{total}} = 10^6$), the overall join rate λ will be $10^6\ \mathrm{d}^{-1}$, or in other words, about 41,667 nodes join per hour. If every node stays online for an average of two hours, i.e., $\mathrm{E}[T_{\mathrm{on}}] = 2$ h, the average ring size will be about 83,333 nodes (4.3).
The *availability* of a node is often defined as its mean session time $\mathrm{E}[T_{\mathrm{on}}]$ (or *Mean Time To Leave* $MTTL$) divided by its $MTBJ$, or equivalently by the sum of its session times divided by its lifetime:

$$\text{availability} = \frac{MTTL}{MTBJ} = \frac{\mathrm{E}[T_{\mathrm{on}}]}{\mathrm{E}[T_{\mathrm{on}}] + \mathrm{E}[T_{\mathrm{off}}]} = \frac{\sum T_{\mathrm{on}}}{\text{lifetime}}. \tag{4.5}$$

In Chapter 5.2.2 we show that the network stability is mainly influenced by the mean session time of the nodes. Longer session times mean lower churn rates, thus less maintenance traffic is required.

Distribution of T_{on} and T_{off} Many measurements in various deployed P2P systems were realized in the last years. Some results are summarized in Table 1 in [RGRK03]. In these studies, median session times in the order of tens of minutes, and a median availability of 30% were observed [BSV03]. From these results, different models of the node session time are specified. Online and offline times are usually modeled by a Negative Exponential Distribution (NED), with means $\mathrm{E}[T_{\mathrm{on}}]$ and $\mathrm{E}[T_{\mathrm{off}}]$, respectively. Then, its Cumulative Distribution Function (CDF) is:

$$F(t) = 1 - e^{-\frac{t}{\mathrm{E}[T_{\mathrm{on}}]}}. \tag{4.6}$$

The exponential distribution is memoryless, i.e., the time at which a node fails is not correlated to its session duration. Thus, distinguishing nodes with long session time from

other nodes is difficult. In [MCVR03] the following rule to identify the 10% of nodes with the longest session time is presented: If the regarding session time of a live node is larger than $\frac{1}{2}\,\mathrm{E}[T_{\mathrm{on}}]\log 10$, it starts to accept extra roles in the system's routing and maintenance.
In contrast to that, some researchers [BC98, CL99] state that a Weibull distribution fits the offline times they measured in Web traffic better. Similar to the results, [Sch05] shows, that the Weibull distribution is a more accurate approximation of session times in P2P overlays. The Weibull distribution has a memory effect, i.e., the longer the current live time of a node, the higher the probability that it will stay online for another time interval. This also reflects the evaluations performed in [MM02, SGG02]. In P2P overlays this effect can be exploited to assign extra roles to peers with long online times. Other researchers suggest using a power-law distribution [KSS05, TJ07] that has an even stronger memory effect.
In our simulations, we assign the same functionality to all peers, and we do not set up a hierarchical overlay. Thus, we are not interested in estimating the time a peer will stay online. That is why we prefer the first model with exponentially distributed session times due to its simplicity.

Distribution of search rate Finally, the time between two queries is exponentially distributed, i.e., searches follow a Poisson model with a rate of $\frac{1}{\mathrm{E}[T_{\mathrm{search}}]}$. This is a common approach used in telecommunication systems [Sch05, Sri01]. The distribution of the search duration in structured P2P overlays will be evaluated in Chapter 5.

4.2. Modeling transmission time in overlay simulations

Modeling the network is a mandatory part for simulating networking applications. In many simulations, it is sufficient to use a model that adds a constant delay to all packets. In some cases a transmission delay may even be neglected at all. In our research on applying P2P mechanisms to Voice-over-IP (VoIP) solutions (Section 7), we try to achieve a certain Quality of Service (QoS) by, among other methods, reducing the call setup delay. This delay is defined as the time interval between entering the last dialed digit and receiving the ringback [ES00]. In VoIP systems this delay consists of several Transmission Times (TTs) to contact the proxy or redirect server and receive its answer, the time to locate the user in the database and small computation delays. In the P2P variant, the user database is distributed among the participating peers. Thus, we can reduce the call setup delay by accelerating lookups in the distributed database.
We use a DHT to store different kinds of resources in the network. For example, a resource might be a pair ⟨nickname; IP:port⟩. If someone wants to contact another user with a known nickname, the network will be queried for the respective resource, resulting in the IP address the VoIP application must contact. In DHTs, each lookup is routed through the network passing several other peers. Each hop of this route adds an additional delay to the overall lookup time. The DHT protocol is responsible for the number of hops

Model	Computational cost	Memory	Comment
Analytical function	simple, inexpensive	$O(1)$	no geographical information, high jitter unavoidable
Lookup table	simple, inexpensive	$O(N^2)$	high precision, data available
Network topology	complex	high	problematic data acquisition
Coordinates-based	inexpensive (runtime), expensive (in advance)	$O(N)$	good precision, data available

Table 4.1.: Different approaches for modeling network Transmission Time

that a lookup takes on average. Chord, for example, finds a resource within $O(\log_2 N)$ hops in a network with N nodes. The overall lookup time will be decreased if either the number of hops is reduced or the network transmission delay is decreased.
In general, connections to geographical close peers have a smaller delay than connections to more distant peers. As geographical positions are available very seldomly, proximity is often defined by short network transmission delays (see Section 6.2.1.2).
Simulating proximity requires an accurate network model, where connections have realistic transmission delays. Table 4.1 gives a short overview of different approaches to model TTs. The most simplistic way is to use analytical distribution functions, for example, negative exponential distributions. While they neither require difficult computations, nor huge amounts of memory, they are not able to cope with the geographical network topology. As a direct consequence, different network TTs between two nodes are calculated for every packet, which also makes high jitter values unavoidable. Thus, using an analytical distribution function is not feasible when simulating proximity-aware protocols.
Storing all inter-node TTs in a lookup table would lead to very high precision, but is not applicable in huge networks, as the size of the table exhibits quadratic growth with the number of nodes.
Modeling the network topology with routers, autonomous systems and links is a common method to build complex models of the Internet, and therefore, it is applied by many topology generators as Inet-3.0 [WJ02] or BRITE [bri]. The drawbacks of using this method are that it is problematic to acquire real Internet topologies and a large amount of memory is required for huge networks. Also, the computation of routing paths and TTs is complex and therefore slows down each simulation run.
We present a topology model, which is based on network coordinates. It is characterized by a relatively high precision, yet low memory and computation costs during the simulation [KNH+07]. The required memory scales linear with the number of nodes in the network. The computation of the network coordinates is time expensive, but is done offline and the coordinates may be re-used in different simulations. Real Internet measurements are available from CAIDA [cai], which allows simulations to be as close as possible to real network conditions. The basic idea is using network coordinates for estimating the TT between two nodes. The inter-node TT is directly proportional to the geometrical distance in the coordinate space. Note that the inter-node TT is not always directly proportional to the geometrical distance in the real world. For example, nodes

connected to the same Internet Service Provider (ISP), but located in different countries or even on different continents, sometimes may be able to communicate with a smaller TT than nodes that are in geographical proximity, but belong to different ISPs. However, as shown in Section 4.2.3, a certain correlation between TTs and geographical proximity is noticeable. In Subsection 4.2.1 we describe the Global Network Positioning (GNP) method that we use to construct the coordinate space. Subsection 4.2.2 explains how GNP is used in our simulations and Subsection 4.2.3 shows results that are obtained by using this network model.

4.2.1. Global Network Positioning (GNP)

GNP [NZ01] was originally developed for predicting packet delays from one host to another. Each node periodically pings a set of *monitors* (or *landmarks*) $\mathcal{M}$ and measures the required Round-Trip Times (RTTs). With this information and the known monitor coordinates, the nodes are able to compute their own position in the geometrical space. Creating a new d-dimensional coordinate space at first requires calculating the coordinates of the monitors. To achieve a high precision, it is suggested to choose monitors located as far apart as possible. All RTTs between the monitors must be known and the number of monitors M must be greater than the number of dimensions d ($M > d$). The error between the *measured* distance $\hat{t}_{n_1n_2}$ and the *calculated* distance $t_{n_1n_2}$ between the two nodes n_1 and n_2 is defined as:

$$\epsilon(t_{n_1n_2}, \hat{t}_{n_1n_2}) = \left(\frac{t_{n_1n_2} - \hat{t}_{n_1n_2}}{t_{n_1n_2}}\right)^2 \tag{4.7}$$

Subsequently, we can compute the coordinates of the monitors c_{m_i} by minimizing the following objective function for every monitor m:

$$f_{obj,m}(c_{m_1}, \ldots, c_{m_M}) = \sum_{i,j \in \{1,\ldots,M\}: i>j} \epsilon(t_{m_im_j}, \hat{t}_{m_im_j}), \forall m_i, m_j \in \mathcal{M} \tag{4.8}$$

After measuring the RTT to at least M' ($d + 1 < M' \leq M$) monitors, a node n can compute its own coordinates c_n by minimizing the following objective function:

$$f_{obj,n}(c_n) = \sum_{i \in \{1,\ldots,M\}} \epsilon(t_{m_in}, \hat{t}_{m_in}), \forall m \in \mathcal{M} \tag{4.9}$$

The estimated TT $t_{n_1n_2}$ between two arbitrary nodes n_1 and n_2 with coordinates $(c_{n_1,1}, \ldots, c_{n_1,d})$ and $(c_{n_2,1}, \ldots, c_{n_2,d})$ can finally be obtained by computing the geometric distance between the two nodes in the coordinate system:

$$t_{n_1n_2} = \sqrt{(c_{n_1,1} - c_{n_2,1})^2 + \cdots + (c_{n_1,d} - c_{n_2,d})^2} \tag{4.10}$$

We use the Simplex Downhill Method proposed by Nelder and Mead [NM65] to solve these minimization problems, because it is very easy to implement.

4.2.2. Applying GNP for modeling network transmission

We use GNP coordinates in a slightly different way in combination with ping measurements acquired from CAIDA's skitter project [cai]. There are 14 monitors available in the dataset (Table 4.2), which are mostly positioned at DNS roots. These monitors perform daily RTT measurements to a list of selected nodes, which are spread over the entire IP space. We are not going to use all monitor nodes for the computation of the coordinates, because good values can already be gained with $d + 1$ monitors and the computation duration will increase significantly if more monitors are used. Using $d = 5$ we achieved almost accurate TTs. As mentioned above, it is important to carefully select the monitors. A lot of research has been carried out in this area [NZ01, TC04]. We select our monitors with the help of a *maximum separation* algorithm, i.e., we try to select monitors that have a maximized inter-monitor distance (by means of TTs). This maximization can be solved very easily, as there are only 14 different monitors available, and it leads to good results. Another promising, but more computation expensive method is the *Greedy algorithm* that chooses the set of monitors, which minimizes the average distance error (4.7) between all monitors.

Table 4.3 shows the symmetric RTT matrix achieved from a subset of 6 monitors that we used to build a 5-dimensional coordinate space. The coordinates the monitors can now be calculated by minimizing Equation (4.8) for all monitors.

The skitter data set comprises no inter-node RTT measurements, but it provides us with RTT measurements from each monitor to about 300,000 hosts (Table 4.4). Coordinates for these hosts can be computed by minimizing Equation (4.9) for all hosts. This computationally intensive multi-dimensional minimization problem is solved offline. Coordinates for the CAIDA dataset have to be computed once, and can consequently be reused for all simulations without any further computation costs. The mean TT for the CAIDA measurements is about 80 ms.

Scenarios we are simulating are described in a *source file*, where parameters like the number of total participants N_{total}, the number of online nodes N and the average online time $\mathrm{E}[T_{\text{on}}]$ are set. From this file, a traffic generator computes all join, leave and search events, as well as the IDs of nodes and content. We call its output *event file*. The event file can then be put into our *coordinate tool*, which assigns a random host from the CAIDA dataset to each node in the event file. The tool also adds the appropriate coordinates to the event file. Our simulator will automatically detect whether coordinates are set or not, and uses the coordinates or a negative-exponential distribution to compute TTs, respectively. TTs between nodes are calculated with Equation (4.10), but would be constant for each transmission between the same two nodes. Therefore, a log-normal distributed jitter will be added to the TTs, if coordinates are used. This proceeding is based on real Internet measurements [HMT+05] and results in an even more realistic model. A lognormal distribution is denoted as $\lambda(\mu, \sigma^2)$, and its Probability Density Function (PDF) is expressed as:

$$\Phi(x;\mu,\sigma) = \begin{cases} \frac{1}{x\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right) & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

Monitor name	Location	IP address
arin	Bethesda, MD, US	192.149.252.8
b-root	Marina del Rey, CA, US	129.9.0.109
cam	Cambridge, UK	128.232.97.8
cdg-rssac	Paris, FR	195.83.250.10
d-root	College Park, MD, US	128.8.7.4
e-root	Moffett Field, CA, US	192.203.230.250
i-root	Stockholm, SE	192.36.144.117
ihug	Auckland, NZ	203.109.157.20
k-peer	Amsterdam, NL	193.0.4.51
k-root	London, UK	195.66.241.155
nrt	Tokyo, JP	209.249.139.254
riesling	San Diego, CA, US	192.172.226.24
uoregon	Eugene, OR, US	128.223.162.38
yto	Ottawa, CA	205.189.33.78

Table 4.2.: CAIDA monitor hosts

	b-root	d-root	i-root	k-root	nrt	ihug
b-root		68.882	186.476	172.536	127.812	185.123
d-root	68.882		118.987	95.266	208.739	229.618
i-root	186.476	118.987		36.523	315.139	319.436
k-root	172.536	95.266	36.523		275.874	312.360
nrt	127.812	208.739	315.139	275.874		138.511
ihug	185.123	229.618	319.436	312.360	138.511	

Table 4.3.: Inter-monitor RTTs (in milliseconds)

	b-root	d-root	i-root	k-root	nrt	ihug
18.166.0.1	84.055	10.535	117.495	85.541	210.628	251.454
81.165.0.1	146.550	85.889	36.159	9.554	284.824	291.408
198.31.255.254	8.777	98.625	177.254	145.013	127.879	196.591
200.63.11.1	249.277	184.413	1060.883	309.182	376.213	523.068
217.200.12.1	172.939	107.576	75.661	27.682	309.860	321.287
...	...	...	...	...	...	...

Table 4.4.: Host-monitor RTTs (in milliseconds)

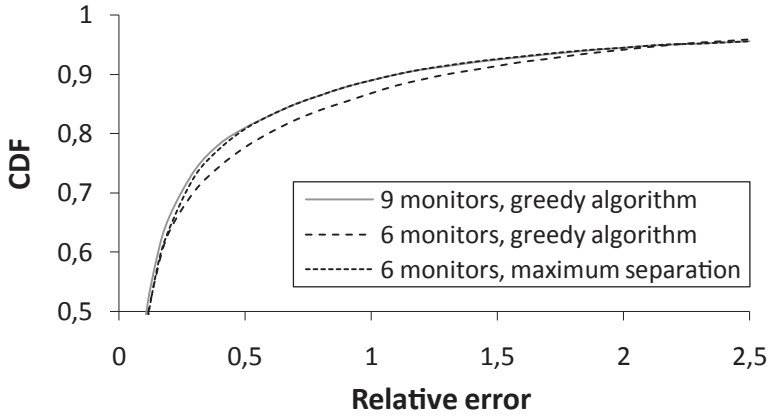
The distribution can be calculated from measurements where the mean TT μ and the standard deviation σ are known.
Additionally, our model takes *packet loss* into account. We assume that packets are dropped with the same probability. As our model does not construct a detailed physical topology, it is not possible to consider congestion, and therefore higher packet loss rates, in certain regions of the topology.

4.2.3. Results

In order to evaluate the quality of our coordinates, or in other words, how accurate we can estimate the RTTs between the nodes compared to the real measurements, we use the directional relative error metric:

$$r = \frac{t_{n_1 n_2} - \hat{t}_{n_1 n_2}}{\min(t_{n_1 n_2}, \hat{t}_{n_1 n_2})} \tag{4.12}$$

Therefore, we select two monitors that have not been used to compute the coordinates and calculate the relative error between them and 2,000 random nodes from our dataset. A directional relative error of plus (minus) one means, that the calculated distance is larger (smaller) by a factor of two as compared to the measured value, whereas a error of zero is a perfect fit. Figure 4.2 shows the performance of both algorithms. Maximum separation with 6 monitors has a performance which is comparable to the Greedy algorithm with 9 monitors. 81% of the calculated RTTs reveal a relative error of less than 50%. On the other hand, 50% of the calculated RTTs have a relative error of less than 12.3%. We use maximum separation, as it requires significantly less computational effort.



Percentile max. sep.	10	20	30	40	50	60	70	80	90
Relative error $\leq$ (in %)	1.83	3.84	6.20	8.90	12.35	17.68	26.93	47.57	111.23

Figure 4.2.: Monitor selection method comparison

To evaluate the precision of calculated RTTs with respect to the measured times, we have grouped the measured times and the corresponding calculated times in bins of 50 ms and plotted the directional relative error of each pair on a vertical line (Figure

4.3). The mean directional relative error is indicated by squares, the 25^{th} and 75^{th} percentiles are indicated by the outer whiskers of the line. The figure also shows that GNP performs quite well for distances under 350 ms. A general trend to undershoot in calculated values is apparent; especially for distances of more than 350 ms, GNP undershoots significantly. Still, only 7% of all evaluated distances are more than 350 ms, so the influence of significant errors for large distances can be neglected. These large errors result from nodes that are located in areas far apart from the monitor nodes, therefore their coordinates cannot be computed precisely.

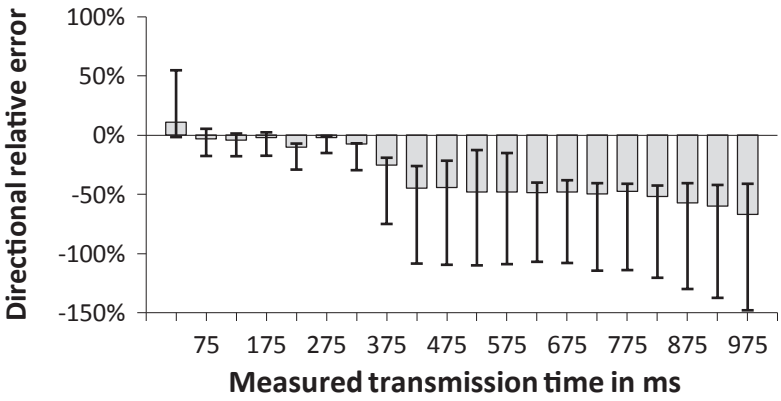


Figure 4.3.: Directional relative error over measured distances

We are mainly interested in using GNP for calculating TTs for our simulations. Thus, we compare the distribution of measured TTs from the CAIDA dataset to TTs calculated with GNP (Figure 4.4 (left)). We have simulated a network with 2,000 nodes performing 200,300 searches in total. Note that the average TTs is the same for both curves (91 ms). The negative-exponential function has a clearly higher standard deviance ($\sigma = 90.99$ ms) than the distribution based on a realistic topology ($\sigma = 61.85$ ms), and there are much more very small (< 25 ms) and large (> 200 ms) values.
Lookups in DHTs are forwarded through the overlay network until the responsible node for the queried key is found. This results in a series of packets that are sent over the network. The total lookup time consists of the corresponding TTs and small additional local computation delays. Figure 4.4 (right) shows the measured lookup times from simulations with and without using coordinates. As expected, both lookup time distributions are very similar. The curves resemble a Gaussian distribution and have approximately the same mean value (NED: 550.64 ms, GNP: 541.04 ms). The curve corresponding to the NED is a bit wider, because the standard deviation is bit larger for the negative-exponential distribution. According to the *Central Limit Theorem* [PP01], the sum of an infinite number of statistically independent random variables has a Gaussian distribution, regardless of the elementary distributions.
However, the GNP-based network model provides simulations with a more realistic framework. Thus, we are able to apply Proximity Neighbor Selection (PNS), i.e., using network latency as the metric by which to choose between neighbor candidates. We will present PNS in greater detail in Section 6.2.1.2, and show its influence on the lookup duration by presenting simulation results.

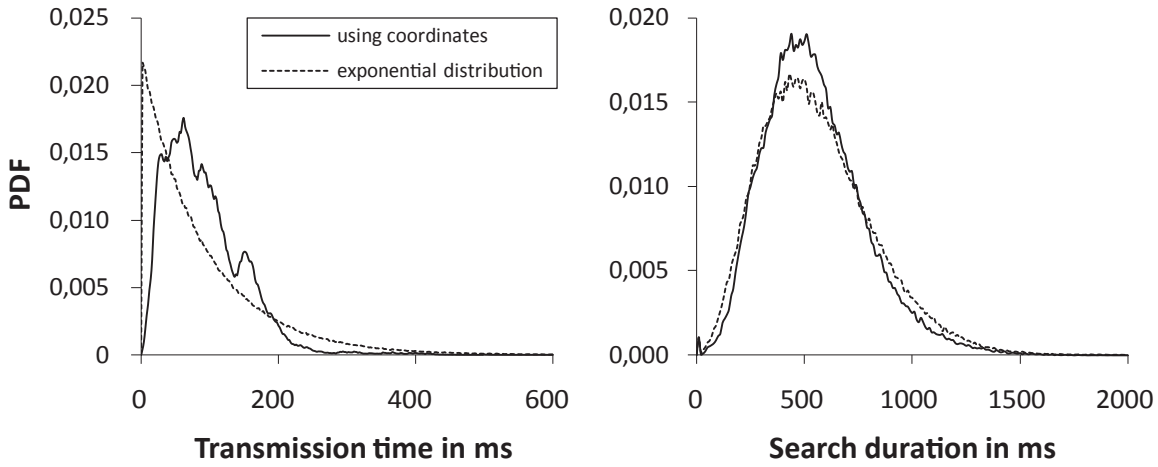


Figure 4.4.: TT distributions (left) and corresponding lookup time distributions (right)

Another interesting phenomenon is shown in Figure 4.5. If our 5-dimensional coordinates are projected to a 2-dimensional coordinate space, a remarkable amount of clustering can be recognized. If we compare the clusters to a world map, even "continents" may be identified in the coordinate space. This is astoundingly, as coordinates have been calculated from TTs only. We take this as another indication, that the calculated coordinates are a good representation of the real Internet topology.

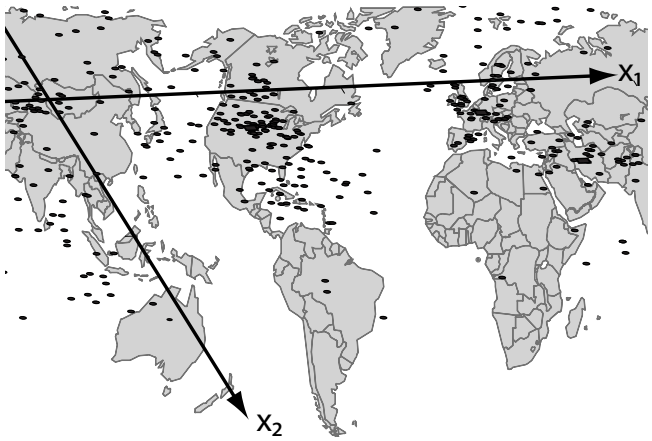


Figure 4.5.: Node distribution in a 2D projection

4.2.4. Predicting inter-node transmission times

The presented network model provides a basis for both estimating and predicting TTs. Usually, nodes estimate the TTs to neighbor candidates by evaluating existing traffic to these nodes or by sending active probe packets. This signaling overhead can be reduced by using synthetic coordinates. Thus, inter-node latencies can be predicted without having to perform an explicit measurement to determine the latency. Network coordinates can be calculated by making use of monitor nodes as it is done with GNP [NZ01] or

PCA [TC03], or by simulating the positions of the nodes with a distributed algorithm like Vivaldi [DLS+04, DCKM04]. We are using the Vivaldi coordinates in our simulations, as the algorithm is fully distributed and computationally inexpensive. Therefore, it seems particularly suitable for applying it to P2P networks.
Please note that the GNP coordinates used for modeling the transmission times must not be accessed from the simulated protocol, as using the same coordinates would result in a perfect prediction. The protocol must implement its own synthetic coordinates and assign them to the live nodes.

4.3. Simulation environment and GUI

To be able to study various performance aspects in detail, we developed a highly-scalable event-based simulator for DHT overlays in ANSI-C [KR88]. The simulation environment was implemented in a joint project with the Institute of Distributed Systems at the University of Würzburg, Germany. It is capable of handling tens of thousands of nodes participating in the network in acceptable simulation time. We are able to do so by simulating only the overlay network and modeling the physical network properties. Additionally, we developed different implementation techniques like compact data structures [BHKE07, EHBK07, BHKE09].
A good overview and comparison of related work is given in [NBLR06]. Despite a large variety of existing tools we decided to implement our own simulation environment (see Figure 4.6), as most simulators are not able to handle such large overlays. Also, other simulators provide only poor or no documentation at all, and are often not flexible enough to fulfill our needs. Mainly, features are missing and the simulation environment is often difficult to extend.
A simulation run is defined using our own very abstract script language. The source file describes the scenario using a number of commands and parameters listed in the appendix. For example, the command `peers <N_total>` defines a total number of clients N_{total}. The command `join <N> <E(T_join)>` denotes that N nodes should join the overlay at a mean rate of $\mathrm{E}[T_{\text{join}}]^{-1}$ ms^{-1}.
The command `user <duration> <E(T_on)> <E(T_off)> <E(T_search)` describes a period of *duration* seconds, where nodes join and leave the network with the parameters $\mathrm{E}[T_{\text{on}}]$, $\mathrm{E}[T_{\text{off}}]$, and $\mathrm{E}[T_{\text{search}}]$ as specified in the source file.
This source file is parsed by the traffic generator, which translates the code into actual events (see Appendix B). Thereby, it generates random IDs for all clients and documents, and it calculates the start time of all join, fail, and search events. For example, during its lifetime a node performs several lookups at random times for keys selected randomly from a uniform distribution. These events are written to the event file, which can then be used as input for the P2P simulator. The node ID is valid for the whole simulation run, i.e., nodes are not assigned a new ID each time they (re-)join the network. Separating the traffic generator and the actual simulator offers more flexibility. Event files can be re-used for validating the simulation, for running simulations with different parameter settings, and for simulating different protocol versions. Additionally, events from prototype studies or emulations can be extracted and used as input for the simulator. Both traffic generator

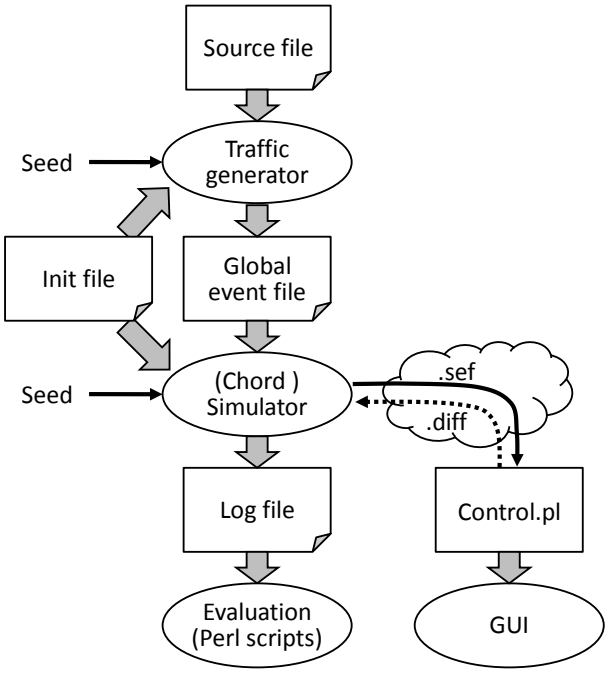


Figure 4.6.: Sketch of the simulation workflow

and simulator may be called with different initial seeds.

The output of the simulation is written to two separate files. The log file contains various debug information, as well as measurements of the statistic function. This file is later on parsed by (Perl) scripts that extract appropriate information and calculate further data, like variances. Additionally, an ordered list of all data changing events is stored in a simulation event file (.sef) that may be played back in the GUI later on.

The Chord simulator is built on top of an event queue [BHKE07]. This queue provides two functions: add a new event at an arbitrary time in the future, and retrieve the next event in time. An event consists of the time it is due, a pointer to the function that is to be called when the event is processed, and optional arguments that are assigned to the called function. The first step of a simulation is parsing the event file and adding all events to the queue. Then, all events in the queue are processed chronologically, where the function triggered by the event may call other functions. It may also add further events to the queue. Typical examples are adding the next execution time of periodic events, or adding a timeout event.

The basic Chord protocol is implemented according to the specifications in [SMK+01b]. Modifications to the protocol are described in the appropriate text passages. Each simulation is run with a different set of parameters. In order to be able to evaluate the influence of each individual parameter apart from all other parameters, we change only one parameter in each of our simulations. All other parameters are kept constant with the values given in Table 4.5 (except if stated otherwise).

In our simulations we use a total number of 40,000 peers, which we found to be sufficiently large to capture all important effects regarding the overlay size. Also, we consider

$m = 28$	Size [in bit] of the Identifier (ID) space
$N = 20{,}000$	Number of live nodes
$N_{\text{total}} = 40{,}000$	Number of total clients
$K/N = 10$	Number of keys per node
$L = 5$	Number of successor entries
$R = (L+1)/2 = 3$	Size of Replication group
$\text{E}[T_{\text{on}}] = \text{E}[T_{\text{off}}] = 30$ min	Mean duration of online sessions and offline periods
$t_{\text{stab}} = 30$ s	Neighbor stabilization period
$t_{\text{fu}} = 5$ min	Finger update period
	(all different fingers are updated once in t_{fu})
$t_{\text{rep}} = 5$ min	Replication period
$E[TT] = 80$ ms	Mean Transmission Time of packets
	(Negative Exponential Distribution) [cai]
$t_{\text{hop}} = 1625$ ms	Hop timeout expiration
$t_{\text{search}} = 10$ s	Search timeout expiration
$TTL = 20$ hops	TTL counter for avoiding loops

Table 4.5.: Default values for common simulation parameters

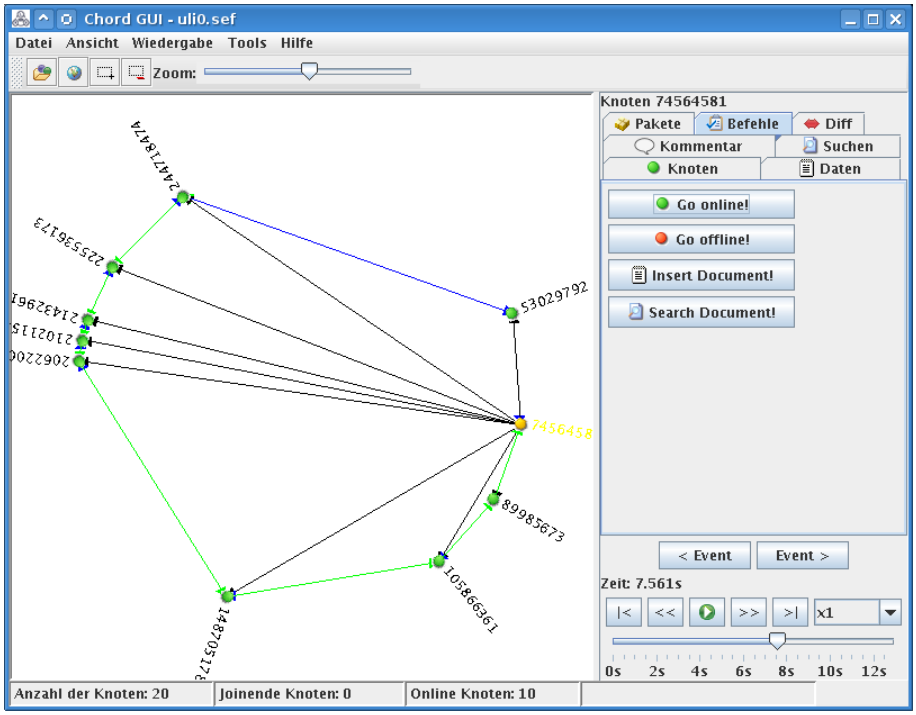


Figure 4.7.: GUI: Visualization of the overlay

only node failures and no graceful node leaves, thus simulating the worst case for the stabilization of the overlay, and putting the maximum stress on the network. Generally, packet loss will not be simulated, except if explicitly stated in the analysis. We simulate UDP packets, and retransmission in case of packet loss is covered by the P2P protocol itself.
The statistic function is called every 10 s. It calculates the sums and mean values of various parameters for the last interval of 10 s. All available parameters are listed in the appendix and used parameters are explained in Section 5.1. Most results are evaluated by calculating the mean $E[X]$ of all samples X within one simulation. All experiments were run until the confidence intervals became negligibly small and could thus be omitted in most of the figures.

Graphical User Interface (GUI) We also developed a Graphical User Interface (GUI) for our Chord simulation environment that serves various different purposes. First, it helps the developer of a protocol to debug its functionality. It provides the developers with a simple and intuitive way to monitor and comprehend the internal processes of the protocol. The impact of changes and enhancements can easily and quickly be tracked and evaluated. By zooming to different views, the overall behavior as well as specific details can be monitored and controlled. Moreover, the GUI can be used for demonstrating the mechanisms of the simulated protocol. This feature is useful for teaching the basics of P2P protocols, as well as for giving an insight to the latest improvements of a specific algorithm.

In its main windows, the GUI visualizes Chord's circular ID space (Figure 4.7). Nodes are displayed using different colors indicating the current node state. The user may select any node by clicking on it. The selected node is highlighted, and its fingers are visualized in the graphical representation of the ring structure. Moreover, in an information panel on the right part of the GUI various detailed information on the selected node is displayed, including all predecessor, successor, and finger entries, the current node state, shared documents and stored ⟨key; value⟩-pairs. Moreover, all packets sent and received by the selected node can be listed on a graphical time line, and a '*comment*' tab allows for printing any debug information implemented in the simulator.

In the bottom of the window, some general information regarding the overlay is given, like the current number of live nodes. The lower right corner provides a remote control for the simulator. Actually, no real-time control of the simulation is given, but the simulation is recorded in a `.sef` file that is played back by the GUI. The user may play the recorded simulation with variable speed, proceed forward/reverse event by event, or skip to a certain event or point in time. Thereby, integrated search functionality assists finding the right events.

Additionally, the simulation may intuitively be modified during the play-back. By clicking the appropriate button, the selected node goes online or offline, inserts, or searches a document. Moreover, an editor for adding and deleting diff-events is integrated in the GUI. Thereby, instead of actually modifying the simulation, the modifications are transferred to the simulator in a `.diff` file, and the simulation is restarted from the beginning. This approach was implemented, as periodically storing the complete state and restarting the simulation from the time of the first modification is not feasible. Although the GUI is able to visualize even large scenarios, smaller networks are better suited for demonstrating and analyzing the protocol. Thus, re-running the simulation is finished within a few seconds and the user is hardly aware of it. The communication between simulator and GUI is handled by a perl script (`control.pl`). Thereby, simulation and GUI must not run on the same machine. Modifying the simulation during play-back is especially valuable to perform "*what if*" analysis, i.e., to easily (re-)produce and analyze certain tricky special cases.

A major aspect of distributed systems is communication. Nodes frequently exchange information by sending different types of packets through the network. Thus, a visualization of the packet flow is a valuable tool for debugging implementation flaws and understanding the system behavior in detail. For the sake of clarity, we decided to display a packet sequence diagram for selected nodes in a separate window, instead of additionally visualizing all packets in the main window. In the screenshot in Figure 4.8 we selected four nodes. Two of them can be observed joining during the displayed time window (denoted by the blue dot), and going online after a successful stabilization (green dot). Moreover, the figure shows a received NOTIFY packet from node 933476 that is not selected, as well as a lost STABILIZATION packet (red X). In the context menu the user may filter certain packet types and vary the displayed time scale.

Summarizing, the GUI is a powerful tool that we intensively used for analyzing the impact of modifications and implementation details. Also, the GUI provides means to demonstrate and lecture the basic functionality of structured P2P protocols.

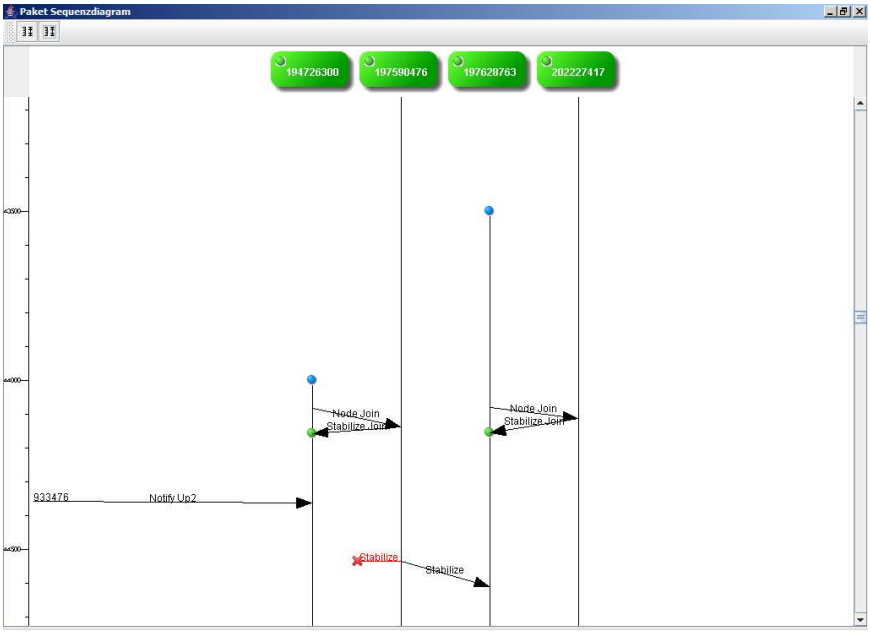


Figure 4.8.: GUI: Packet sequence diagram

4.4. Conclusion

In this chapter we presented suitable models for simulating the user behavior and realistic transmission times. The user model applied in our simulation environment is based on mean online $\mathrm{E}[T_{\mathrm{on}}]$ and offline $\mathrm{E}[T_{\mathrm{off}}]$ times. Different churn rates can be achieved by modifying these values. The distribution of online/offline periods and search rates is usually modeled with a NED, however some researchers argue that a Weibull distribution better fits the measured values. Applying a Weibull distribution, the remaining online time is a function of its current live time.

We also presented a scalable and realistic model for Internet transmission times that is based on network coordinates. In a network with N nodes, our model scales with $O(\log_2 N)$, whereas a simple lookup table would be of size N^2. Its main advantage compared to other models (using analytical functions) is the fact, that the transmission delay between any two nodes is not random (e.g., based on a negative-exponential distribution) but constant over time. An additional jitter (using a log-normal distribution) and a constant packet loss rate make our model even more realistic. We showed how to calculate network coordinates from a given dataset of transmission times by minimizing the relative error between measured and calculated distances between the peers. We also evaluated the approximation error and precision of this model.

A corresponding technical report [HBS+05] addresses many other topics, like whether to simulate on packet or on application level, how to model bandwidth in file sharing systems or how to design efficient data structures and event algorithms.

Finally, we presented our simulation environment that is built around a highly scalable event-based simulator for DHT overlays. A traffic generator translates the abstract de-

scription of a scenario into actual events. These events are then parsed by the simulator. By separating traffic generator and simulator, event files can be re-used for simulating different settings and protocol versions. Usually, our simulations consist of 40,000 peers, with around 20,000 peers being online at the same time. A statistic function calculates sums and mean values for various parameters and writes the results in a log file. Moreover, a `.sef` file containing all simulated events is generated, and the file can be played back in the GUI. The GUI is a powerful tool enabling its user to easily monitor the inner-workings of the simulated overlay and intuitively evaluate the impacts of modifications. A control script provides a feedback channel for the GUI, thus supporting a direct graphical modification of the simulated scenario.

CHAPTER 5

Performance, robustness, and cost analysis

In this chapter we evaluate the scalability of the Chord protocol. In this context, functional scalability is defined as the ability of the protocol to scale to large networks with moderate or no effects. Similarly, stochastic scalability is the ability to handle high user activity without serious impact on the system [Bin08]. We evaluate this impact by analyzing the performance, robustness, and costs of the protocol [KBH05]. The routing performance is the average lookup path length, i.e., the number of hops traversed for finding keys. The overlay must also be capable of repairing changes in its structure. We will call a protocol robust if it is able to maintain its functionality and overlay structure despite high churn rates. Furthermore, costs for finding keys and maintaining the overlay are evaluated. Thereby, an efficient trade-off between small maintenance cost and a correct overlay structure must be made.
We introduce these metrics in detail in the next section, and then use them to evaluate the Chord protocol. Thereby, our main focus is on the stability of the overlay structure, as only correct routing information assures correct and short lookups.

5.1. Metrics

5.1.1. Lookup path length and search duration

Structured P2P protocols have been developed to scale better than earlier protocols. Using a proactive routing scheme, it is possible to route lookups on one short and deterministic path to the node responsible for the requested content. Infrequent items can be resolved as good as popular content. A short *lookup duration* is important, as in structured P2P almost all processes, such as joining the network, finding new pointers to distant nodes, inserting content, and of course searching for content, are based on lookups for keys (see Section 3).

The duration of a lookup mainly depends on the lookup path length and the average Transmission Time (TT) in the network. We define *lookup path length* as the number of steps (hops) required in the lookup process. In each step, another node, which is closer to the queried key, is contacted. However, the lookup duration is not directly proportional to the lookup path length and the average TT in all scenarios. In dynamic networks timeouts will occur if, due to obsolete pointers, nodes try to contact other nodes no longer participating in the system. As shown in Section 5.2.2 these timeouts significantly affect the lookup duration negatively. In contrast to that, certain methods (see Section 6.2), like preferring neighbors that are close in terms of TTs, or applying parallel lookups, have a positive effect on it. Also, applying iterative or recursive routing results in different lookup durations.
Thus, we think that the lookup path length is a more meaningful metric for evaluating and comparing different lookup protocols, as this metric is more closely related to the algorithm itself and depends less on extraneous influences. Structured P2P approaches differ, amongst other things, in the average lookup path length: from $O(\log_2 N)$ [SMK+01a, RD01, MM02] to $O(\sqrt{N})$ [RFH+01] to $O(1)$ [GBL+03, GLR04] hops. In this regard, Chord scales with $O(\log_2 N)$ (see Section 5.2.1).

5.1.2. Robustness of the overlay structure

The second metric we use to evaluate structured P2P protocols is the *robustness* of its overlay structure. In literature, this metric is also referred to as the *stability* of the overlay.
We measure the robustness of an overlay by comparing all local neighbor lists with a global view of the network and counting the discrepancies. In structured P2P protocols nodes monitor their direct neighbors in order to detect failed nodes and reporting the failures to other nodes. Also, correct pointers to neighboring nodes must be maintained to determine which keys a node must store, and where to correctly locate a queried key. In this context, we distinguish between nodes that have wrong direct neighbor pointers and nodes that have any wrong neighbor entry. Wrong direct neighbor entries are worse than other false entries in the neighbor list, as a node mainly communicates with its direct neighbors and its fingers. Packets which are sent to wrong neighbors are detoured and must be resend or forwarded to the correct node, thus increasing the lookup duration. Additional neighbors are mainly stored as a replacement for failed entries. Nonetheless, these entries also should be correct in order to be able to correctly replace failed direct neighbors. In Section 5.2.2, amongst other things, we evaluate the influence of wrong neighbor entries on the average lookup duration.
Nodes that join, leave or fail induce wrong entries in the neighbor lists. Joining and leaving nodes lead to temporal discrepancies that do not affect the lookup success, but rather cause a short indirection and thus a small increase of the lookup duration. In contrast to that, failures must be detected by adjacent nodes. In the time between the failure, its detection and the correction of all neighbor lists, lookups that are transmitted to the failed node are lost and must be re-transmitted to another node.
In our simulations, we periodically compare the actual node state to the global view of the ring ($\Delta t = 10$ s). Therefore, we maintain a second overlay topology, where the

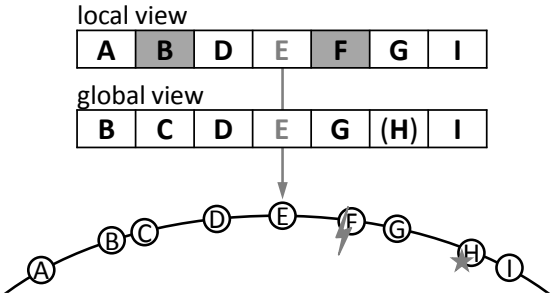


Figure 5.1.: Errors in node E's neighbor list (local view) are detected by comparing it with a global view on the ring.

neighbor lists are corrected immediately after a join, leave, or failure event has occurred. Then, we can compare overlay networks with each other and detect all errors in the node states. We use a rather complex comparison algorithm that does not simply compare entry by entry. If a node has left the network, and its direct neighbor is not yet aware of it, but the rest of its neighbor list is correct, our algorithm will count only one error. If a node is still joining the ring, we will not increase the number of errors, if a neighbored node has not yet learned about the new node.
Figure 5.1 shows a segment of a sample chord ring and the local and global neighbor list of node E. The direct neighbor F has recently left the network (lightning symbol), but E still has a pointer to F (Error 1). Node H is currently joining the network (star symbol), and has not initiated its own list of neighbors. Yet therefore, it is correct that H is not in the list of neighbors of E. On the contrary, node C has already finished its joining algorithm and is in state present. However, it is not yet inserted in the neighbor list of E (Error 2).
In related work, the stability of a P2P network is often not considered at all or it is inferred from the average search duration. However, evaluating stability by search duration yields two disadvantages. First, as discussed previously, the search duration does not only depend on the applied stabilization algorithm, but also on search and content replication mechanisms. Additionally, the average search duration is nearly constant for scenarios with moderate and low churn rates (see Figure 5.2). Therefore, it is difficult to estimate network stability from search durations.

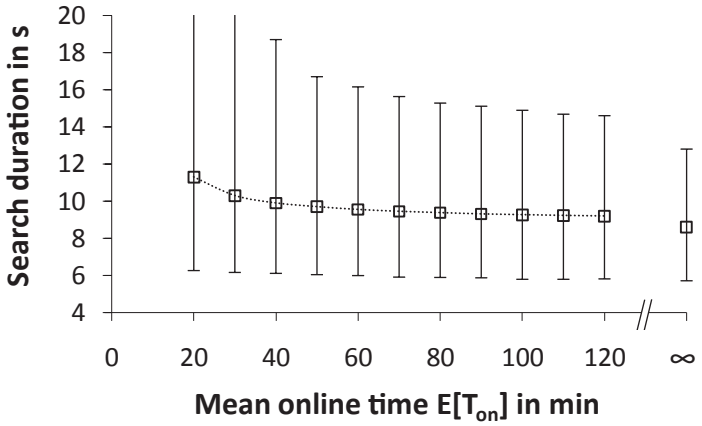


Figure 5.2.: The median of the search duration increases only slightly if moderate or low churn rates occur.

5.1.3. Maintenance overhead

The *costs* of keeping up the overlay structure can be evaluated by measuring the bandwidth required for signaling messages. These costs do not include bandwidth used for transmitting payload like the ⟨key; value⟩-pairs. Signaling traffic can be divided into three different cost types:

- Costs for keeping up the overlay structure, including all messages that are sent while nodes join and leave the network, as well as all messages that are sent during stabilization.
- Costs for keeping the routing entries up to date.
- Costs for inserting, republishing and looking up content in the network.

The costs for keeping the routing entries up to date may be difficult to distinguish from other costs, as finger entries may be updated with information that is acquired during lookups, e.g., if recursive routing or symmetrical routing using freebie fingers (see Section 6.2.3) is applied. Costs for content management mainly depend on the applied replication rate and algorithm.
In the context of this evaluation, we solely focus on costs required for keeping the network structure and the routing entries up to date. Hereby, all protocols are able to adjust certain design parameters in order to find a trade-off between frequent and extensive signaling messages (i.e., high signaling overhead) and a low signaling overhead (i.e., low stabilization frequency and thus a less stable overlay topology). The most important parameters we evaluate in Section 5.2.3 are (1) the number of neighbors nodes maintain and exchange between each other, (2) the frequency of keep-alive and stabilization messages, and (3) the interval of the routing entry updates.

5.2. Evaluating the Chord protocol

In the following section we analyze parameters like the number of participants N, their dynamic behavior (churn), and various design parameters, using the introduced metrics. Among these, only the design parameters can be influenced directly, whereas the number of participants and their behavior might only be influenced by providing incentives to them. Note that we simulated and evaluated a slightly advanced Chord stabilization using symmetrical stabilization and NOTIFICATION messages (see Section 6.1.2).

5.2.1. Number of Participants

Structured P2P protocols were designed to address the scalability problem inherent to flooding. In this section we show the results of our evaluations. We demonstrate that regarding the stability of the network, its search rate and search success, as well as the required signaling traffic the protocol scales well with the size of the network.

Lookup path length In Chord the average number of hops for finding the predecessor of an ID is $1/2 \log_2 N$ hops [SMK+01b] in a stable scenario. Thus, a lookup for the successor of an ID is resolved in $1/2 \log_2 N + 1$ hops ('+1', as we need an extra hop to reach the succeeding node).
In the following we assume a stable 128-bit network. For each network size we simulated 10^3 different networks with N nodes and random IDs, and in each simulation random nodes initiated 10^5 lookups for random keys, hence, 10^8 lookups will be performed in total. Note that in our implementation each node n initiating a lookup checks if it is responsible for the queried key itself. If so, it answers the lookup without actually querying the network.
Figure 5.3 shows the lookup path length for varying network sizes. As expected, Chord's average path length is almost exactly $1/2 \log_2 N+1$ hops. Only very small networks exhibit a smaller hop count, as many lookups are directly answered by the initiating node, and no lookup is routed through the network. In contrast to that, in the mathematical model the lookup is initiated in any case. For larger networks this special case is very rare and thus has no significant influence on the lookup path length. Also, its median is close to the average, indicating a symmetrical distribution of the lookup path length.
The authors of Chord also prove mathematically that the maximum path length is $\log_2 N$ hops in a fully populated network, whereas w.h.p. the predecessor of a key can be located with at most $2 \log_2 N$ hops in a densely populated network [SMK+01b]. However, worst case scenarios with a longer lookup path length can be constructed. Consider an 8-bit network with 8 nodes whose IDs are 0, 64, 96, 112, 120, 124, 126, and 127. If node 0 is querying for ID 128 7 hops will be necessary to reach the predecessor of ID 128, although $2 \log_2 N = 6$. For larger ID spaces similar artificial scenarios exist. In our simulations, the maximum observed path length was by far less than $2 \log_2 N$ hops (see Figure 5.3). For medium to large networks ($N \geq 512$), 99.9% of all lookups could even be resolved with at most $\log_2 N$ steps.
In order to find out the influence of the node density on the lookup path length we performed a set of simulations with a constant number of nodes $N = 2^{12}$, but varying

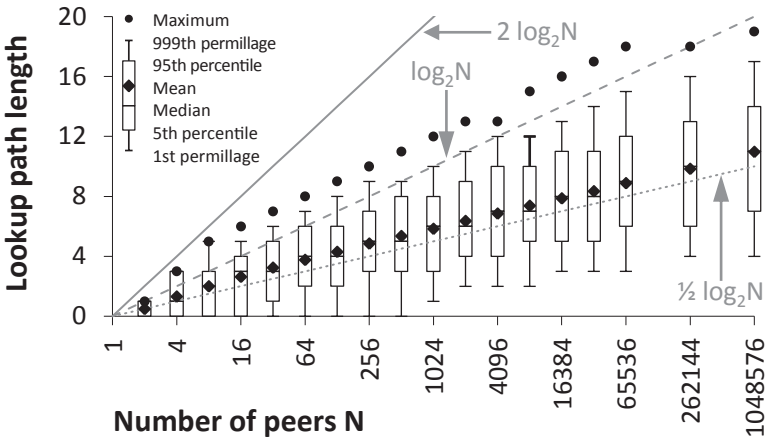


Figure 5.3.: Lookup path length for varying network sizes

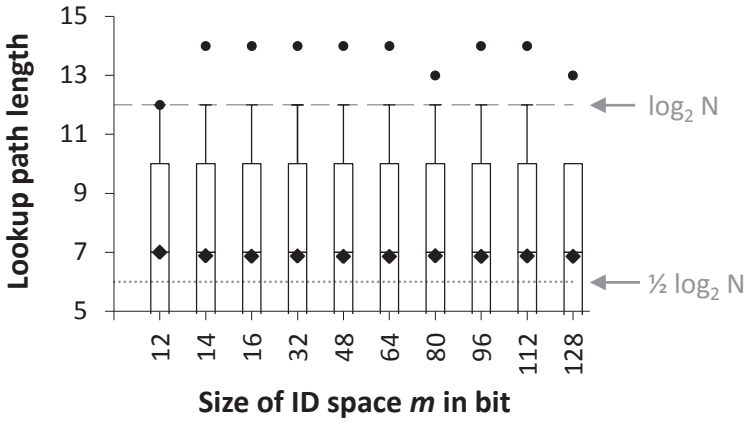


Figure 5.4.: Lookup path length in a 4096 node network for varying sizes of the ID space

sizes of the ID space (2^m). Simulation results are shown in Figure 5.4. The x-value on the left-hand side corresponds to a fully populated network (2^{12} nodes in a 12-bit ID space). Keeping the number of nodes constant, the larger the size of the ID space, the more densely populated the overlay is. For each x-value we triggered 10^5 lookups with random initiators and keys, and repeated this for 10^3 random overlays without churn, resulting in a total of 10^8 lookups. The error bars show the 1$^{\text{st}}$and 99$^{\text{th}}$percentiles. The average path length is 7 hops for the fully populated network ($m = 12$), and around 6.85 hops for larger ID spaces. This difference can be explained by the fact that, w.h.p., in densely populated networks the actual finger pointers differ from the theoretical finger IDs, as no nodes with the theoretical IDs exist. Thus, the first node succeeding the theoretical ID is selected as finger. As a result, finger intervals are slightly larger, and the distance to the queried ID is reduced a little bit more with each hop. The figure also shows the maximum observed path lengths, i.e., 12 hops for $m = 12$ and up to 14 hops for $m > 12$. The 999$^{\text{th}}$permillage is approximately $\log_2 N$ hops in this scenario.
In Figure 5.3 the 1$^{\text{st}}$and the 99$^{\text{th}}$percentile are arranged symmetrically to the median and the median is close to the mean value. This indicates that the Probability Density Function (PDF) of the path length is symmetrical to its average value. Figure 5.5 shows

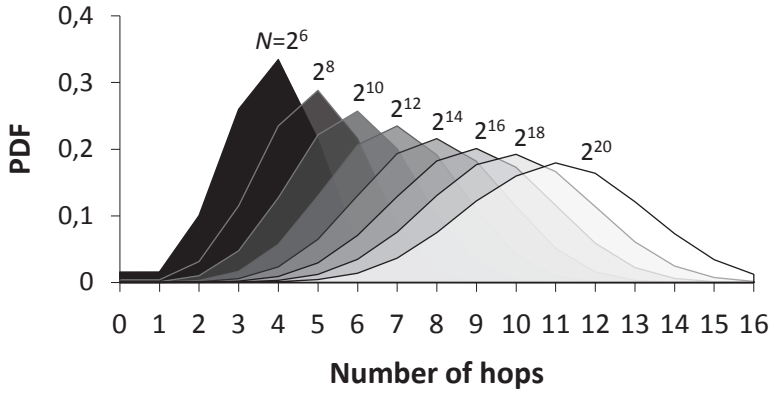


Figure 5.5.: PDFs of the lookup path length for varying number of participants N

the PDF for overlays with $N = [2^6..2^{20}]$ nodes. The values are approximately Gaussian distributed with an average of around $1/2 \log_2 N + 1$ hops. Note that the number of hops is a discrete value. Thus, the correct visualization would be a bar chart, yet, we think an area chart is more revealing here. Looking up the node responsible for an ID is used in several algorithms. For example, if a new node wants to join the network, it must at first find its successor. Also, a lookup must be initiated prior to inserting a document in the network. As a result, the number of messages and the duration of the process is distributed similarly to Figure 5.3.
A more detailed mathematical analysis of the lookup delay in Chord rings can be found in [BTG04].

Stability/Robustness In the previous paragraph, we verified that the lookup path length scales logarithmically with the number of live nodes N, whereas in this paragraph we show that the overlay stability is almost independent of N. For different overlay sizes the number of errors in the neighbor lists is nearly identical if the churn rate is the same. Figure 5.6 shows the number of nodes with errors in their neighbor lists for different overlay sizes N and varying churn rates. It proves that the number of peers with errors in their list of successors (upper 4 curves) or with wrong direct successors (lower 4 curves) is independent of the size of the overlay ring, as the different curves for the different ring sizes almost coincide. Note that dotted curves approximate the expected path between measured values and were included for the sake of clarity. Regarding stabilization and ignoring fingers, nodes in Chord have a limited local view on the network as their list of neighbors only spans a few nodes. This is why changes due to joining and leaving nodes only affect a small constantly-sized part of the overlay. The total size of the network hardly affects the capability of error detection and recovery. However, we also learn from this figure that the mean online time has a significant impact on the stability of the network. We have a closer look at the influence of the mean online time of the nodes in Section 5.2.2.
Another influence of the overlay size N on the number of erroneous pointers will be evaluated in the following. The basic Chord protocol uses a periodic stabilization algorithm, i.e., each node will periodically check whether it is still the direct predecessor

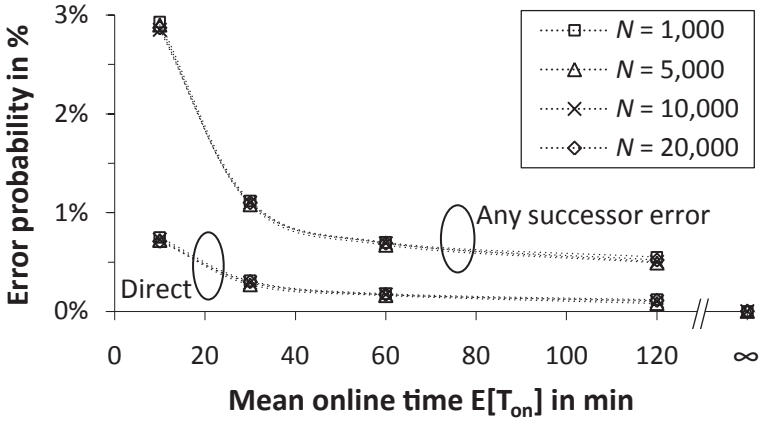


Figure 5.6.: The shorter the sessions, the less stable the overlay structure.

of its successor. If a node n_1 has joined between an existing node p and its successor s, s would return the ID and IP address of its new predecessor n_1. However, if two or more nodes (n_1, n_2, ... n_n) had joined between p and s in the meantime, s would have returned node n_n instead of the new successor n_1 of node p. In each stabilization run, one of the new nodes is integrated in the ring structure (see also Section 3.1.1). So, more than one stabilization period is necessary to repair all neighbor entries. We can calculate the probability, that more than one node joins between two neighbored nodes within one stabilization period by using series expansion.

$$P(N,J) = 1 - \left(\frac{N!}{N^J \cdot (N-J)!} \right) \begin{array}{lcl} with\ J & = & r_{\text{join}} \cdot t_{\text{stab}} \\ and\ N & = & r_{\text{join}} \cdot MTTL \end{array}, \tag{5.1}$$

where J is the average number of join events in one stabilization period and N the number of live nodes. The join rate $r_{\text{join}} = \frac{N_{\text{total}}}{MTBJ}$ is determined as the number of join events per Mean Time Between Two Joins (MTBJ). If, for example, in a network with $N = 10$ evenly distributed participants $J = 2$ nodes join, the ID of the second joining node might be situated in 9 ID ranges where no other node has joined, or in the same ID range as the first joining node. Therefore, the probability of both nodes joining within the same ID range is $1/10$.
In a sample network with a total number of one million users, a MTBJ of one day, and a Mean Time To Leave (MTTL) of two hours, we can calculate a join rate of $r_{\text{join}} = \frac{10^6}{24 \cdot 60}\ \text{min}^{-1} = 694.4\ \text{min}^{-1}$ and an average ring size of $N = 83{,}333$ nodes using Equations (4.1) and (4.3). If we assume a stabilization period t_{stab} of one minute, J will be about 694 and P about 0.945 (5.1).
Figure 5.7 shows the correlation between P and the design parameter t_{stab} for several MTTL values and a fixed join rate $r_{\text{join}} = 694.4\ \text{min}^{-1}$. We discern that, although joins affect only a local part of the overlay, simultaneous joins in that part lead to a longer transient state. The probability of such an event increases with more join events per stabilization interval (J) or with less nodes online (N). Thereby, J will be larger the higher the join rate r_{join} is and the longer the stabilization period t_{stab}. N rises proportionally with r_{join} and the MTTL.

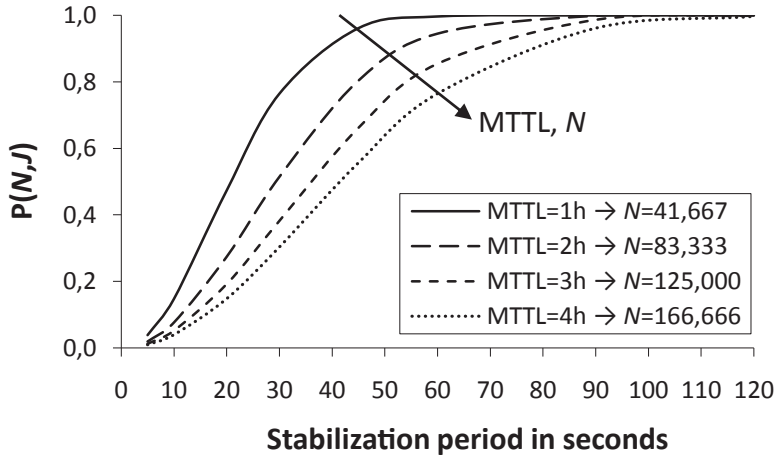


Figure 5.7.: The probability of two or more nodes joining between two neighbored nodes within one stabilization period increases with shorter MTTL values and longer stabilization periods.

Yet, joining nodes do not influence the lookup duration or lookup success, it will just take some time until the nodes are integrated in the ring structure. In contrast to that, failing nodes are much more critical, as both the lookup duration and the lookup success are negatively influenced. Furthermore, a failed node will be only detected after the absence of multiple stabilization messages. Thus, the probability that another node in the same local area fails within that time is much higher than the probability of multiple joining nodes within one stabilization period. By replacing J with the average number of failing peers within one timeout period $F = r \cdot t_{\mathrm{TO}}$, we can use Equation (5.1) to calculate the probability of multiple adjacent node failures.
In our evaluations (e.g., Figure 5.6) the percentage of nodes with errors will comprise all errors in the neighbor lists, no matter whether they are caused by joining nodes or by failing nodes. In both cases, the ring structure is not in a stable state.

Signaling traffic Structured P2P protocols also scale well with an increasing number of participants in terms of signaling traffic. Most algorithms require a constant bandwidth independent of the network size. Stabilization messages are always exchanged between a node and its neighbors, and the number of neighbors remains constant. The number of different fingers is approximately $\log_2 N$. Yet, finger entries are updated periodically and may be probed by a simple ping message. Thus, the influence of the network size N on the overhead per peer is negligible.

Conclusion Structured P2P protocols were designed with the intention to be highly scalable in large scenarios. Summarizing the above evaluation, all analyzed metrics confirm that Chord indeed scales well with the number of participants in the network. The lookup path length is in $O(\log_2 N)$ and may even further be reduced by techniques presented in Section 6.2. Furthermore, the size of the overlay has almost no influence on its stability and the required signaling overhead per peer.

5.2.2. Churn Rate

In the previous section, we became aware of the considerable impact of high churn rates on Chord. In the following, we will evaluate the influence of various design parameters under a wide range of churn rates. Starting from completely stable scenarios without any churn, we decrease the mean session durations down to 10 min. Thus, the churn rate is increased to extreme values and much stress is put on the stabilization of the overlay. Note that the churn rate is defined as the number of changes to the overlay (i.e., joins, leaves, and failures of nodes) within a certain unit of time.

Lookup path length Errors in neighbor lists and routing pointers increase the lookup path length in dynamic scenarios. Changes in the overlay structure transitionally cause wrong pointers. If a node tries to contact such a false pointer, an additional hop will be required to route the lookup. Thus, the lookup path length will be increased by one hop. In scenarios with high churn rates, lookups might even encounter several false pointers, thereby increasing the average hop count by some hops. In this context, the size of the network will influence the lookup path length: The larger the network grows, the longer the average path length without errors will be. As with each hop a false pointer is hit with a certain probability, the longer the path and the larger the probability that additional hops are required. However, the lookup path length is only increased by a few hops even in large networks.
Despite this fact, the average search delay will be significantly increased if false pointers are contacted. If a new node n has joined the network, but the node formerly responsible for that part of the ID space n_{old} is contacted, n_{old} will inform the sender of the message. Either the sender of the message, or n_{old}, then immediately forwards the message to the correct node n and the search is delayed by 1-2 TTs. In contrast to that, if a node tries to contact another node n_{fail} that has left the network, a certain timeout duration elapses until the sender will assume that the packet is lost. Thus, the packet is resent with a much larger delay (see also Section 5.2.3).
In the basic Chord protocol recursive routing is applied, i.e., each node forwards the lookup until the successor of the queried key is reached. Using UDP packets, nodes will not be aware whether the packet is received or lost. Thus, the initiator of the lookup must monitor the lookup process by using a timer. If no answer is received after a certain time, the initiator will restart the lookup. Thereby, the timeout value must be selected large enough to avoid unnecessary duplication of lookup messages, e.g., the 99^{th}percentile of the distribution of the search duration. Then, each lost packet prolongs the search by one timeout duration.
Figure 5.8 shows the results from a simulation with a timeout value of $t_{\text{TO}} = 500$ ms. A large t_{TO} value was used to clearly distinguish searches that experienced a different number of timeouts. In the corresponding scenario, 80% of nodes in a stable network with 4,000 nodes fail within half an hour. The figure shows the PDF of the total search delay. The (white) bars of the left hill indicate lookups where no timeouts occurred. Despite the extremely high churn rate, about 80.8% of all lookups are resolved without using any false pointers. The bars appear to follow a Gaussian distribution with a long tail and a mean of approximately 150 ms. The other (light gray (13.5%), dark gray

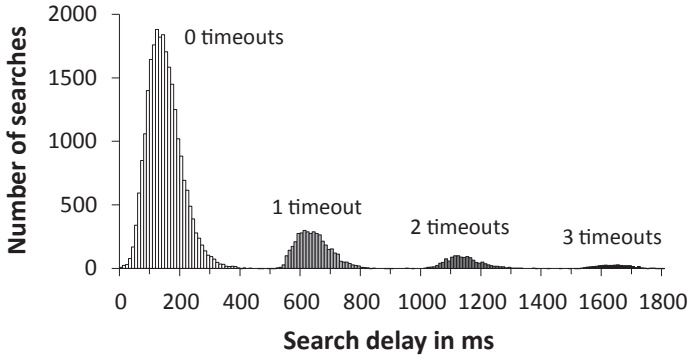


Figure 5.8.: PDF of search delay in a high churn scenario

(4.3%), and black (1.4%)) bars indicate lookups that experienced 1, 2, or 3 timeouts, respectively. The shape of the envelope of the bars is similar for all hills (grayscales), but the more timeouts, the lower and farther to the right the bars will be. By modifying the timeout value, we can shift the center of groups to the right or left. If we subtract the timeout values from the bars, then the mean of all groups will be approximately 150 ms. The total average search delay in this scenario is 281 ms. If the timeout value is increased to $t_{\mathrm{TO}} = 1$ s, the mean search delay will be 413 ms. Concluding from these results, we recommend using iterative lookups or hybrid routing (see Sections 6.2.1.1 and 6.2.2), considerably smaller timeouts can be used as each hop is monitored separately. Furthermore, applying parallel queries (see Section 3.1.3) would significantly decrease the mean search delay, as it is unlikely that all messages that are sent in parallel encounter a timeout. Using these techniques, the average search delay in this scenario can be decreased to less than 150 ms.

Last but not least, the correlation between mean online time $\mathrm{E}[T_{\mathrm{on}}]$ and mean lookup path length is visualized in Figure 5.9. The mean path length increases nearly linearly for moderate to large mean online times. An additional hop might be required due to erroneous pointers that were not yet repaired. The more errors exist, the higher the probability of using a false pointer. In contrast to that, high churn rates, i.e., small mean online times of less than 40 min, will result in a more seriously damaged overlay structure. Then, lookups might encounter multiple failures as errors sum up. We will take a closer look at the impact of the error ratio on the stability of the overlay topology in the next section.

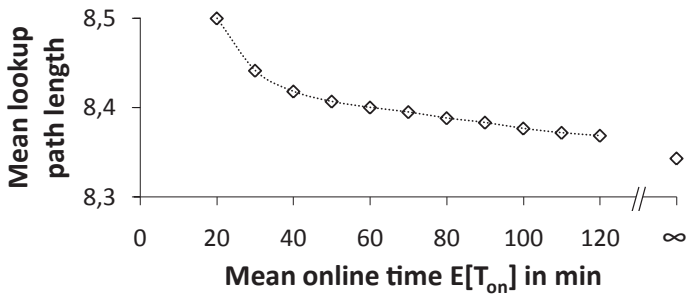


Figure 5.9.: Correlation between mean online time $\mathrm{E}[T_{\mathrm{on}}]$ and mean lookup path length

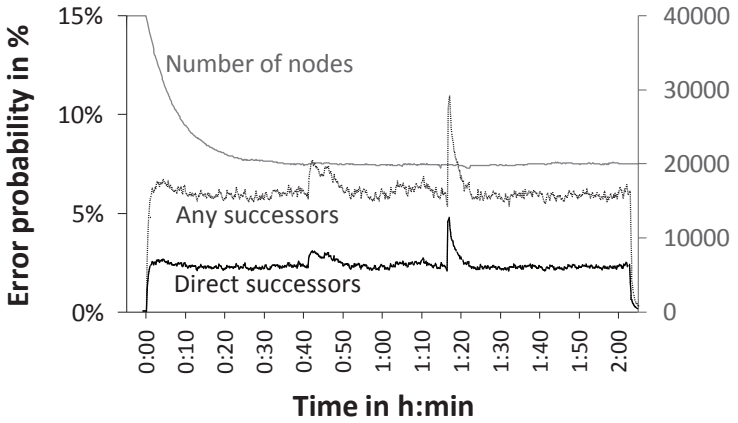


Figure 5.10.: Error probability in the course of time

Stability/Robustness Figure 5.10 plots the average number of errors in the neighbor lists for a sample scenario against time t. In an initial phase (not part of the figure) $N = N_{\text{total}} = 40{,}000$ nodes successively join the network. At time $t = 0$, the network is in a stable state with no neighbor errors. Then, in the so-called *user phase* ($t > 0$), nodes join and leave the network with mean online ($\mathrm{E}[T_{\text{on}}]$) and offline times ($\mathrm{E}[T_{\text{off}}]$) of 30 min. As previously mentioned, in our simulations nodes do not correctly leave the network, but nodes just fail without informing their neighbors. As $\mathrm{E}[T_{\text{on}}] = \mathrm{E}[T_{\text{off}}]$, the number of live nodes N levels off at around 20,000 ($= 1/2 N_{\text{total}}$) nodes.
We discern three things from this figure: First, churn causes errors in the neighbor lists. On average, circa 2% of direct successor pointers are incorrect and the error probability of all pointers is around 6.5%. The neighbor list in this scenario was intentionally chosen very small ($n = 2$) in order to accentuate the impact of churn on the number of errors. The influence of the number of neighbors on the overlay stability is evaluated in Section 5.2.3.
Second, we identify two points in time where the number of errors significantly peaks ($t_1 = 0{:}40$ h and $t_2 = 1{:}15$ h). At these times errors amplified due to some disadvantageous concurrence of joins and leaves. The overlay structure remains considerably inconsistent for several minutes. Again, these outbreaks will be less distinctive if nodes store more neighbors. A third observation is the fact that the number of errors does hardly depend on the size of the overlay. In the beginning 40,000 live nodes existed, whereas at the end of the simulation only about 20,000 nodes are online. Note that in the underlying simulation our advanced symmetrical Chord stabilization was used. Thus, the number of wrong predecessor pointers is similar to the number of erroneous successor pointers.
The influence of the churn rate on the stability of Chord is evaluated in Figure 5.6. It shows that the Chord protocol scales well in scenarios with moderate churn rates (more than 60 minutes mean session duration). Here, occasional changes due to joining and failing peers occur, yet, direct successor errors usually are repaired with the next stabilization call. Additional successor pointers show an error probability, which is about 4 times higher, than the error probability for direct successors. Here, it is more likely that several stabilization calls are necessary to repair more distant neighbors. The mean

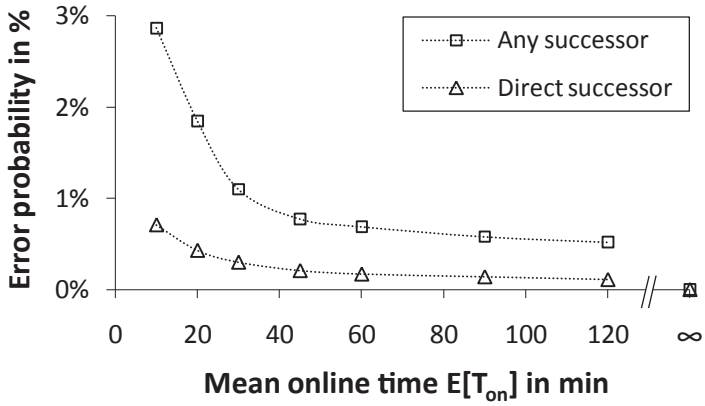


Figure 5.11.: Short mean online times significantly increase the probability of errors.

lookup path length, however, is almost directly proportional to the mean online time in that range.

In contrast to that, the basic Chord protocol is not able to scale linearly with high churn rates. Short mean online times result in a more severely damaged overlay structure. Then, errors sum up and it takes multiple stabilization calls to repair the routing state of a node. Extremely high churn rates ($\mathrm{E}[T_{\mathrm{on}}] < 10$ min) even prevent any ring-like structure and connections among peers are more random than structured. Moreover, the higher the error probability and the longer the lookup path length, the higher the probability for searches to fail. The error probability could be reduced by sending stabilization messages more frequently. However, calling STABILIZATION more often increases the signaling overhead and, as we will show in the next subsection, this approach just defers the problem. Summarizing the above, stochastical scalability is not given for Chord.

Signaling traffic The churn rate also has a significant influence on the signaling traffic. Figure 5.12 shows results from our simulations, where the signaling bandwidth for stabilization traffic is plotted against the mean online time. We are simulating an advanced Chord stabilization using symmetrical stabilization and NOTIFICATION messages. STABILIZATION messages are sent periodically to both neighbors each 30 s and include the complete list of neighbors. Thus, the required stabilization bandwidth is (nearly) independent of the churn rate. A small increase for short mean online times is explained by the fact that a STABILIZATION message is sent immediately when a new direct neighbor is detected. As the network is extremely unstable for these scenarios a lot of changes occur, resulting in an increased number of STABILIZATION messages.

Moreover, each change in the overlay structure is propagated to a small local area using NOTIFICATION messages. Notifications include information about the corresponding change. More changes in the overlay mean a higher resulting bandwidth. In accordance with other evaluations, the number of notification messages is very small for $\mathrm{E}[T_{\mathrm{on}}] \leq 60$ min, but the required bandwidth increases significantly for higher churn rates.

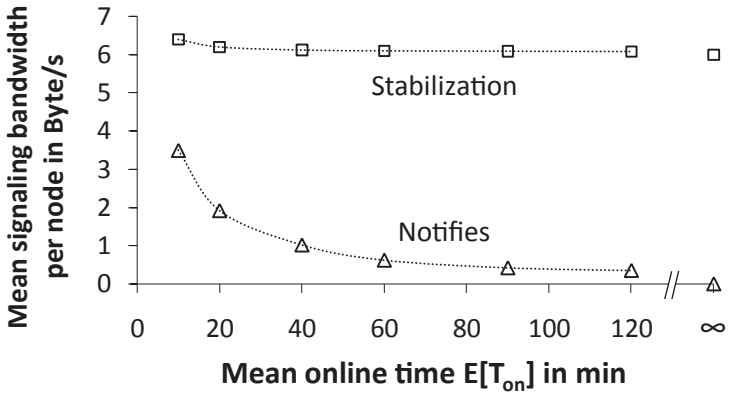


Figure 5.12.: The shorter the sessions, the more notifications are sent.

Conclusion We showed that churn has a significant influence on the performance of the protocol. Chord is able to handle scenarios with moderate and low churn rates ($\mathrm{E}[T_{\mathrm{on}}]$ less than 60 min). Yet, search duration and stability of the overlay degrade for higher churn rates. Sending more stabilization messages can help reducing this derogation, however the basic problem is not solved.
In networks with extreme churn rates maintaining the overlay structure is not feasible at all. Thus, Chord does not provide stochastical scalability. In Section 6.1, we will propose algorithms improving the stability of the overlay, despite reduced bandwidth requirements. An evaluation of various stabilization variants, including worst case scenarios where a large fraction of nodes fail simultaneously, can be found in Section 6.1.3.2.

5.2.3. Design parameters

The parameters we evaluated so far cannot be influenced directly. The number of users and their behavior may only be influenced indirectly by giving incentives to the users. In contrast to that, design parameters represent means of finding an optimal trade-off between the performance of the protocol and the related signaling overhead.

Number of Neighbors L The main reason, on the one hand, to store more than one successor is that Chord's ring structure is lost as soon as all successors of a single node fail[3]. On the other hand, the packet size of the stabilization messages grows with the number of neighbors L, as all neighbor entries are included in the stabilization packets. Furthermore, if notification messages are used, more notification packets will have to be sent, as more neighbors must be informed about changes in the overlay topology.
The authors of Chord recommend using $L = \lceil \log_2 N \rceil$ neighbors in order to be able to resolve queries w.h.p., even if half of the nodes fail simultaneously [SMK+01b]. However, if nodes fail with a high, but realistic, failure probability of $p_{\mathrm{fail}} = 0.01$, less successors will be sufficient to prevent a disruption of the ring structure with high probability [BSH05b]. In our simulations ($\lceil \log_2 20{,}000 \rceil = 15$), five successor and five predecessor entries proved

[3]The probability of such a ring break is approximated in [BSH05b]. This probability gets smaller, the more neighbors a peer stores.

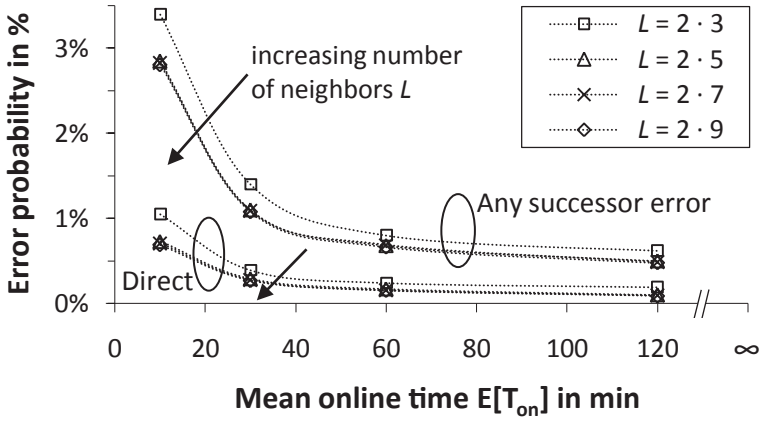


Figure 5.13.: Using more than $2 \cdot 3$ neighbors does not significantly improve the stability of the overlay topology.

to be sufficient for reducing the number of erroneous pointers to a moderate value under realistic circumstances (see Figure 5.13). No notable improvement in terms of a reduced number of erroneous neighbor entries can be discerned for larger values of L. Again, we observe that Chord scales well with the network size N (Section 5.2.1). Thus, the parameter L should not be selected as a function of N (as suggested in [SMK+01b]), but depending on the respective churn rate.

In Figure 5.13, the upper 4 curves represent the percentage of nodes with any errors in their successor lists, and the lower 4 curves represent nodes with direct successor errors. We still recommend using a slightly larger set of neighbors in the final implementations to prevent network break downs by all means. Moreover, compared with Figure 5.10 no such striking peaks in the error probability will occur if neighbor lists span at least $2 \cdot 3$ neighbors.

Neighbor Stabilization Period t_{stab} The frequency of the stabilization calls has a significant impact on the stability of Chord's ring structure. Figure 5.7 shows that raising the length of the stabilization period (t_{stab}) increases the probability that more than one join occurs within one stabilization period between two adjacent nodes. If node failures are also taken into account, the probability that two or more topology changes between closely neighbored nodes will happen in a short period of time, will be even higher.

Using a list of several predecessor and successor entries and sending notification messages (Section 6.1.2) can reduce the time necessary to repair all neighbor list entries. Still, a longer stabilization period leads to more neighbor list errors (see Figure 5.14). The uppermost curve belongs to a stabilization period of 120 s. The upper 4 curves show the percentage of nodes with any error in their successor entries, whereas the lower 4 curves, which all coincide, show nodes with a direct successor error.

Sending complete neighbor lists can repair direct neighbor errors even for long t_{stab} values almost as good as for short values. In contrast, non-direct neighbor errors increase with longer stabilization periods. Reducing the stabilization period from 120 s to 60 s, for

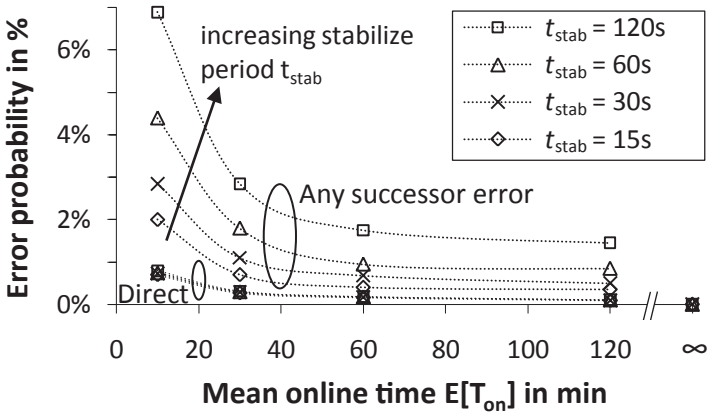


Figure 5.14.: Chord's stabilization period has a significant influence on the correctness of neighbor entries.

example, decreases the percentage of nodes with erroneous neighbor lists from 2.9% to 1.7% in a network with mean session durations of 60 min. This dependency results from the fact that a node, that sends notification messages about a new/failed node n, does not know all other nodes that have n in their list of neighbors. As mainly more distant nodes are not informed about the change, it takes a considerable amount of time until the stabilization algorithm is able to correct all entries.
Again, bandwidth limitations prevent stabilization to be called with high frequencies. If t_{stab} is halved, stabilization will require double the bandwidth and vice versa (not regarding notification messages). Thus, an optimal value for t_{stab} depends on the requirements of the system, the available resources and the user behavior (mainly MTBJ).

Finger Update Period t_{fu} We define the finger update period t_{fu} as the time wherein all different finger entries are update once. Therefore, the more fingers each node stores, the more finger updates will be performed in every finger update period. Updating fingers more frequently requires more bandwidth, but may be reasonable in scenarios with high churn rates as out-dated routing entries result in timeouts and, thus, in increased search durations. Thereby, updating fingers more often increases the efficiency of the system. However, the finger update period does hardly affect the stability of Chord's ring structure as stabilization entries (neighbors) are kept completely separated from routing entries (fingers) in our current implementation.

Timeout value Timeouts are common in P2P networks. Traffic measurements confirm that P2P traffic in particular shows a large fraction of non-malicious outbound connection attempts to non-existing hosts. As (file-sharing) peers are highly unreliable, unsuccessful connection attempts are common in P2P networks [Joh08]. Thus, finding an appropriate timeout value is another important aspect when optimizing P2P networks.
On the one hand, re-transmitting messages too early results in redundant messages and, thus, increased signaling overhead. On the other hand, waiting too long, before assuming

a node failure or a lost packet, results in increased search delays. We distinguish between type 1 errors (false positive) and type 2 errors (false negative). In this context, errors of the first kind means nodes did not (yet) consider a neighbor to be stale, although the node failed. A type 2 error is assuming a node failure while the node is still alive.
TCP connection timeouts are in the order of several minutes [Inf81]. As the overlay structure should be repaired as soon as possible, waiting for a TCP timeouts is not feasible. Also, many P2P protocols use UDP. That is why P2P applications usually implement their own timers. In our implementation we use two different timers. The *one hop timer* t_{hop} is used for monitoring a single hop, whereas the *lookup timer* t_{search} is used for re-starting the complete search. This will be necessary if the lookup terminates early, for example, by reaching a dead node with no further possibility to backtrack.
Rhea et al. [RGRK03] discuss three variants of a timeout calculation: fixed timeout values (as we use them in our simulations), a TCP-style calculation of timeouts (as suggested in Kademlia), and an estimation of optimal timeouts for each link based on virtual coordinates (like Vivaldi [DCKM04]). They show that setting all timeouts to a fixed value (of 5 s), the mean overall lookup latency is more than twice than the one of the other variants, even for low churn rates. This verifies the importance of good timeout values. In scenarios with high churn rates, good timeout values are even more important than the use of Proximity Neighbor Selection (PNS) (PNS will be explained in Section 6.2.1.2). Virtual coordinates show similar results as TCP-style timeouts for low churn rates, but they clearly outperform TCP-style timeouts for high churn rates. That is why we suggest using virtual coordinates in latency-critical applications despite their increased complexity and signaling overhead.

Conclusion We showed that design parameters are important means of adjusting the protocol to the given scenario. Thereby, finding an optimal value for most parameters requires an estimation of the user behavior and the number of participants. Binzenhöfer et al. introduce efficient and precise algorithms to estimate the current size and status of the overlay [BSH05a, BKH07].
The stabilization period and timeout values, in particular, will be able to significantly improve the performance of the system if adapted to the current system status. Then, an optimal trade-off between low signaling traffic and an up-to-date overlay structure can be made, resulting in shorter lookup durations and higher lookup success. Moreover, this section once more showed that Chord offers good functional scalability, but fails to work in scenarios with extremely high churn rates.

5.3. Related Work

During work on this thesis, structured P2P protocols became a very popular research topic and they started becoming implemented in popular P2P applications. Consequently, a lot of related work on evaluating these protocols exists. That is why we concentrate on selected publications, which are closely related to our work, and present important results.

First, most structured P2P protocol proposals evaluate the performance of the new protocol and compare it to related work. Chord was one of the first DHT-based lookup protocols, thus, a lot of evaluations exist [SMK+01a, GLR04, GBL+03, MM02, RGRK03, ZHS+04]. Yet, most authors focus on the mean lookup path length or the overall lookup latency in networks without churn. Furthermore, some simulations evaluate the lookup success rate in scenarios where a large fraction of nodes failed simultaneously. In general, their results are in line with our findings.
Binzenhöfer [Bin08] performed a detailed mathematical delay analysis of Chord-based overlay networks. In his work, he evaluated the impact of the Chord size on the overall lookup latency. For example, in a stable network with 10,000 live nodes and a mean TT of $\mathrm{E}[TT]$, the mean lookup latency is about $7\,\mathrm{E}[TT]$. Moreover, 99% of all lookups are resolved in less than $17\,\mathrm{E}[TT]$, and 99.99% of requests are answered in less than $25\,\mathrm{E}[TT]$. Dynamic networks are not evaluated.
Gummadi et al. [GGG+03] focus on the impact of various DHT designs on resilience and proximity routing. Different overlay geometries provide a different degree of flexibility in the selection of neighbors and routes. Thereby, ring and XOR geometries offer the highest degree of flexibility and outperform more complex geometries. By choosing routing neighbors with low round trip delay, the overall lookup performance is significantly improved (see Section 6.2.1.2). Moreover, being able to choose various lookup paths, the protocols are more resilient to the (simultaneous) failure of random nodes. Yet, the impact of different churn rates is not evaluated. Also, no information about the consumed bandwidth is given.
Jain et al. [JMW03] introduce a relative delay penalty (RDP), i.e., the ratio of the latency experienced when sending data using the overlay to the latency experienced when sending data directly using the underlying network. Their main finding is that better routing heuristics as well as a topology-aware overlay construction result in considerably reduced lookup latency. Moreover, improved heuristics add almost no link stress. However, again only stable networks are evaluated.
Li et al. [LSG+04, Li06] discuss the correlation between performance (e.g., mean lookup latency) and costs (i.e., mean bandwidth consumption) for different structured P2P protocols under churn. Thereby, the authors analyze the impact of tuning different design parameters. For each protocol, all possible parameter combinations are simulated (using the *p2psim* simulator [GKL+]) and a convex hull is calculated for all results. For example, the point $(x; y)$ on the convex hull might reveal the minimal lookup latency y, which can be achieved for a given bandwidth consumption x, given optimal parameter settings. If the convex hull of protocol A is below B's convex hull, protocol A wil be more efficient than B. If the convex hulls cross, one protocol will be more efficient than the other when limited to low bandwidth, while the other protocol will be more efficient if allowed high bandwidth use. Yet, a constant churn rate is applied, i.e., nodes crash and rejoin at exponentially distributed intervals with a mean of one hour. The influence of different churn rates is not considered.
In the following we summarize Li's most important results. All protocols were able to provide similar lookup success for medium to high bandwidth consumption (> 40 Byte/s per node), whereas OneHop, Chord, and Kelips [GBL+03] outperformed Kademlia and Tapestry. As expected, OneHop achieved the shortest lookup latency, followed by

Tapestry, Chord, Kelips and, far behind, Kademlia. Interestingly, only Chord was able to consume less than 7 Byte/s per node for the given parameter settings.

Using their results as basis, the authors give several suggestions for optimal DHT design choices: First, exploiting existing traffic is the most efficient way of acquiring new routing entries. Second, nodes should spend idle bandwidth for expanding their routing table. Moreover, sending parallel lookups is more efficient than sending routing entry updates more frequently. Yet, the staleness of routing entries should be bounded to a moderate value. Finally, the authors introduce Accordion, a self-tuning, bandwidth-efficient DHT-based lookup protocol that tries to consider these design choices.

The influence of Chord's stabilization interval t_{stab} on the lookup performance in dynamic networks is evaluated in [RS05]. The authors simulate a constant churn rate with $\text{E}[T_{\text{on}}] = 60$ min. They verify that more stale routing entries exist for long stabilization intervals. As a result, the probability of successful lookups drops significantly, e.g., 90% of the lookups fail when $t_{\text{stab}} = 103$ s. On the contrary, extremely small values ($t_{\text{stab}} \leq 7.5$ s) may also result in increased routing table inconsistence due to oversampling-phenomena. Moreover, [RS05] reminds us of interpreting simulation results carefully. The simulations unexpectedly showed that the average path length decreases for increasing t_{stab} values. However, this decrease cannot be explained by better performance. Instead, long stabilization intervals increase the probability of stale routing entries. In addition, the longer the lookup paths, the higher the probability to encounter multiple timeouts and, eventually, the higher the probability for the search to fail. As a result, successful lookups are likely to have short lookup paths.

Only few related work exists on the evaluation of Chord for varying churn rates. Rhea et al. [RGRK03] compare FreePastry [fre08] to Chord/DHash+[DKK+01], two real implementations. Thus, they are able to evaluate real Internet conditions instead of a simplified network model. They show that FreePastry fails to successfully complete a majority of lookup requests under heavy churn rates ($\text{E}[T_{\text{on}}] = 23$ min). In contrast to that, almost all lookups in Chord returned consistent results for this churn rate. Yet, this comes at the cost of high bandwidth consumption and long lookup latency. However, the authors perform no real parameter study, but just use the default protocol settings. We argue that various design parameters should be adapted to the observed churn rate.

A detailed theoretical analysis of Chord is available in [EAKAH04]. The authors calculate the correctness of successor pointers under churn. From that, they calculate the probability of a network disconnection (or "break-up"; see Section 6.1.4) and the fraction of stale finger entries. Moreover, the costs of finger stabilizations and lookups are computed. The authors verify their formulas by comparing them to simulation results. Amongst other things, the authors affirm our findings: The resilience of the system does not depend on the size of the system, but is a function of the ratio of stabilization rate to churn rate. For example, compared to a stable system, the performance will degrade by 60% if the ratio is 30 (e.g., 60 stabilization calls per node per hour with 2 join/leave events per node per hour). This degradation is independent of the overlay size.

5.4. Conclusion

In this chapter we showed that an application based on Chord is feasible in huge networks, as they were designed to address the scalability problem inherent to flooding. On the contrary, networks with high churn rates may be an obstacle for structured P2P protocols, as the signaling overhead grows with shorter session durations. Compared to unstructured P2P networks, structured overlays yield the advantage of being able to always find content, even if it is rare or unique.
We have analyzed the influence of different parameters on the stability and performance of the Chord structure. Parameters, like the current churn rate, have a significant influence on the network. Yet, the number of participants and their behavior can only be influenced indirectly, by, for example, creating incentives for the users. Nonetheless, knowledge of these parameters is essential for setting the design parameters of the system to optimal values. For example, tuning the stabilization period to small values is necessary for networks with high churn rates. Also, timeout values are very critical and should be adapted to the current network conditions. Thereby, a major result is that optimal parameters do hardly depend on the network size, but primary depend on the current churn rate. This knowledge helps application developers to set their parameters to optimal values for their specific application environment.
Other researchers reported simulation and analytical results similar to our evaluations. Chord maintains a simple and clear overlay topology using simple protocols. Yet, its ring geometry is highly flexible and its performance in terms of lookup path length, robustness, and signaling overhead is similar to other DHT protocols.

CHAPTER 6

Optimized robustness and performance

The evaluations in the previous chapter showed that it is important to improve both robustness and performance. In this chapter, we present related work and introduce our own contributions. Though most mechanisms can be adapted to different DHT protocols, this paper concentrates on their application to Chord.

6.1. Optimized overlay robustness

In the following, we present several modifications to Chord's stabilization protocol in order to make the resulting overlay structure more stable.

6.1.1. Related Work

6.1.1.1. Improved stabilization

[GT06] suggest adjusting the stabilization rate dynamically. The authors argue, that the churn rate varies over time, with occasional peaks [GSG02]. Thus, calling stabilization frequently causes high communication overhead in long periods of low churn rate, whereas reducing the stabilization rate means erroneous node states and a high rate of lookup failures. In order to adjust the stabilization rate, peers must estimate the overlay size and dynamism.
The authors suggest separating *liveness checks* and *accuracy checks*. Liveness check means sending a ping request in order to find stale routing table entries ($O(1)$ hops). Accuracy checks consist of a key lookup (usually $O(\log_2 N)$ hops) and are required to repair constraints imposed by the DHT protocol (e.g., in Chord, a finger entry should point to the successor of the theoretical finger position). By separating both checks, the costs of stabilization can be controlled more effectively. Moreover, the stabilization decision is taken separately for each single pointer. That way, the signaling costs can be reduced further.

For each pointer p, the probability of the entry being stale P^p_{tout} and the probability of no longer abiding the protocol-specific constraints P^p_{inacc} are calculated. The re-calculation of the probabilities is triggered both periodically and by external events (like receiving a message from a peer or the detection of a failed peer). If one probability reaches a certain threshold, a liveness or accuracy check for pointer p will be performed.
Using the *p2psim* simulator, the authors of [GT06] show that their adaptive stabilization framework is able to determine a suitable stabilization rate on-the-fly. Thereby, a target lookup rate is defined and the stabilization rate is tuned accordingly. The presented solution clearly outperforms periodic stabilization in scenarios with constant and variable churn rates. Moreover, a performance vs. cost trade-off nearly as good as the theoretical estimation given by [MCR03] could be achieved. Thus, the decision of choosing the threshold values may be supported by predicting performance and costs.
[LLD04] apply a reactive routing state maintenance strategy. Information about new nodes and node failure is piggy-backed on lookups and query replies. By using parallel lookups, nodes are able to adjust the number of messages sent. In lookup-intensive workloads the query rate, and thus the stabilization rate, is high. This also means that the number of parallel lookups may be reduced. In contrast to that, in churn-intensive workloads, a high stabilization rate is required. If the query rate is low, more parallelism will have to be added in order to increase the number of exchanged messages. Additional stabilization messages will be sent if the query rate is too low to piggy-back enough information for maintaining the overlay structure. The protocol adjusts well to different churn and lookup rates.

6.1.1.2. Security concerns

General security concerns There are different kinds of security concerns in DHT-based P2P networks. Most research so far concentrates on misbehaving nodes not implementing the protocol correctly or which simply cannot be trusted. [SM02] gives a good overview of security problems which are inherent to large P2P systems. The focus is on adversary peers which mislead legitimate nodes by providing them with false information. The authors concentrate on attacks against the routing and against the data storage system. [Sit02] presents improvements to the Chord protocol for detecting malicious nodes. First, existing verifiable properties of the protocol, like the assignment of node IDs, the monotony of lookup progress, or the ordering between predecessors and successors, should be checked. For example, Chord nodes should check whether the neighbor lists of their own successors overlap properly. Another more active check is performed by sending a random number to another node n. By replying to the challenge and echoing the number, n shows that it is alive and its Chord ID is the correct hash of its physical address information. Moreover, public keys can be introduced to sign messages. By recording messages of a suspicious node, its behavior could be checked. Also, cryptographic authentication may be realized, thus allowing nodes to identify nodes they previously communicated with.
[CDG+02a] studies attacks aimed at preventing correct message delivery in structured P2P overlays and presents defenses to these attacks. A secure routing algorithm is proposed which allows tolerating up to 25% malicious nodes while providing good per-

formance when the fraction of compromised nodes is small.

Disconnection of the overlay topology In this thesis, we concentrate on reachability and stability of the overlay in network without attacks. Nonetheless, the overlay might get disrupted and fragmented.
Binzenhöfer [Bin08] calculates the probability for a non-malicious local disconnection in ring-based overlays. A ring will be *disconnected* if any peer lost all of its L successors. Previous work showed that it is very unlikely that all successors will fail, if $L = \Omega(\log_2 N)$ and the failure probability of peers p_{fail} is less than $1/2$ [SMK+01a]. Binzenhöfer disagreed with this line of reasoning. Although it is unlikely that one specific peer gets disconnected (local disconnection), one cannot draw the conclusion that a global disconnection (at least one peer in the overlay loses all its successors) is also very unlikely. Moreover, the author argues that the failure probability of nodes should be specified for a corresponding time frame, e.g., the probability for a peer going offline within one stabilization period t_{stab}.
In the following, important results are summarized ($L = \lceil \log_2 N \rceil$). All analytical results are backed up by simulation. Obviously, the size of the network N influences the probability of a global disconnection p_{gd} in two ways: Larger overlays N cause larger successor lists L, and thus, lower probabilities for local disconnection. Moreover, the larger the overlay size N, the higher the probability for at least one node getting locally disconnected. For $N = 10^6$ and $p_{\text{fail}} = 1/2$ a local disconnection is very unlikely (10^{-6}). Interestingly, p_{gd} clearly decreases with the size of the overlay for $N \lesssim 10^2$ and asymptotically reaches a value of about 40% for larger N. This means that almost 40% of rings will get globally disconnected if 50% of nodes fail within one stabilization period. However, realistic values for p_{fail} are less than 0.1, where the probability of a global disconnection is less than 10^{-12} for $N \gtrsim 10^5$. Yet, p_{gd} is calculated within a single stabilization period. However, the longer the overlay exists, the larger rises the probability of a global disconnection within its lifetime. For example, the probability of a disconnection within one month is larger by order of a magnitude than p_{gd}.
A few papers discuss network partitioning issues: Mechanisms for discovering other partitions as well as merging multiple partitions are required. In unstructured P2P overlays merging isolated overlays is trivial. In contrast to that, complex algorithms are necessary in order to gracefully merge structured overlay networks.
[HJTW03] shows a simple merging algorithm to recover from partitions for the SkipNet protocol. Due to its path and content locality feature, SkipNet [Pug90] is able to handle disconnections gracefully. In order to discover other partitions, pointers to some well known nodes in each organization are maintained. When an organization is disconnected from the Internet, SkipNet will partition itself into several disjoint, but internally well-connected, fragments. Using the pointers to the well known nodes within the same organization, SkipNet is able to discover and merge all partitions from the same organization. If the connectivity to the Internet is repaired, the global SkipNet might be discovered by contacting other well known nodes. Moreover, nodes that lost either successors or predecessors, assume that they are the edge of a disconnect event. Thus, these edges periodically ping their unreachable neighbors to learn about the restored network connection. Merging two segments is straightforward and involves only the edge nodes

of each segment. In SkipNet, only few routing pointers need to be restored to connect segments. In contrast to that, structured overlay networks without locality features are much more "shattered" when network disconnection occurs.
[DA06] discusses the problem of merging two separate DHT-based networks. The authors show that circular DHT-based networks will not correctly operate until the merger operation completes. If two partitions of similar size merge, almost all peers will have to update their neighbor lists. Most important, a correct ring topology must be re-established in order to provide correct lookups of keys. This is a prerequisite for transferring stored ⟨key; value⟩-pairs to the new corresponding peer. Only afterward all keys may be found. From the perspective of individual peers, merging two overlays can look very similar to churn. However, from a global point of view, the magnitude of routing state changes is much higher when two partitions are merged. Usual maintenance operations may not be able to deal with the changes, thus, additional merger algorithms need to be implemented. Yet, these algorithms are not fully understood and still in experimental phase.

6.1.2. Advanced Chord stabilization

We implemented an advanced Chord variant with several small modifications compared to the basic Chord protocol. These modifications mainly affect the overlay stabilization.

Symmetrical neighbor state Instead of storing only successors, we introduce symmetrical neighbor lists. Each node additionally stores the same number of preceding and succeeding nodes. Moreover, STABILIZATION messages are sent to both direct neighbors. These messages include the complete neighbor list of the sender and do not require any reply. The messages serve two purposes. First, they act as keep-alive messages and inform the direct neighbors that the node is still alive. If no such messages are received for several stabilization periods, the neighbor will be expected to be dead and will be removed from the list of neighbors.
Second, the information included in the STABILIZATION messages almost correlates with the neighbor list of the recipient. Thus, nodes can use the information to update their own neighbor list. New nodes reported in the received list are integrated in the own neighbor list, and nodes reported as failed are removed from the neighbor list. Thereby, lists received from the successor are expected to hold more accurate information about succeeding nodes, whereas lists received from the predecessor are expected to contain up-to-date information about preceding nodes. That way, it is possible to transmit information about several changes within one message, thus speeding up stabilization. A faster stabilization also means a lower probability of errors summing up. Moreover, several nodes simultaneously joining between two adjacent nodes can be integrated in the ring with one stabilization call. In contrast to that, in Chord a separate stabilization call is required for each joining node.
Moreover, compared with the two-way stabilization in Chord, the number of messages is halved with this approach. Thus, either the signaling overhead is reduced, or stabilization might be called twice as often, resulting in more stable overlay.

Notification messages The advanced stabilization scheme is event-triggered, i.e., instead of sending only periodic stabilization messages, it uses notification messages to inform other nodes about the fact that a new or failed node was detected in the neighborhood of the node. For example, after joining the overlay and receiving a neighbor list from its successor, the new node announces itself by sending corresponding notification messages to all of its neighbors. Then, almost all neighbor lists are updated at once, without the need to wait for the next stabilization period.

On the downside, each NOTIFICATION message increases the signaling bandwidth. However, the stabilization period could be stretched in exchange, as the NOTIFICATION messages already update almost all neighbor lists. In scenarios with low churn rates, sending NOTIFICATION messages, together with a large stabilization period, increases the robustness of the topology and reduces the required signaling bandwidth (see results in Section 6.1.3 for a comparison).

Another drawback of notification messages lies in the fact, that they are sent at irregular points in time, which could lead to traffic peaks, whereas Chord's stabilization algorithm produces a constant bandwidth. If different notifications were sent within a short period of time and in a small part of the ring topology, e.g., if two or more nodes observe the same topology change at nearly the same time, the available bandwidth might be insufficient and packets would be lost. We approach this problem with the following rule: if a node s observes that its predecessor n has failed, it will send notification messages to all nodes in its list of neighbors. If any other node observes a failure of node n, it will inform the successor s of the failed node. Node s then decides whether to send notifications or not. Thus, if several nodes observe the same fail event, only one set of notification messages will be sent.

Finger update In our implementation of the protocol, we systematically update fingers f_i starting from the largest finger ($i = m$) and decreasing i with every call of the procedure. If a finger identical with the successor of n is reached ($i \approx m - \log_2 N$), smaller fingers will be skipped (as they all point to n's successor) and the finger update procedure will be restarted with the largest finger. Thus, no redundant finger updates are performed for these fingers.

Moreover, nodes do not execute FIND_SUCCESSOR() lookups for updating fingers f_i, as this operation requires $O(\log_2 N)$ overlay hops. Instead, nodes directly send ping-like FINGER_UPDATE messages to their fingers. Receiving such a message, a node n will check whether it is the successor of the theoretical finger position $f_{i,\text{theo}}$. In this case it returns a simple ACK message. Otherwise, if n knows the successor s of $f_{i,\text{theo}}$ (as it is in the list of predecessors of n) it will return s. If n does not know the correct successor s, n will return NACK and the originating peer should run a FIND_SUCCESSOR ($f_{i,\text{theo}}$) lookup to find the correct finger entry. If no response to a FINGER_UPDATE message is received, the finger entry will be considered as stale and a FIND_SUCCESSOR ($f_{i,\text{theo}}$) lookup will be necessary to find a new finger entry.

In our simulation and testbed, we additionally update our finger tables by exploiting existing traffic in the network. Each time a packet from a node in the finger list is received, we know that this node is still participating in the network and we can skip

this finger entry in the next FIX_FINGER() invocation. More traffic per node means a packet from one of our fingers is received more frequently. As a result, less signaling overhead is required for updating finger entries.
Finger list may be updated even faster by spending some additional bandwidth. A node n joining the network may copy the finger list from its successor. These pointers are very useful as a start and already provide good lookup performance, as they are close to the correct fingers of n. Moreover, node n can reversely calculate the theoretical IDs of nodes that should have fingers pointing to it. By searching the predecessors of the theoretical IDs, the corresponding nodes can be found and informed about node n.
Furthermore, we do not call FIX_FINGERS() every t_{ff} seconds. Instead, we define a larger finger update period t_{fu} and determine t_{ff} such that all different fingers are updated once within t_{fu}. Thus, we can guarantee that no finger entry is older than t_{fu} seconds.

6.1.3. Symmetrical stabilization using tokens

Analyzing the advanced Chord protocol, we found out that sometimes STABILIZATION and NOTIFICATION messages were received in the wrong order, as nodes call the STABILIZATION() function independent of each other. This results in errors in the neighbor lists, which are not repaired until the next stabilization call. As the Chord overlay network is shaped like a ring, we tried to apply some of the fully developed techniques from ring networks [LMW98] to the Chord algorithms. Instead of using Chord's disordered stabilization scheme, we use token-like STABILIZATION packets that circulate in both directions of the ring. In basic Chord, stabilization is called periodically, resulting in a constant bandwidth. On the contrary, our *Token Stabilization* consumes only a small constant bandwidth, and will send more messages if the overlay topology changes more frequently. Nonetheless, less traffic in total is required to achieve higher overlay stability [KNE05].
Each token contains a list of the last nodes that it has passed. When receiving a token, a node shifts the list by one entry, discarding the farthest node, and inserts itself at the top of the list. The token is consequently forwarded to the next node in the direction of the token. Nodes can update their successor and predecessor lists with these tokens. Thereby, the token must contain at least L nodes, with L being the size of the successor/predecessor list. If a token is received from a sender that is not an immediate neighbor, the node notifies the sender of the token about its correct neighbor, if possible. Otherwise the node will suggest a node located closer to the sender. This ensures that the token is always passed along the complete ring without skipping any node. We call this the *token redirection mechanism*. In the following, we explain our token-based stabilization in detail.

6.1.3.1. Algorithm

Node joins When a new node n joins the ring, it asks an arbitrary node b, called bootstrap, to search for its successor s. Using this information, it sends a join request to s (Figure 6.1 ①). Node s does not yet add n to its list of neighbors, but sends both a token request to its predecessor p and a counterclockwise notification token to n (Figure 6.1 ②). Node p answers the token request by sending a clockwise notification

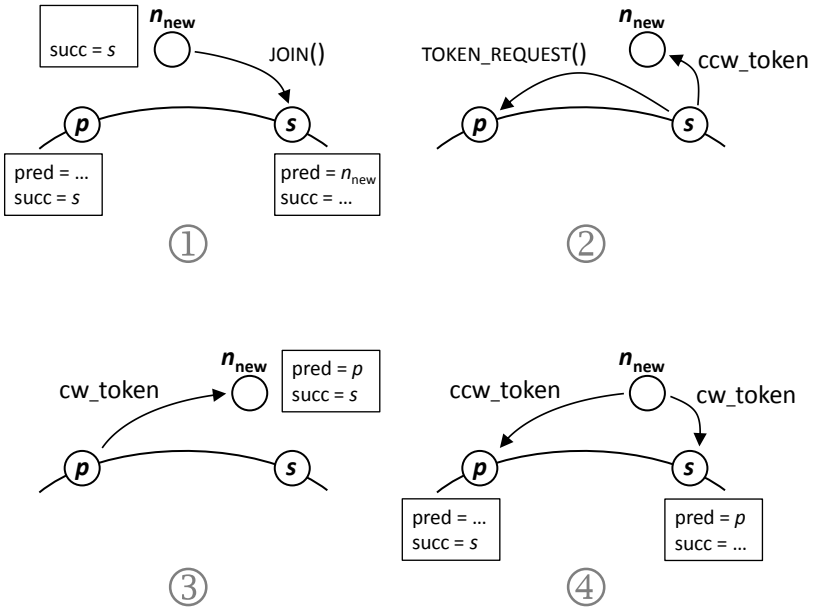


Figure 6.1.: Illustration of a *join* event using tokens.

token to n (Figure 6.1 ③). Note that the first token arriving at node n has to be queued until the second token arrives, as the second token carries necessary information for routing the first token to the next node. After both tokens have arrived at node n, it has all the necessary information to join the ring. As n forwards each token to p or s, respectively, they both add n to their list of neighbors. Node n has now successfully joined the ring (Figure 6.1 ④). Both nodes p and s forward the notification tokens to their neighbors, so that all other nodes in the ring area where n has joined, will, after a certain time, be aware of the presence of n.
As joining nodes only affect the neighbor lists of a locally limited region on the ring (L nodes in every direction) we use the following rule: A notification token will be discarded if it arrives at a node outside this region, i.e., the information carried by the token does no longer change the neighbor list of the node. Combined with the token redirection mechanism, even simultaneous joins of several nodes between two existing nodes can be handled without any problems.

Node failures Keep-alive messages will be used to verify whether both immediate neighbors are still participating in the network. Therefore, each node periodically sends a keep-alive message to both of its neighbors, and consequentially receives keep-alive messages from both neighbors. If successive keep-alive messages fail to appear, the node will have to assume that the neighbor has failed. The keep-alive messages can be small packets, which contain only the IP addresses of both sender and receiver (e.g., an ICMP packet). For this reason, they will not lead to high overhead even if sent in short periods. Additionally, node failures will also be detected if acknowledgments from any packet (e.g., FIND_SUCCESSOR()) fail to appear.

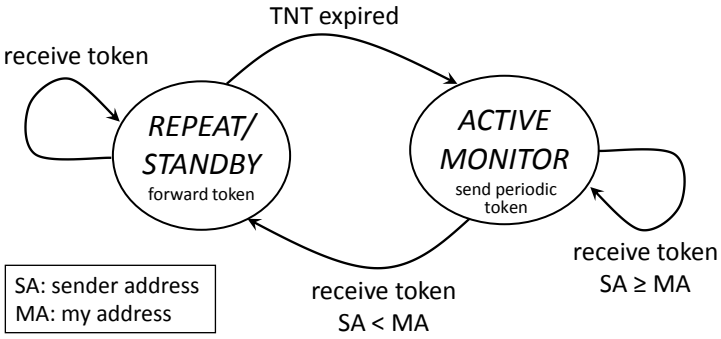


Figure 6.2.: Token state diagram: The active monitor is responsible for sending periodic tokens.

Propagation of node failures is also done with notification tokens. If any node detects a node failure, it will delete this node from its neighbor lists and send a notification token in both directions on the ring. The tokens contain the IP address of the failed node and a *node failure* flag. Again, the token is discarded as it arrives at a node outside the affected region of the ring.

Circulating tokens Not only join and fail events cause tokens to be sent on the ring. We also circulate tokens for global maintenance and information spread. Trying to keep the network load low, it is not desirable to have more tokens on the ring than needed. Therefore, a simple arbitration scheme similar in the style of ring networks is used (see Figure 6.2).

Every node is initialized in *repeat/standby* state, i.e., it forwards any received token. If no token has been received for a certain time, a Timer No Token (TNT) will expire. The node will become an *active monitor* and start sending tokens on the ring. It will switch back to *repeat/standby* state if it receives a token that has been initialized by a node with an ID smaller than its own. This mechanism ensures that after a transitional state only one node generating tokens remains. This is done independently of each other for both clockwise and counterclockwise tokens.

Joint use of tokens Tokens circulating around the whole ring can be used for a great variety of purposes. They are mainly used to maintain and repair neighbor lists but they can also carry other payload, such as status information. Circulating tokens can, for example, be used to precisely measure the size of the network and propagate the result to all nodes. Then, this information can be used for tuning various other settings in order to improve the networks performance and reliability.

There may also be scenarios where a central control station, capable of monitoring the ring and its structure, has to be implemented. In this case, a circulating token could periodically send its list of last visited nodes back to the control station, thus creating a *bird view* on the ring. A similar approach for creating *snapshots* of the overlay is introduced in [BKH07].

	Basic Stabilization	Advanced Stabilization	Token Stabilization
Stabilization period	$t_{\text{stab}} = 7$ s	$t_{\text{stab}} = 17$ s	$t_{\text{ping}} = 15$ s, $t_{\text{token}} = 30$ s
Number of neighbors	1 predecessor, 5 successors	5 predecessors, 5 successors	
Size of replication group	3 successors		

Table 6.1.: Selected design parameters related to the overlay stabilization

6.1.3.2. Analysis and simulation results

The Chord protocol works well as long as low churn rates are used. The more frequently nodes join and leave the network, the more inconsistencies and failures occur during the simulation, because the Chord stabilization mechanism fails to keep up with the topology changes. Calling the STABILIZATION function with shorter periods improves the stability of the overlay, but leads to more overhead traffic. This assumption is true for all kind of stabilization mechanisms, but other approaches lead to a higher stability requiring the same traffic.

In the following, we evaluate overlay stability and related traffic load for three different stabilization algorithms, namely a basic Chord implementation (cf. Section 3.1.1), an advanced Chord stabilization (cf. Section 6.1.2) and the presented Token Stabilization. Note that we are only interested in evaluating the different stabilization mechanisms. Thus, all simulations performed in this chapter vary only the stabilization mechanism and use the same algorithms for all other parts of the protocol.

To be able to compare the different stabilization schemes, we adjust the stabilization periods in a way that with the highest churn rate, all three variants cause approximately the same signaling overhead. These stabilization periods are then kept constant for all simulations. Consequently, we set the periods for the different algorithms to result in the same traffic load for an average online time of 5 min. Then, the average online time is varied from 5 min to infinity while keeping the stabilization period fixed. Mean offline times are set to the same values than the simulated mean online times ($\mathrm{E}[T_{\text{on}}] = \mathrm{E}[T_{\text{off}}]$). The values of other parameters that have an influence on the ring stabilization may be found in Table 6.1.

All simulations were performed with an average number of 10,000 online nodes and a total of 20,000 nodes. If any node loses all of its neighbor entries, it will try to rejoin the network with the help of the bootstrapping system. Anyway, rejoins are rare even for high churn rates. The faster the stabilization mechanism is able to repair erroneous neighbor list entries, the smaller the probability that a rejoin is necessary. In worst case simulations, where a big fraction of nodes fails at one point in time, stabilization consequently has no chance to repair neighbor lists of nodes, where all neighbors have failed during the breakdown. In such case, increasing the number of neighbor entries (e.g., to $\log_2 N$) would reduce the number of rejoins, however, at the same time, increase the consumed bandwidth.

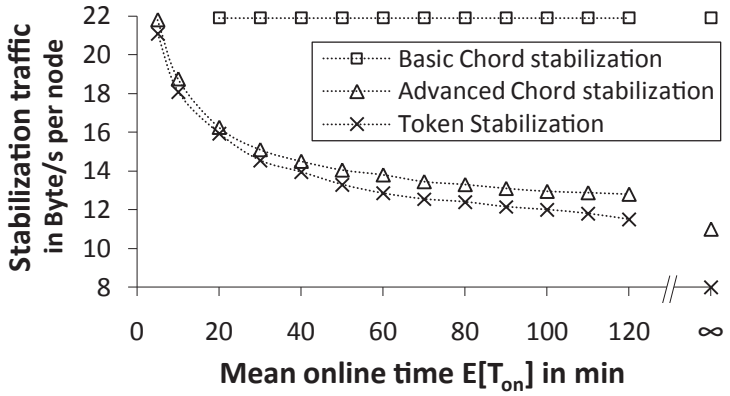


Figure 6.3.: The average stabilization traffic in Byte/s per node is constant for periodic stabilization mechanisms, but clearly increases with high churn rates for event-triggered stabilization schemes.

Stabilization traffic Figure 6.3 shows the stabilization traffic in Byte/s per node for the different protocols and the given parameters. Thereby, messages are transported in a UDP packet (28 Byte header). We can see clearly, that the average traffic is constant for the basic Chord protocol, as stabilization is executed periodically. Both the advanced Chord stabilization and the Token Stabilization have a relatively small amount of periodic traffic. Most traffic will be generated if changes in the topology are detected and notification messages are sent. The fraction of pure stabilization traffic without any notification messages can be read out at the rightmost x-values that correlate to a network with infinite online times, i.e., no churn at all.

Stability/Robustness The number of nodes with no errors in their neighbor list is shown in Figure 6.4. In this diagram, Chord achieves a slightly higher stability than our advanced Chord protocol, but we have to keep in mind that basic Chord requires an average bandwidth almost twice as high as the improved variants. If the stabilization period was re-adjusted to get the same bandwidth consumption, basic Chord would result in significantly more erroneous nodes than both improved variants. The Token Stabilization, despite its lower bandwidth requirement, outperforms the other stabilization variants. Unfortunately, we were not able to simulate the network with small online times for basic Chord, as there was too much churn to be manageable by Chord.
In a correctly shaped ring, tokens should only be received from an immediate neighbor. Due to inconsistencies caused by joined or failed nodes, a sending node may not yet be aware of these changes and forward the token to a (now) *invalid* neighbor. The redirection mechanism ensures that tokens are always delivered to the correct successor or predecessor, respectively. With this prerequisite, these tokens will automatically repair partly inconsistent rings, without the need to wait for further STABILIZATION calls.

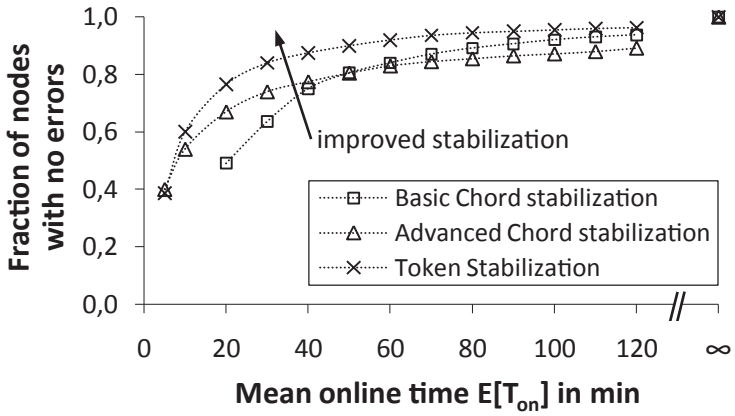


Figure 6.4.: The fraction of nodes without errors in their neighbor list decreases significantly if the churn rate increases.

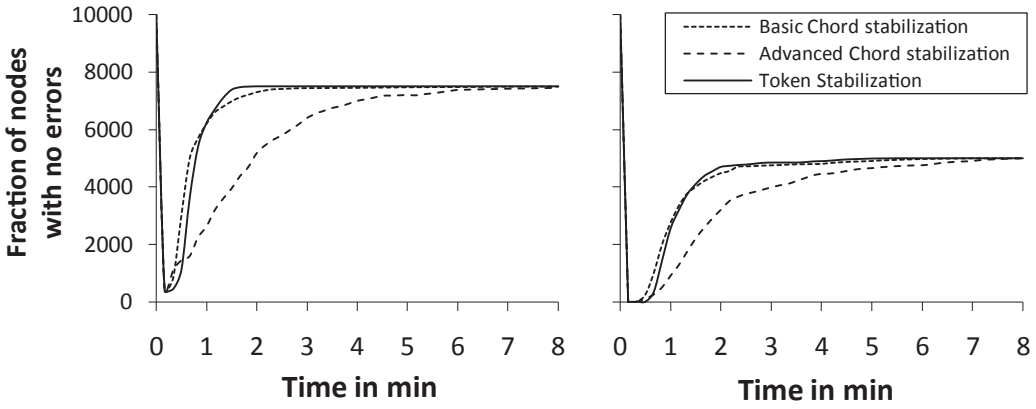


Figure 6.5.: All three stabilization algorithms are able to recover from breakdowns where 25% (left) and 50% (right) of all nodes fails simultaneously.

Worst case scenarios We also analyzed some worst case scenarios, where a big fraction of nodes fails at a certain point in time. All stabilization mechanisms are able to handle these breakdown rates, but the higher the percentage of failed nodes, the more nodes lost all neighbors and had to rejoin the network with the aid of the bootstrap servers. We assume that the bootstrap system itself does not fail and is able to handle all (almost simultaneous) rejoin requests. The need for rejoins will be reduced if each node stores more neighbors, thus, decreasing the probability of losing all neighbors.

Figure 6.5 shows two sample worst case simulations. At first, we join 10,000 nodes into one Chord ring and wait until the ring is stable. No churn takes place during the simulation, but at a certain point in time, a great fraction of nodes fails simultaneously. In the left figure, a quarter of all nodes fail at $t = 0$. We can see that almost all nodes have at least one error in their neighbor lists. Only five percent of the nodes are not affected by failed neighbors. With the given parameters, basic Chord stabilization and Token Stabilization perform quite similar. The advanced Chord stabilization falls behind, because its stabilization period is smaller and failures are detected not that fast. After about two minutes, the overlay is repaired. Now, replication mechanisms must ensure

that all content, that was not lost with the failed nodes, is moved to the new replication groups. However, a stable overlay is a crucial requirement for shifting content to the correct nodes. The right figure shows a simulation where fifty percent of the nodes are affected by the breakdown. In this scenario, the failures led to errors in every remaining node. Three minutes after the fallout, an almost perfect ring-shaped overlay could be re-established. This demonstrates how effective all three stabilization algorithm can handle even extensive simultaneous node failures.

6.1.3.3. Conclusion

In networks with low churn rates the Chord stabilization algorithm is able to maintain a correct ring structure. As churn increases, the stabilization period of every node has to be reduced to handle all changes in the overlay topology, resulting in a higher network load. In this chapter we proposed a token-based stabilization mechanism that copes with node joins and failures in significantly reduced time, without generating additional overhead. In general, we suggest that a stabilization algorithm will be more reliable and efficient if nodes do not call STABILIZATION() independent from each other but in the correct order, as it is given for our token-based approach.
Future work includes modifying the token mechanism to work together with a symmetrical Chord variant, where lookups can be routed in both directions. We believe that both our token-based stabilization, as well as a symmetrical FIND_SUCCESSOR() algorithm (e.g., S-Chord (see Section 6.2.1.4) can support and benefit from each other.
In our implementation of the Token Stabilization, we set our focus to the analysis of the algorithms performance in terms of lookup times and stability (measured in neighbor list errors). As expected, the overhead caused by the token stabilization algorithm is much lower than for the basic Chord algorithm.
However, there are some improvements that have not been implemented yet. We could try to reduce the overhead caused by keep-alive messages. Therefore, a timer is set for both neighbors (timeout value $\approx$ stabilization period). When a node n receives any packet from another node n' the timer is reset. Instead of sending periodic keep-alive messages, keep-alive packets would be sent only if the timer expired, i.e., no packet was received from that node for a while. The measured round-trip time and packet loss probability from ping-pong packets could also be used to estimate suitable timeout settings, i.e., when should a packet be re-sent or when can we safely assume that a remote peer is down.
Furthermore, the token forwarding mechanism is not based on a send queue yet. Therefore, more than one notify token, traveling in the same direction, could be queued at one node: To reduce traffic, queued tokens should rather be merged and the resulting single token should be forwarded. Especially in the case of high churn rates, the probability that more than one token is held at a node that currently experiences timeouts at the next node is significant. For these cases, the amount of necessary traffic could be reduced.
Furthermore, we suggest a neighborhood discovery mechanism. If a node n_i detects that its adjacent node n_{i+1} has failed, it will try to forward the token to the next node n_{i+2}. Taking the amount of time needed for failure discovery into account, forwarding a token over several failed nodes may require a lot of time. Still, the number of packets

sent is considerable: transmission will be tried for several times until the destination is considered as stale. Then, the process continues for the next destination. In case of a transmission error, it may be faster to probe the whole neighborhood for responding nodes and then selecting the best node that responded to the probe. This could be done using a simple ping-pong mechanism that would involve only small packets. Especially in case of high percentages of nodes failing at the same time, for instance due to a power loss in a local network, this mechanism could provide both increased stability and decreased routing overhead.

6.1.4. Repairing disrupted or partitioned overlays

The ring structure of Chord is especially vulnerable to attacks since each disruption of the overlay can cause a disconnection of the overlay ring. In the worst case, the network is split into two separate rings, which are not aware of each other. Such disconnections cannot only be caused by malicious attackers but also by churn, i.e., by the frequency at which new users join and leave the system. There are different proposals of how to handle churn in a structured P2P network [RGRK03], however, it is impossible to entirely avoid failures in the system.
To increase the stability of Chord-like P2P systems, we present a novel self-protecting approach which is able to detect possible problems at an early stage and to react accordingly. However, while it is certainly important to try to prevent attacks and failures, one cannot entirely avoid them. As experience shows, distributed systems will encounter failures and consequently should be designed for it. Therefore, we additionally aim at the recovery from failures rather than at failure-avoidance alone. Our self-repairing algorithms are able to automatically detect disruptions and will initiate redundant countermeasures to re-establish the structure of the overlay [KB06].

6.1.4.1. Security issues (and their detection)

Loss of all successors Erroneous successors can lead to erroneous lookups. In the worst case, they can even cause disruptions in the overlay topology. Chord's ring structure can encounter two different kinds of serious damage. First, if a peer loses all of its successors, the ring will break (see Figure 6.6). Second, the ring structure may fragment in two halves or two separate sub-rings. In this section we discuss different offensive scenarios that result in such overlay disruptions. In particular, we identify different threats and their impacts.
Due to churn or a well directed Denial-of-Service (DoS) attack on at least L successive nodes on the Chord ring (with L being the size of a neighbor list), node p preceding the affected part of the ring, will no longer be able to contact any of its successors. In fact, it can be shown that the probability to lose all successors due to churn is not negligible [BSH05b]. After sending several ping messages to these non-responsive nodes, a timer expires and the nodes are removed from the successor list of p. Consequently, the ring structure breaks as depicted in Figure 6.6 ($L = 3$). Node p can easily detect such a break in the ring as soon as it discovers its list of successors to be empty.

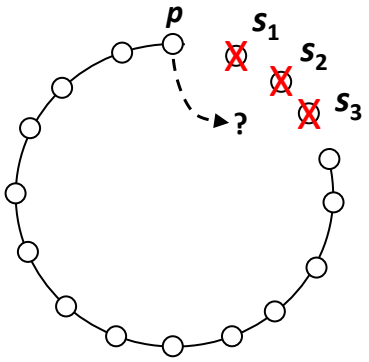


Figure 6.6.: Concurrent failure of p's successors

As Chord lookups are only performed clockwise, p is not able to search for its new successor. If it started a query FIND_SUCCESSOR(ID(p) + 1), the lookup would first search the predecessor of ID(p) + 1. This implies that the lookup would come back to node p. Therefore, performing a rejoin is no feasible solution for this kind of disruption. The consequence of a loss of all successors is a transient routing state. That is, some nodes might no longer be reachable, while others might not be able to forward search queries correctly.

Partitioning of the overlay Another threat to the network is a partitioning of the overlay structure, i.e., the ring fragments in two or more separate overlays [SMK+01a]. This scenario will be likely to occur if gateways between physically separated networks fail. Chord's stabilization mechanism updates all erroneous successor pointers. After a certain time, two or more consistent sub-rings emerge. Lookups can still be performed correctly in all fragments, but due to the partitioning, not all data stored in the original overlay is still available in all sub-parts. A company running a global DHT application, for example, will no longer be able to access all data stored in the DHT, if one plants access point fails. Running a DoS attack on nodes that have a critical location in the physical network is sufficient to damage the whole network.
In mobile ad hoc networks (MANETs), network splits are even a common issue. The overlay is likely to be partitioned due to frequent and fast node movement, node failures and MANETs that are out of each others range. Successive splits without any countermeasures finally result in many sparely populated subnets.
Mechanisms (see Section 6.1.4.3) reducing the risk of a ring split exist, but are not able to avoid them entirely. Moreover, the above examples clearly indicate that the overlay protocol must be able to recover from a partitioned network. In the following, we will introduce some efficient mechanisms, which are able to detect and merge sub-rings.

6.1.4.2. Recovery

Recovery from a partitioning of the overlay If an overlay is split into several partitions, but the nodes are still connected in the physical network, it will be likely that there are still fingers in each partition pointing to nodes in other parts of the network. Lookups will pass through different sub-rings and finally return an erroneous result. However, nodes can use their finger entries and information gathered during lookups to learn about nodes in other partitions. By inserting all other appropriate nodes into their own successor list, the separate rings will merge automatically.
However, in scenarios where no physical connections between separate sub-rings exist, as pictured in the previous section, the partitions cannot be merged. Fingers pointing to nodes in other parts cannot be contacted and the algorithm that updates the fingers removes these entries after a while. If the physical connection between two rings is re-established, nodes will not learn about the other ring by themselves.
A simple approach is to run a periodic rejoin at every node. In doing so, each node starts a lookup for its direct successor via the bootstrap service. It will not make any difference if the bootstrap mechanism is a local or remote cache of available nodes or a single server. The proceeding is similar to a node joining the network. If the bootstrap service by chance returns a node from another partition, this information will be suitable to merge both rings. In our simulation environment, we observed that two rings will merge within a few minutes, if at least one node learns about any node in the other ring. The main drawback of this approach is that each node periodically has to perform a rejoin operation and therefore stresses the bootstrap service. Shorter rejoin periods mean faster detection of different rings but cause also more load on the bootstrap mechanism. Therefore, this algorithm will not scale for huge overlay networks.
In a more efficient variant of this mechanism only one well-defined peer in each ring, for example, the peer with the smallest ID, sends a periodic message to the bootstrap server. A peer will assume that it has the smallest ID if its predecessor pointer has a higher ID. The bootstrap server will notice separate rings as soon as it receives messages from different peers. By informing all involved peers, a merging process can be started. As only one peer per ring sends periodic messages, this variant is highly scalable. Also, the frequency of performing this algorithm could be increased significantly, resulting in a much faster detection of sub-rings.

Recovery from loss of all successors If the ring breaks due to a failure of L successive nodes, the peer preceding the disrupted part of the ring will not be able to contact any of its successors. As discussed in the previous section, a standard lookup for the successor of the node will also not return any result. We present a modified search algorithm that is capable of performing lookups regardless of disruptions. The key functionality is an algorithm that will redirect a lookup request in counterclockwise direction if the lookup skipped one or more nodes. We call this method *redirection mechanism*. It can also be used in normal operation when a lookup request skips the keys correct successor and is received by the wrong succeeding peer. As soon as a peer recognizes that a search overshot its target, it applies our redirection mechanism. A node n can easily detect that a lookup did overshoot the correct successor, if it receives a lookup message for a key k

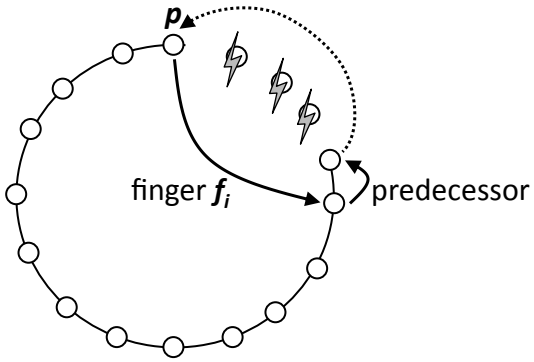


Figure 6.7.: Automatic disruption recovery, initialized at the beginning of a break.

located between the initiator of the lookup and itself, but it is not k's successor. Using its predecessor, node n is able to redirect the message toward the correct successor s. The message may also be redirected over several nodes until the correct node is reached.

In case of an open ring, node p, preceding the disruption, can use the *redirection mechanism* to repair the overlay disruption. It simply sends a lookup message for its own ID + 1 to the closest available finger. In general, this is the smallest finger that is situated outside the former successor list of the node. Then, as shown in Figure 6.7, this node will redirect the message in counterclockwise direction until the message arrives at the other end of the disruption. This peer no longer possesses a valid predecessor as all of its preceding peers have failed. Therefore, it assumes that the initiator of the message is its new predecessor. For the same reason it assumes that it is responsible for the searched ID and answers the lookup. The initiator of the message inserts the sender of the answer in its successor list and initializes a stabilization procedure with its new successor. The disruption is repaired and correct routing is re-established.

If a peer also stored enough predecessors (i.e., maintaining a symmetric neighbor list), a similar recovery mechanism would be suitable to be used by the peer at the end of the disruption. A node that has lost all of its predecessors initiates a lookup for its own ID (see Figure 6.8). The lookup will traverse the ring until it arrives at the node at the beginning of the disruption. If this node is not aware of the disruption yet, it tries to forward the lookup message to one of its successors. As all successors have failed, the node will receive no acknowledgments and, after a certain period of time, delete all successors from its list. A node that is aware of the disruption, as it has lost all successors, inserts the sender of the lookup message into its own list of neighbors. It then forwards the lookup to its new successor and starts stabilizing with it.

If both nodes at the edges of the broken part of the ring run a recovery algorithm, the disruption will be detected faster and can be repaired with higher probability. In the worst case, one redundant lookup message is routed through the ring. Note that if symmetrical routing [MCVR03] is applied, the redirection mechanism will no longer be necessary. Both nodes at the edges of the disruption can initiate a symmetrical lookup for their own ID.

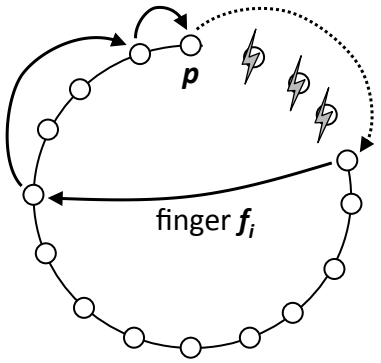


Figure 6.8.: Automatic disruption recovery, initialized at the end of a break.

Recovery using token based stabilization The Token Stabilization introduced in the previous section could also be used for recovery. In normal operation nodes are in the *repeat* state, or in other words, they forward all incoming tokens to the next node on the ring. A node situated at one end of a broken ring does no longer receive token messages from the disrupted part of the ring. Therefore, it changes to the *active monitor* state and starts generating periodic tokens. All tokens contain ID and IP address of its initiator. Acknowledgments prevent tokens from being lost as nodes fail. The token will be passed through the ring until it reaches the peer at the other end of the broken part. There, the information about the initiator of the token can be used to repair the ring disruption. The initiator is inserted into the empty neighbor list and a stabilization process is initiated. However, this algorithm does not scale well with the ring size as the token is forwarded from node to node, requiring N times the average transmission time to circulate the ring.

6.1.4.3. Avoidance

Regarding the correctness of the Chord overlay, we observed that the probability of disruptions can noticeably be reduced by some simple modifications to Chord's stabilization algorithm. To avoid a disruption in the ring structure, nodes should prevent an empty successor list at any rate. If the number of entries reaches a critical minimum, nodes will be able to fill their successor list with any active node they known (e.g., finger entries) or learn about (e.g., from received messages). The redirection mechanism will still guarantee correct lookups.
In order to increase the correctness of the overlay structure, nodes can also increase the frequency of sending stabilization messages. The more often stabilization messages are sent, the more up-to-date neighbor entries are. We suggest an adaptive mechanism, which will increase the stabilization frequency if the number of known successors shrinks or if the overlay structure is measured to be more dynamic. Additionally, the size of the neighbor list can be adjusted adaptively to the current churn rate in the network. However, the more often stabilization messages are sent and the more successors are included in the messages, the more bandwidth is required. Nodes should pay attention to their current resource usage to avoid performing a DoS attack on themselves.

Most important, we recommend that nodes should make use of all information they can gather about other nodes. They should check whether the sender of any message they receive fits in the list of neighbors or fingers. If the sender of the message is already part of a list, the Time Last Seen (TLS) for this entry will be able to be updated. Thus, a node learns about new nodes without the need to wait for the next stabilization. Additionally, the necessary bandwidth for checking the availability of finger entries can be reduced. No finger update is performed for recently seen finger entries.
We also suggest to send information about failed nodes to all neighbor nodes. Thereby, nodes can replace failed neighbors much faster. Yet, we dissuade from blindly trusting in information received from other nodes, as this information may be incorrect. Nodes should verify the information, for example, by sending a ping message to the responsible node. If recursive routing is applied, nodes will exchange a lot of messages with their successors and fingers. Therefore, nodes are aware of failed contacts much faster.
Finally, we recommend using a symmetrical Chord variant with symmetrical neighbor lists (see Section 6.1.2) and symmetrical routing [MCVR03]. Additional symmetrical fingers can be achieved by exploiting the existing overhead (see Section 6.2.3). Symmetrical routing enables nodes to search in both directions, so a simple disruption in the ring can be avoided.

6.1.4.4. Conclusion

Disruptions in structured P2P overlays cannot only be caused by well targeted attacks against specific nodes but also by churn, i.e., by the dynamic behavior of the participating peers. In this section we presented efficient mechanisms to actively prevent the loss of the overlay structure in both scenarios. Using simple modifications to the standard algorithm, a peer is able to exploit the existing overlay traffic to improve the stability of the overlay. We also introduced a self-repairing mechanism, which is able to detect a disruption in the overlay network and to take appropriate countermeasures. The algorithm was designed to be redundant in order to speed up the healing process and to improve its success rate. Finally, we introduced a scalable solution to detect the existence of disjoint overlay partitions and showed how to automatically recombine them. Applying our modifications to a Chord-like P2P system can greatly improve its security and robustness.

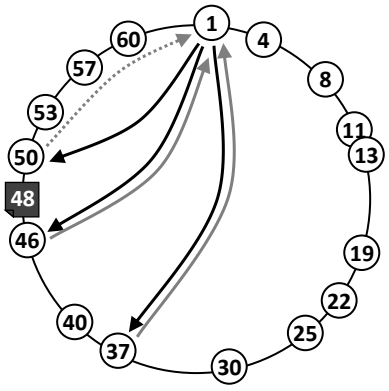


Figure 6.9.: Iterative routing: The nodes communicate only with the originator.

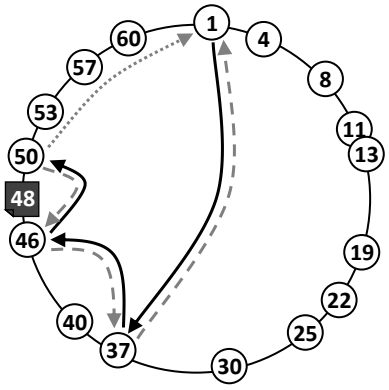


Figure 6.10.: Recursive routing: Each node forwards the query to the next node.

6.2. Optimized lookup performance

In the second part of this chapter we present several modifications to Chord's lookup protocol with the aim of reducing the mean lookup path length and search duration, as well as the lookup success rate.

6.2.1. Related Work

6.2.1.1. Iterative vs. recursive lookups

Iterative routing The base Chord [SMK+01a] and Kademlia [MM02] algorithms use iterative routing (see Figure 6.9). When looking up a certain key k, the originator sends a FIND_PREDECESSOR(k) request to its finger, which is closest to the predecessor of the key. This node answers with the closest finger it knows preceding the key. After receiving this information, the originator sends the request to this node and waits for an answer. In each step, a node that is closer to the searched key is obtained and the procedure is continued until the predecessor of the key is reached. The predecessor finally returns the successor of the key, which in turn is responsible for the key. As a result, the originator of the lookup is involved in all steps of the query, it handles the complete traffic, and contacted nodes return the next hop to the initiator.

This proceeding has two main advantages. First, the originator can easily keep track of the lookup route. In case of a failure, the search can be continued somewhere next to the absent node, thereby skipping all previous hops. Second, as each hop is monitored by the initiator, the search timeout can be set to a short value, e.g., the 95^{th}percentile of the current Round-Trip Time (RTT) distribution of the network. Then, only 5% of the messages will be retransmitted unnecessarily, despite a relatively short timeout value. Thus, when the lookup fails due to an absent node, the originator is soon aware of the failure and probably can continue the lookup at the previous hop.

Recursive routing Among others, Dabek [DLS+04] describes recursive routing and its benefits. Each node forthright forwards the query to the next node until it reaches the predecessor p of the key. Node p directly returns the successors of the key (i.e., its own successor list) to the originator (see Figure 6.10).
Recursive routing also yields some advantages. During the lookup, each node forwards the request to one of its fingers and receives an acknowledgment. Thus, nodes permanently monitor their fingers without the requirement of sending extra FIX_FINGER messages. Additionally, TCP connections to the fingers can be set up as all lookups are sent along one of these routes. Moreover, by sending the acknowledgment and forwarding the lookup at the same time, the latency of each hop is halved. Simulations show that recursive lookups are on average 40% faster than iterative ones will be.
However, the originator of the lookup is not involved in the lookup process. It initiates the query and finally receives the answer. Thus, the originator must keep a global search timer $t_{\text{search}} \gg t_{\text{hop}}$ in order to detect failed lookups. If this timer expires, the lookup is restarted.
The timeout t_{search} can be calculated by estimating the search duration. The Transmission Time (TT) distribution of packets, which are transmitted over k overlay hops is calculated by a k-times convolution of the distribution of a single TT. Then, q% of packets will result in timeouts if the timeout is set to the $(1-q)^{\text{th}}$ percentile of the overall TT distribution. In network simulators TTs are often modeled using a Negative Exponential Distribution (NED) (see section 4.2). The Probability Density Function (PDF) of a NED has the form

$$f(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & , x \geq 0, \\ 0 & , x < 0. \end{cases} \tag{6.1}$$

Its expected value is $E[X] = \lambda^{-1}$. The k-times convolution of NEDs is known as the Erlang-k distribution.

$$f(x;k,\lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \qquad \text{for } x \geq 0 \tag{6.2}$$

The corresponding Cumulative Distribution Function (CDF) is

$$F(x;k,\lambda) = 1 - e^{-\lambda x} \sum_{n=0}^{k-1} \frac{(\lambda x)^n}{n!} \qquad \text{for } x \geq 0. \tag{6.3}$$

Assume a network with $N = 100{,}000$ nodes and an average TT of 100 ms, i.e., $\lambda = 1/100$. The parameter q is set to 95, i.e., less than 5% of the packets are resent because their timer expired. Applying the Chord protocol, the maximum routing path length in a stable environment is $2\log_2 N$. With iterative routing, the hop timeout t_{hop} must consider both the transmission of the packet as well as the transmission of its acknowledgment, i.e., $k = 2$. By numerically solving the above equation we find out that a timer value of at least 475 ms must be used.
Using recursive routing, packets travel up to $k = \lceil 2\log_2 N \rceil + 1 = 35$ overlay hops before they return to the initiating peer. Therefore, a search timeout value t_{search} of at

least 2,550 ms is required in order to assure that less than 5% of the packets traveling the maximum number of overlay hops are retransmitted. This timeout value must be increased for larger networks as the number of overlay hops depends on the number of participating peers.
The average search duration will be able to be decreased if local timers (timeout value t_{hop}) are initiated in each node that forwards the packet. After the packet has been forwarded to the next node, an acknowledgment is sent to the node the request was received from. If the request or the acknowledgment is lost, the local timer will expire significantly earlier than the global timeout of the originator. Thus, the failure is detected faster and the lookup is delayed less. However, in some rare cases, the request might still get lost, for example, if the node currently monitoring the packet fails and the packet is lost at the same time.
As we showed earlier, current P2P networks face high churn rates. Therefore, routing failures are common, timeouts occur quite often, and packets must be retransmitted frequently. Thus, short timeout values are essential to provide fast lookups.

6.2.1.2. Route and neighbor selection

In basic DHT protocols, neighbors are often selected according to a strict deterministic rule. Other neighbor selection heuristics relax this strict computation and neighbors may, for example, be selected from a deterministic ID range close to the exact neighbor position. In basic Chord the i^{th} finger is exactly the successor of $\text{ID}(n+2^{i-1})$ $(i \in [1,m])$. In a more flexible implementation, however, all nodes in the interval $[n+2^{i-1}; n+2^i[$ are candidates for the i^{th} finger [SMK+01b].
DHT protocols usually use a greedy routing scheme, i.e., messages are forward to the neighbor, which is "closest" to the destination, where in return "closeness" is commonly defined as distance in the ID space. In contrast to that, the term "proximity" is defined as the overlay TT between two nodes. Proximity route/neighbor selection techniques consider both metrics. The next hop is selected by striking a balance between making progress towards the destination in the ID space and choosing the closest routing table entry according to the network proximity [CDHR02]. Thereby, short latencies usually indicate short physical paths, thus reducing the overall traffic in the underlying network and better reflecting the underlying IP topology.

Long Lifetime Node Selection (LNS) Kademlia bases its node selection on node liveness information. As we showed in Chapter 5, the distribution of node lifetimes is usually heavy-tailed, i.e., the probability of a node being still online for a certain time interval is a function of its current session duration. In Kademlia live nodes are never removed from the buckets, thus nodes with long lifetimes are preferred.
A similar approach is introduced in [ZY06]. Here each node includes its own lifetime in every packet it sends, and it learns about the lifetimes of its neighbors by incoming messages. The authors also introduce a routing metric, which considers both liveness probabilities and proximity. They show that timeouts are a significant component in overall lookup latencies. LNS results in more stable neighbor links, and thus, less timeouts. Moreover, by applying LNS the selected neighbors are less dynamic. Thus, the

stabilization interval may be increased, resulting in a reduced signaling overhead.

Proximity Neighbor Selection (PNS) With PNS network latency is used as the metric by which to choose between neighbor candidates. Thus, most hops on the lookup path have shorter delays, whereas lookups are still resolved in $O(\log_2 N)$ hops [GGG+03]. As a result, the overall lookup duration is significantly reduced.
Gummadi et al. state that sampling 16 candidates for each neighbor results in almost optimal proximity in most cases. Dabek et al. even argue that using PNS, the total average lookup duration will stay close to 3δ in a network with a mean one-way delay of δ independent from the network size N. They also found out that the last hops in a lookup actually dominate the total lookup duration. These hops use neighbors with small indexes, i.e., small ID intervals where neighbor candidates are selected from. However, small intervals mean few potential candidates, and thus non-optimal PNS.
Instead of randomly sampling nodes to find new neighbors, the authors of Pastry suggest to copy neighbor entries from the neighbors of a node (*neighbors of a neighbor*) [RD01]. In Tapestry's nearest neighbor algorithm [ZHS+04], each node samples those nodes that have the same neighbors as itself (*inverse neighbors of a neighbor*). Both techniques are motivated by the fact that nodes are clustered, thus we can expect that neighbors with short latencies also provide nearby nodes. A detailed comparison of various PNS techniques can be found in [RGRK03].
The authors of Chord published a technical report [SMK+01b], which extends the basic protocol description in the proceedings. Instead of implementing a pure finger table, Stoica et al. use a *location table* that is a cache of all recently discovered nodes. Finger entries are pinned in the location table. Other entries may by replaced by nodes that are closer in terms of network latency. As these nodes are also likely to be physically close neighbors, *node locality* is introduced. Instead of merely using fingers, nodes from the location table, which are close predecessors *and* are close in the network, are used.
PNS is also implemented in a Chord-based Cooperative File System (CFS) [DKK+01]. Here, more flexibility in selecting the next hop is achieved by storing multiple pointers in each finger interval. A cost model is used to determine the best neighbor:

$$C(f_i) = d_i + \overline{d} \cdot \text{hops}(f_i)$$

$$\text{hops}(f_i) = \text{ones}((f_i - k) \text{ AND } mask)$$

$$mask = \text{NOT } 2^{(m+1-\log_2 N)}$$

$\text{hops}(f_i)$ is an estimate of the number of hops that would remain after contacting finger f_i, $\overline{d}$ is an estimate of the network's average latency, and d_i is the measured latency to finger f_i. $\log_2 N$ approximates the mean number of *significant* high bits in an ID, i.e., adjacent nodes are likely to agree in these bits, but not in less significant bits. N is an estimate of the number of live nodes in the system [BSH05a]. A higher density of nodes means adjacent nodes agreeing in more high bits. The remaining number of hops ($\text{hops}(f_i)$) is estimated by the number of ones (ones()) in the significant high bits of the binary distance Δ between f_i and destination k. Consequently, each 1 approximately corresponds to one overlay hop of span 2^j, with j being the position of the 1 in Δ. The

significant high bits can be extracted by using a bitmask *mask* with $\log_2 N$ ones in the high bits (similar to a subnet mask). Then, the total cost $C(f_i)$ will be an estimate of the remaining lookup duration if finger f_i would be used. After computing the costs for all fingers f_i in the corresponding finger interval, the candidate with the minimum $C(f_i)$ is selected as next hop. Hereby, selecting only fingers from the largest finger interval still preceding the destination assures a mean hop count of $O(\log_2 N)$.

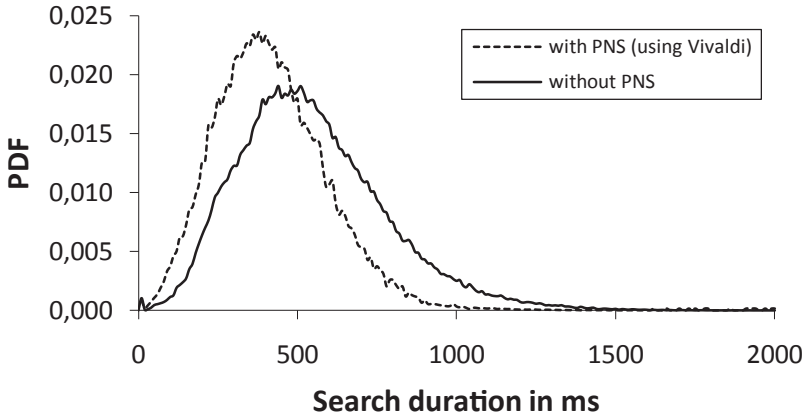


Figure 6.11.: Search duration with and without PNS

[KLKP08] evaluates the impact of PNS and Proximity Route Selection (PRS) in Kademlia. The authors show that the mean lookup latency is approximately divided by three. Furthermore, Internet Service Providers (ISPs) may reduce costs as 40% of lookup traffic does not leave their domain. [CZK05] compares PNS in Tapestry and Chord. Both networks show similar performance gains in lookup performance. The simulations also show that the probability of meeting randomly failed nodes in a lookup is not reduced, since neighbor selection does not reduce the mean lookup path length. Moreover, using PNS an uneven distribution of the incoming node degree of a node is observed, as nodes near the center of the network and nodes with high capacity are preferred. As a consequence, the network will be more vulnerable to attacks, because attacking nodes with high in-degree affects network connectivity more severely.
We also integrated a basic variant of PNS in our Chord implementation. Figure 6.11 compares the measured mean lookup duration in simulations with and without PNS. While the average number of hops is still in $O(\log_2 N)$, the overall lookup time is significantly shorter, because a close node can be selected as the next hop of a search request. Using PNS, the average lookup time in this scenario could be reduced to 80% of the lookup time using the original Chord protocol. Note that in our simulations new neighbors were only discovered by exploiting incoming messages. We did not use any of the above mentioned mechanisms to actively search for new neighbor candidates by random sampling or copying (inverse) neighbors of a neighbor. Thus, the mean lookup duration is still larger than $3\delta = 240$ ms in our simulations.

Proximity Route Selection (PRS) In PNS the selection of the next hop is based on two independent steps. First, a set of entries in the routing table is selected (e.g., a finger

interval in Chord or a bucket in Kademlia). Then, one of these pointers is selected based on measured or estimated TTs. In contrast to that, PRS tries to find an optimal next hop in one step.
PRS was first proposed in CAN [RFH+01]. Here, messages are "forwarded to the neighbor with the maximum ratio of progress" by trading off the number of hops in the path against the network distance traversed at each hop. Therefore, *any* known neighbor closer to the destination in the ID space is a valid next hop. Consequently, the applied heuristics for PRS are more complicated than algorithms used in PNS. CAN, however, presents no explicit metric for choosing the next hop.
Usually, PRS heuristics try to select routes with a similar hop count. Imagine two Chord nodes n_1 and n_2 that are $O(N)$ distance apart. The first node n_1 knows approximately $\log_2 N$ exponentially distributed fingers, which can be used for routing a message to n_2. The next node in the lookup path can chose between approximately $(\log_2 N) - 1$ fingers, and the i^{th} node ($i < \log_2 N$) in the lookup path will roughly know $(\log_2 N) - i + 1$ potential fingers. As a result, approximately $(\log_2 N)!$ different routes exist. It is important to note that all of these routes show a total path length of $O(\log_2 N)$ hops [GGG+03]. Chord uses greedy routing, thereby making most progress toward the destination in the first hop (largest span), and skipping exponentially decreasing parts of the ID space in succeeding hops. This corresponds to sorting the hops in decreasing order of their spans. Yet, this route is rarely the shortest route in terms of latency. However, sorting the $\log_2 N$ hops in any other order and taking each of the different spans just once will reach the destination in $O(\log_2 N)$ hops.
[GGG+03] presents heuristics for ring, XOR, and hypercube geometries that offer such flexibility. Therein, the next hop is selected from a subset (candidate set) of the known neighbors, which is selected in such a way that the routing path length is usually not increased. In the following, we concentrate on the heuristic for circular overlays. Similar to Chord, fingers with exponentially increasing spans are stored in the routing tables, with the i^{th} finger of node n being in the interval $[n + 2^{i-1}; n + 2^i[$. The PRS algorithm at first expresses the distance to the destination in the ID space in binary notation. If there is a 1 in the i^{th} position of this binary notation, the i^{th} finger will be selected as a candidate. Finally, the candidate with the shortest latency is selected as next hop. This algorithm may also be coupled with PNS, and the closest nodes measured by latency are selected as fingers. However, the additional benefit is quite limited [GGG+03].
Furthermore, another PRS heuristic might be even more flexible by allowing to traverse multiple hops of smaller spans instead of one hop with a large span. As a result, the mean hop count is increased. Yet, if the sum of the latencies of the small hops is less than the latency of the larger span (e.g., because the involved nodes are within the same local network), the total search duration will be decreased nonetheless. However, with only local information available, we believe that such an approach will be difficult to realize.
Additionally, shorter RTTs can be achieved by using a cross-layer communication channel between the physical layer and the application layer [GSK06], thus avoiding inefficient routes in the physical layer. In Section 6.2.4 we present a Fuzzy-based Route Selection (FRS), which is able to combine PNS, PRS, LNS, as well as any further suitable heuristics.

6.2.1.3. Parallel lookups

Sending asynchronous parallel lookups is another possibility to reduce the overall lookup latency, as the impact of timeouts can be reduced. Copies of the lookup may proceed while other copies encounter stale routing entries and have to wait for the timeout to expire. Moreover, lookups may be exploited to learn about new neighbors as well as the liveness of existing neighbors. [LSM+05] shows that exploiting this information is more efficient at lowering latencies than frequently checking existing neighbor liveness or active exploration of new entries. In literature, several proposals for parallel lookups exist.

Parallel iterative lookups Kademlia [MM02] sends requests to β nodes in parallel. For each lookup, the initiator stores a list of nodes that are "closest" to the queried key k. Kademlia uses closeness in the ID space as the distance metric, however, also latency could be used. In the first step, the initiator n selects β nodes from its own buckets and sends parallel lookups to them. These nodes search their buckets for the closest β nodes they know and return the information to n. After receiving an answer, node n copies the received nodes in its list. Then, it selects the closest node from the list that has not yet been queried and sends a message to it. Thus, all the time β lookups are executed in parallel. This proceeding is continued until the node is found, which is responsible for k. Parallel iterative lookups are also used in EpiChord [LLD04]. By exploiting lookup traffic a large routing state may be maintained, thus, achieving $O(1)$ lookup performance under lookup-intensive workloads. The authors state that both mean path length and mean lookup latency are reduced by a factor of 3 when issuing 3 parallel asynchronous lookups. Moreover, additional network information (e.g., notifications about new or failed nodes) is piggy-backed on lookups and query replies. Thus, under reasonable lookup traffic, EpiChord is able to keep its routing entries up-to-date without additional signaling overhead. Thereby, the authors observe a synergistic relationship between a large routing state and parallel lookups. On the one hand, a larger routing state means shorter lookup paths, and thus, less lookup messages. That is why EpiChord can afford to issue parallel lookups without generating excessive amounts of lookup traffic. On the other hand, more information may be exploited from parallel lookups, thus, a large routing state may be maintained.
As discussed earlier, the initiator n of a lookup will be in control of the complete lookup traffic, if iterative routing is used. Thereby, n can easily adjust the amount of parallelism, and thus, the number of lookup messages. Node n can also influence the paths lookups take, thereby, preventing duplicate messages. Moreover, n learns useful information about other nodes participating in the system. However, compared to recursive routing, nodes communicate less frequently with nodes from their own routing table. Thus, these entries are more likely to be stale.

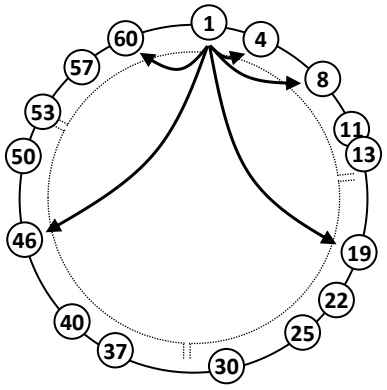
Parallel recursive lookups Hence, [LSM+05] introduce parallel recursive lookups in their Accordion protocol. Here nodes may exploit lookup traffic to learn useful information about their neighbors. Yet, recursive parallel lookups are difficult to control and nodes might receive identical copies of a lookup. As a result, more messages are required to achieve the same amount of parallelism than compared to iterative parallel lookups.
In Accordion, a self-organizing mechanism is used to control the number of parallel lookup messages. Similar to the above variant, the originator of a lookup selects β_n nodes, whose IDs most closely precede the queried key k, and sends parallel lookups to these nodes. The copy, which is sent to the node whose ID is most closely to k, is marked with a "primary" flag and is given high priority.
Receiving a copy, a node n directly forwards the request to β_n (≥ 0) nodes which are closer to the destination. Thereby, a large β_n might be chosen, if enough free bandwidth is available, whereas β_n will be small or zero, if the bandwidth of n is not sufficient. Nodes independently adjust their β_n value so that they stay within their bandwidth budget. Thereby, the prediction of future bandwidth needs is based on the past lookup rate. In addition, the parallelism will be increased if more exploration messages than the number of lookups that have passed through this node have been sent.
If n receives a lookup that it has already seen in the recent past, it will drop this message. However, the "primary" copy must be forwarded in any case. Thus, the lookup marked as 'primary' travels the same path a non-parallel lookup would have taken, while other copies decrease lookup latencies and increase information learned.

Using multiple hash function Another solution for parallel queries is introduced in CAN [RFH+01]. Here, *multiple hash functions* may be used to map content to multiple points in the coordinate space. Thereby, the availability of the content is increased, as it is stored on multiple peers. This kind of replication also offers the possibility of *parallel queries*, i.e., all replicas can be queried in parallel. Thus, the average query latency is reduced. This improvement can easily be implemented in other DHTs.

6.2.1.4. Symmetrical Chord routing (S-Chord)

The Chord finger geometry provides nodes with a lot of routing information in close neighborhood and with little information about farther nodes (see Figure 6.12 on the left). No pointers to preceding nodes exist. In the figure, the right part of the Chord ring (seen from the node on top) is well covered with fingers, whereas the left part is blank. Therefore, the idea of extending the finger table to the complete ring comes to mind. S-Chord [MCVR03] proposes to keep the same number of fingers to be comparable in performance with Chord, but organizes the finger table in two approximately symmetric sides (Figure 6.12 right).
Like in Chord, strictly deterministic fingers are maintained, i.e., there is no flexibility in choosing the i^{th} finger $f[i]$ of node n:

$$f[i] := \begin{cases} \text{successor}^{+}(n + 4^{i-1}) & , i \in [1, g], \\ \text{successor}^{-}(n - 4^{2g-i}) & , i \in [g+1, 2g]. \end{cases} \tag{6.4}$$



index i	ID $n_1 + 4^{i-1}$	Node successor$^+(ID)$	ID $n_1 - 4^{2g-i}$	Node successor$^-(ID)$
1	2	n_4		
2	5	n_8		
3	17	n_{19}		
4			49	n_{46}
5			61	n_{60}
6			0	n_{60}

Figure 6.12.: Sample finger distribution in S-Chord for node n_1.

Thereby, successor$^+(x)$ and successor$^-(x)$ denote the first node succeeding x going clockwise or counterclockwise, respectively. By using $g = \lceil log_4 N \rceil$ and 4^x operations, the same number of theoretical fingers as in Chord is obtained.

This finger assignment posseses the following properties. The theoretical finger positions are symmetrical to the axis n and $n+N/2$ (*finger table symmetry*). Thus, the actual finger table is nearly symmetrical. *Routing entry symmetry* states that for any two nodes, n_1 and n_2, if n_2 has a finger to n_1, then n_1 will have a finger to n_2 or very close to it. Also, it is very likely that the lookup path length from node n_1 to n_2 is equal to the path length from n_2 to n_1 (*routing cost symmetry*). However, both paths are not equal (i.e., there is no routing symmetry).

Like in Chord, each node n in S-Chord is responsible for storing all keys in the interval $]\,\text{predecessor}(n); n]$. However, due to the symmetrical nature of S-Chord, we suggest to define a more symmetrical responsibility, like the interval $[\text{predecessor}(n); \text{successor}(n)]$. Thereby, intervals of adjacent nodes overlap with each key being assigned to 2-3 nodes. Thus, a replication of $\langle \text{key}; \text{value} \rangle$-pairs (see Section 3.2) is introduced in the protocol. A higher replication grade can easily be defined by increasing the size of the interval.

Also, the "responsibilities" of the fingers are adapted to the symmetrical lookup. In Chord, fingers are situated at the beginning of the interval, whereas in S-Chord fingers are situated within the interval. The authors of S-Chord define the responsibility of a finger i starting from the half way point between it and the $i-1^{th}$ finger, and ending at the half way point between it and the $i+1^{th}$ finger (dotted lines in Figure 6.12). If a queried key is in the responsibility of the i^{th} finger, the lookup will proceed with this finger.

In [MCVR03] the authors prove that the maximum path length in a stable environment and a fully populated ID space is $\lceil \frac{3}{4} \log_2 N \rceil$ hops, compared with $\lceil \log_2 N \rceil$ hops in Chord. They also simulate the protocol and observe circa 10% shorter lookup paths. In Section 6.2.3 we introduce Freebie Fingers, i.e., additional routing information obtained by exploiting finger update traffic. A comparison of the routing performance of Chord, S-Chord, and our Freebie Fingers variant can be found in Section 6.2.3.

6.2.1.5. Chord#

Most DHTs use cryptographic hash functions in order to achieve good load balancing. Yet, a perfect cryptographic hash function, which receives two input values that differ in only one bit, returns two IDs that differ in half of their bits [GMB03]. Thereby, locality is removed. As a result, range queries are not applicable with DHTs. However, only locality assures that consecutive keys are stored on logically neighboring nodes. Thus, if range query functionality is required, locality preserving hash functions (key-order preserving functions) will have to be used.
However, locality preserving hashing results in a load imbalance, as keys are not uniformly distributed in the ID space. Some parts of the ID space are densely populated, whereas almost no keys are stored in other regions. Thus, additional load balancing mechanisms have to be applied (see Section 3.3). Usually, these mechanisms shift nodes and their responsibilities in order to achieve an equal number of keys per node. As a result, these equally loaded nodes are no longer evenly distributed in the ID space.
In common DHTs the pointers are calculated in the ID space, for example, in Chord the i^{th} finger of node n points to the first node that succeeds n by at least 2^{i-1}, with $1 \leq i \leq m$. Yet, applying this function to an ID space with unevenly distributed nodes raises two main problems. First, more than the average number of fingers point to the node at the end of an above average sized ID region. Consequently, this node must handle significantly more finger updates and search requests than nodes which are only responsible for a small part of the ID space.
Also, the distribution of the search path lengths is changed for the worse. Searching for keys in densely populated regions requires more hops, because there are many nodes in that region. In a DHT providing a $O(\log_2 N)$ routing, each doubling of the number of nodes corresponds to approximately one additional hop. That is, if all nodes are situated in one quarter of the ID space, the mean lookup path length for IDs in that part will be increased by two hops. In contrast to that, the mean lookup path length for IDs in sparely populated parts will be reduced. Thereby, each hop still approximately halves the distance to the key in the ID space, but as nodes are unevenly distributed, the distance "in nodes" is not halved. Thus, the variance of the lookup path length, as well as its maximum value are increased.
That is why in Chord# [SSR05, SSR08], pointers are computed using the set of nodes $\mathcal{N}$. The i^{th} finger of node n_j points to node n_{j+2^i} $(1 \leq i \leq m)$. Thus, fingers cross $1/m, \ldots, 1/8, 1/4, \ldots, 1/2$ of the *nodes* in the ring. Due to the logarithmic placing of the fingers, it is possible to lookup a key in $O(\log_2 N)$ hops. In their work, the authors of Chord# suggest a pointer placement algorithm that takes the actual node distribution into account. As a consequence, Chord# keeps the same routing performance as Chord, but allows range queries and active load balancing.
The node distribution can be described by a density function $d(x)$. Mercury [BAS04] uses random walks to estimate $d(x)$, however this comes at additional costs. Hence, Chord# avoids the calculation of $d(x)$ by using a recursive formula for placing the pointers, i.e., the i^{th} pointer of node n is the $i-1^{th}$ pointer of node n's $i-1^{th}$ pointer. On the contrary, the density function $d(x)$ will not significantly change over time, and thus could be transmitted to a node during its bootstrap process, whereas placing the pointers by

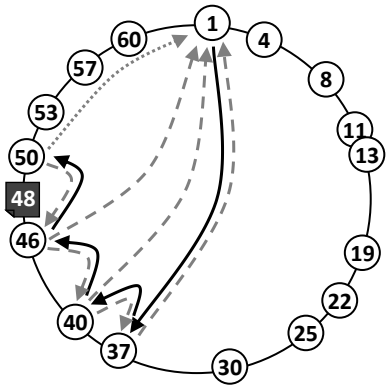


Figure 6.13.: Hybrid routing: Each node forwards the query to the next node and sends an acknowledgment to the originator.

using the recursive formula must be periodically repeated as nodes frequently join and leave the system.

6.2.2. Hybrid routing strategy

In Section 6.2.1.1 we discussed iterative and recursive routing. Based on this discussion, we propose a hybrid routing strategy. It possesses the low latency of recursive routing in error-free circumstances, while providing the fast failure recovery of iterative routing [Kun05]. Basically, we perform recursive routing with a fundamental modification: every node forwarding a packet sends an additional acknowledgment back to the originating peer (see Figure 6.13). Although this comes at additional overhead costs, a lot of useful data can be gained from it.
Like in iterative routing, the originator acquires information about the path of a packet through the overlay. Using our hybrid routing, a peer that performs a lookup initiates a local timer for each hop that the packet travels. In the first step, the timer value must consider the average RTT. In succeeding steps the timer value can even be reduced to the q^{th} percentile of the TT distribution. Therefore, the initiator can react quickly to failures and lost packets. Additionally, it has information about what went wrong and where and how to retry the lookup in a way that is more likely to succeed. If, for example, a packet is lost due to a failed node, the lookup might be continued at a previous node on the routing path. By including information about the failed peer, the new routing path can bypass this peer.
Besides enabling fast failure recovery, the acknowledgments received by the initiator are useful for further reasons. As mentioned earlier, the initiator learns about other peers participating in the network, including a RTT measurement. This will be especially useful if PNS is applied and any node in a finger interval may be used for routing. These peers can possibly complete or improve the finger table of the initiator. This information can also be used to improve local estimates about the current status of the overlay, like the current size of the overlay [BSH05a], as the more samples the better the estimate.
Like in recursive routing, each node also receives acknowledgments from the fingers it

	Iterative	Recursive	Hybrid
Description	Initiator n handles complete lookup process.	Lookup is recursively forwarded.	Recursive lookup monitored by initiator.
Advantage	Node n learns about many nodes, but this is not exploited in basic Chord.	Fingers are verified frequently, as only local information is used.	Nodes learn about other nodes *and* fingers are verified.
TCP	$\Rightarrow$ TCP unfavorably due to many new connections	$\Rightarrow$ TCP connections feasible as only little changes in the finger tables occur.	
Timers	Short t_{TO} feasible.	Short t_{hop}, but large t_{search} required.	Short t_{hop} and t_{search} feasible.
Mean path length	$\log_2 N$	$1/2 \log_2 N$	$1/2 \log_2 N$
Overhead	$2 \log_2 N$	$(2 \log_2 N) + 1$	$(3 \log_2 N) + 1$

Table 6.2.: Comparison of iterative, recursive and hybrid routing

forwarded the packets to. This information is used to verify the finger entries of the node. Using a Time-to-Live (TTL) field for each finger entry, fingers have only to be actively validated by sending a FIX_FINGER message, if they have not been used for lookups within a certain FINGER_UPDATE period. This reduces the overhead caused by the maintenance protocol of the ring.

Comparison of iterative, recursive and hybrid routing Table 6.2 compares all routing variants. Summarizing, the hybrid routing variant combines the advantages of recursive and iterative routing. The average lookup duration is shorter due to the recursive routing. Moreover, the initiator of a query receives notifications about the current lookup status, thus, being able to quickly react to failures and sending a new search request on a node-disjoint backup path. Due to the notifications, the initiator also learns about other peers and can use this information, e.g., to complete and improve its finger table. On the downside, the hybrid routing variant generates a higher number of signaling messages. Yet, this is compensated as search messages can be exploited to update finger entries.

6.2.3. Freebie Fingers

In Section 6.2.1.4, we showed that the lookup path length will decrease if symmetrical fingers are used. Therefore, in S-Chord fingers in both halves of the ring are maintained. However, the same number of fingers as in Chord is used to be comparable to Chord in terms of overhead costs. In this section, we propose an improvement to Chord, which requires the same overhead, but doubles the number of symmetrical fingers [KS06]. Thus, on average, each hop can travel closer to the queried key than in S-Chord, resulting in even shorter lookup paths.
Our proposal extends the routing information each Chord node stores by a list of all nodes that have a finger table entry pointing to the node. This information comes for

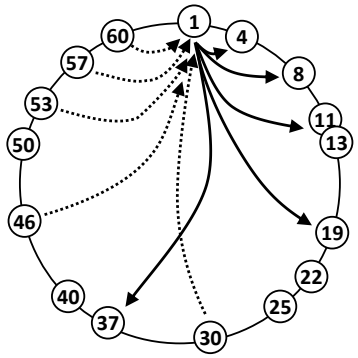


Figure 6.14.: Sample finger distribution in a Chord ring (solid lines) and additional routing information from nodes that store a finger table entry pointing to the node (dashed lines).

free without any additional overhead cost by exploiting the existing signaling messages necessary for the stabilization of the finger tables. This is why we call these back-pointers *Freebie Fingers*[4].

This proposal was presented in July 2005 [KS06]. Interestingly, independent of our research Rossi and Stoica presented an almost identical idea in November 2005 [RS05]. They will call a node n reverse finger (or *regnif*) of another node f, if f is a finger of n.

In both works, all nodes store a cache of freebie finger entries. Each time a FIX_FINGER request is received, the initiator of the request is added to the cache. Entries in the cache are removed after expiration of timer t_{TLS}. Each entry is associated with a TLS value. Entries will be removed from the list, if the entry has not been updated within a certain time t_{TLS}. Thereby, we avoid storing nodes no longer participating in the network, as using these entries would lead to timeouts and therefore significant lookup delays. Reasonable values for t_{TLS} are in the order of one FIX_FINGER period. Thus, the probability that using a finger results in a timeout is similar for freebie fingers and standard fingers.

Nodes in the freebie finger cache are distributed counterclockwise in approximately the same way, as the nodes in the finger table are distributed clockwise, i.e., more nodes in close neighborhood and only a few nodes farther away. Figure 6.14 shows the finger distribution for a node in a sample Chord network (solid lines) and the finger table entries from other nodes that point to the node (dashed lines).

In Section 3.1.1, we mentioned that each node in a Chord network with N nodes on average stores about $F = \log_2 N$ different fingers. Thus, about $F' = F$ reverse fingers exist on average (see Figure 6.15). TLS entries for finger and freebie fingers are updated each time a packet is received from the corresponding nodes. Nodes stay in the cache of freebie fingers as long as they are used frequently. Especially if recursive or hybrid routing (see Sections 6.2.1.1 and 6.2.2) is used, packets will be mainly forwarded to fingers. Thus, each time f receives a request from n, f implicitly knows that n is alive and is still pointing to f. However, due to churn, there may be entries in the cache of a

[4]The noun *freebie* refers to an article or service given or gotten free, usually provided as part of a promotional scheme.

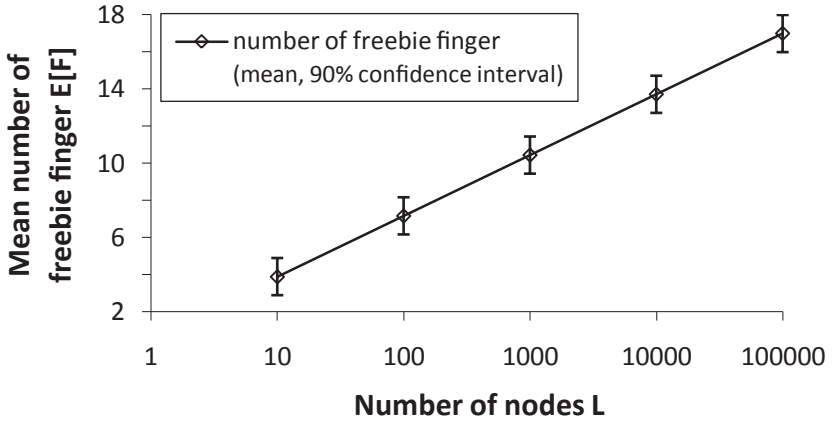


Figure 6.15.: Average number of freebie fingers for different network sizes

node n that actually do not store node n in their finger table anymore. Therefore, the size of the cache should be limited to C entries. In [RS05] the authors show that in a 4,096-node network a cache size of $C = 100$ yields good results. On average, each node can benefit from a routing table with at least $2F$ entries. Costs for updating the finger entries are still equal to Chord. In contrast to that, in S-Chord (see Section 6.2.1.4), the number of fingers in the first half of the ring had to be halved in order to be comparable to Chord.

Using freebie fingers is simple. All fingers, including freebie fingers, are searched for the finger that is closest to the searched ID k. Thus, a maximal decrease of the remaining ID space is achieved and the average number of hops is reduced. Note that with symmetrical routing, the query may be forwarded both clockwise and counterclockwise. Performance can be increased slightly by applying the symmetrical routing algorithm proposed in S-Chord [MCVR03].

Like in Chord, the peer succeeding an ID is responsible for this ID. Therefore, unidirectional Chord routing requires looking up the peer that is preceding the ID. This takes up to $2\log_2 N$ hops, and $1/2\log_2 N$ hops on average. One additional hop is required to contact the peer that stores the required data. In our simulations, we modified the basic Chord routing in such a way that peers, which are querying for an ID they are responsible for do not initiate a search, but immediately answer that query. Moreover, symmetrical routing may approach the queried ID from counterclockwise direction. Consequently, we will answer the lookup, if the succeeding peer is reached and we will not route the query via the preceding peer.

The authors of [RS05] also suggest a different *degree of trust*. In each step, the closest finger f, as well as the closest regnif r to a key k, are determined. If r is closer to k then r will be used. In the case of a timeout, f is used as alternative. Additionally, r is vetoed from being used for routing until the status of r is updated or the timer of r expires and r is removed from the cache. Thereby, w.h.p. it is assured that using regnifs is not worse than the basic Chord routing.

Nodes *leaving* the network inform their neighbors, thereby avoiding any inconsistencies in the structure. Using freebie finger, each node n can additionally inform all nodes,

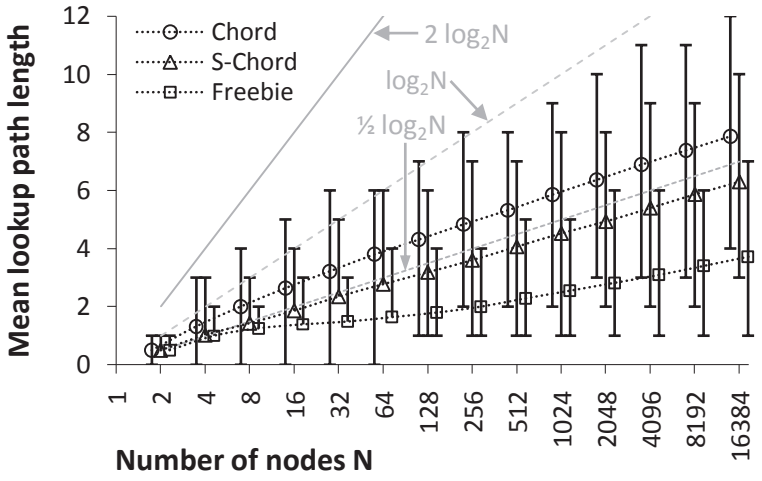


Figure 6.16.: Average path length for varying network sizes and routing strategies

which have a finger pointing to n, when they leave network. Thus, lookups are no longer forwarded to n and timeouts are avoided during the lookup.

Results Our freebie finger approach benefits from two different facts: symmetrical routing and an increased number of fingers. Thus, lookups can be resolved faster, as fewer hops are traversed. More important, by reducing the number of hops, the probability of contacting a failed peer is decreased. We verify the expected improvements by simulating different routing strategies. In the following we assume a stable 128-bit network. For each x-value we simulated 1,000 different networks and initiated 10,000 lookups for random keys in each simulation run, resulting in a total of 10^8 lookups. The error bars show the $1^{\text{st}}$and 99^{th}percentiles.

Figure 6.16 shows the correlation between the number of nodes N participating in the network and the resulting average path length for Chord, S-Chord, and our symmetrical freebie finger routing. Chord lookup path length is discussed in detail in Section 5.2.1. The curve corresponding to S-Chord is about $1/2$ hops below $1/2 \log_2 N$, i.e., the average path length is around 20% shorter than for the Chord protocol. Also, for the simulated network sizes, the 99^{th}percentile of the path length is 1-2 hops less compared with Chord. Freebie fingers offer an even higher benefit. Compared with Chord, the number of hops is reduced by more than 50% for reasonable network sizes. If symmetrical routing is applied, the significantly larger number of fingers will also lead to about 40% shorter lookup paths, compared with using the finger structure proposed in S-Chord. Using freebie fingers, the corresponding 99^{th}percentile is around $1/2 \log_2 N$, and hence, it is even smaller than Chord's mean lookup path length.

The curves rise more shallowly than $1/2 \log_2 N$. We believe this is caused by the fact that the more nodes participate in the network, the more different fingers are in each nodes finger table [BTG04]. Thus, the routing path length is decreased.

Figure 6.17 shows the mean of the lookup path length in a stable $2^{12} = 4{,}096$ node network for varying ID space sizes. In Chord most lookups can be resolved with less

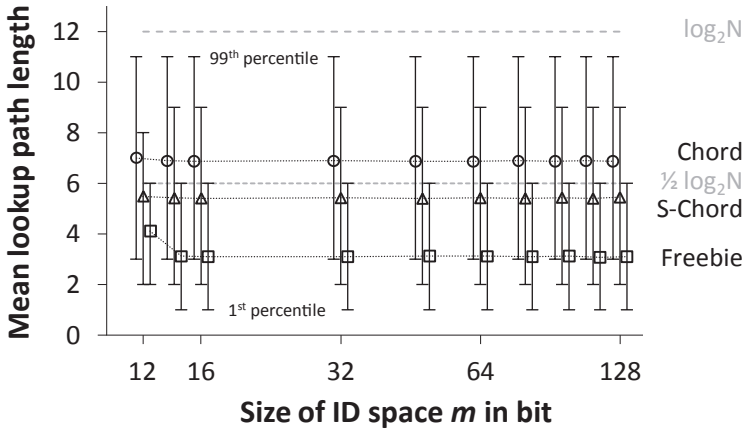


Figure 6.17.: Lookup path length in a stable 2^{12} node network for varying sizes of the ID space

than $\log_2 N = 12$ hops and about $1/2 \log_2 N + 1 = 7$ hops on average (see Section 5.2.1). S-Chord resolves 99% of all lookup with less than 9 hops and the average path length is about 5.4 hops. These values are in accord with the results of other researchers. Using our freebie finger approach, the 99$^{\text{th}}$percentile is 6 hops, and the mean is circa 4.1 hops. The figure also shows that, although the maximal path length is longer, the average path length is slightly shorter in densely populated networks than in fully populated networks. Further evaluations of freebie fingers (regnifs) are available in [RS05]. In addition to our evaluations, the authors show that the system benefits from a large variety of fingers, provided that the stabilization period t_{stab} is much smaller than the mean online time $\text{E}[T_{\text{on}}]$ of the peers. Furthermore, they state that indiscriminately raising the number of successors may result in a decrease of the lookup rate due to oversampling phenomena. An interesting observation is that the longer the lookup path, the higher the probability to incur in several timeouts and, eventually, the higher the probability for the search to fail. Thus, in scenarios with high failure rates, long lookup paths are likely to fail. As a result, the mean path length for successful lookups is reduced deceptively.

Conclusion Summarizing, the Freebie Finger concept adds useful entries to the routing tables of the peers, without additional communication overhead. Thereby, being able to choose from a larger set of routing entries results in a significantly shorter mean path length. Moreover, the symmetrical arrangement of fingers and freebie fingers enables symmetrical routing, thus further reducing the mean path length. Table 6.3 compares the mean number of routing entries and the mean lookup path length for Chord, S-Chord and our Freebie Finger approach.

Reducing the average hop count similarly reduces the required traffic for lookups. It also results in shorter average query times for two reasons. First, in the ideal case, where all finger entries are up-to-date and no absent nodes disturb the lookup path, the lookup time is only reduced by the transmission time the additional hops would have required. Second and more important, the probability of running into timeouts due to wrong finger entries pointing to absent nodes is reduced with every avoided hop. As timer values are

	Chord	S-Chord	Freebie Fingers
Mean number of fingers ($F + F'$)	$\sim \log_2 N$	$\sim \log_2 N$	$\sim 2 \log_2 N$
Mean lookup path length	$\sim 0.55 \log_2 N$	$\sim 0.44 \log_2 N$	$\sim 0.33 \log_2 N$

Table 6.3.: Comparison of Chord, S-Chord and our Freebie Finger solution

set to notably higher values than the average transmission time, the average lookup time will be reduced clearly if fewer timeouts occur. Another advantage of holding more fingers is a higher flexibility in choosing the next hop [GGG+03]. Especially under high churn rates, i.e., nodes joining and leaving the network frequently and therefore fingers pointing to absent nodes with high probability, having more alternatives may be very valuable for performing successful lookups

6.2.4. Fuzzy-based Route Selection (FRS)

In this section, we introduce the idea of a new routing algorithm that uses fuzzy logic to determine the next hop. In the Chord protocol, the next hop is only determined by the distance in the ID space. PNS additionally considers the TT delay before deciding the next hop. In contrast to that, we try to find an optimal next hop considering additional influencing variables, like availability and reliability. For example, as mentioned earlier, the probability of the node being still online is a function of its current session duration. Thus, in order to avoid timeouts, we should prefer nodes that have long online times. Another optimization parameter could be short physical paths. Using fuzzy operations, we compute the *suitability* $S(f_i)$ for all neighbors, and select the neighbor with the maximum suitability as the next hop.

Basic principles of fuzzy logic In the following we give a short introduction to fuzzy logic. Detailed information may be found in related work like [Zim96, NW96, DP80]. In classical set theory an item is either part of an interval or not, whereas in fuzzy logic membership functions usually map items to *fuzzy sets* consisting of multiple objects (see Figure 6.18). Instead of defining a sharp membership interval (e.g., latency is *small (S)* if it is in [0..50] ms), membership functions allow for vague classifications (e.g., a latency of 120 ms is 80% *medium (M)* and *20% large (L)*). Thereby, membership function may have various shapes. Non-numeric *linguistic variables* are often used to facilitate the expression of rules and facts, e.g., short (S) refers to an *almost specific* value of 30 ms, medium is a value *roughly in* [60..120] ms, and large is *around* 140 ms.
Fuzzy rules can easily be defined by using linguistic variables. Rules consists of an antecedent (IF part) and a consequent (THEN part), with antecedents including multiple conditions combined by fuzzy operators, such as AND, OR, and NOT.

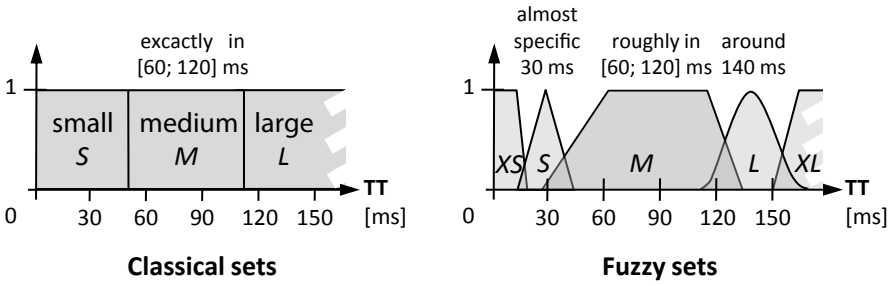


Figure 6.18.: Fuzzy sets allow for fuzzy classifications.

In our P2P scenario, two sample rules could be:

Rule 1:

```
IF latency == short AND online time == long THEN suitability = high
```

Rule 2:

```
IF latency == long OR online time == short THEN suitability = low
```

Using *Zadeh operators*, the AND relationship is an intersection operation, i.e., the minimum of the individual membership functions:

$$\mu_{\mathrm{A}\cap\mathrm{B}} = min(\mu_{\mathrm{A}}; \mu_{\mathrm{B}});$$

Similarly, the OR relationship is the union operation, i.e., the maximum of the individual membership functions:

$$\mu_{\mathrm{A}\cup\mathrm{B}} = max(\mu_{\mathrm{A}}; \mu_{\mathrm{B}});$$

The complement of a fuzzy set A is defined as the negation of the specified membership function. This operation in fuzzy set theory is the equivalent of the NOT operation in Boolean algebra:

$$\mu_{\overline{\mathrm{A}}} = 1 - \mu_{\mathrm{A}};$$

Figure 6.19 illustrates the inter-workings of a fuzzy-based system. Based on various measured input values the output of the fuzzy logic is computed in five main steps:

① **Fuzzification**: Transform the exact input value to its fuzzy membership values.

② **Aggregation**: Combine the antecedents (IF parts) of the rules using fuzzy operators.

③ **Implication**: Compute the fuzzy outputs of the rules. Usually, the minimum operator is applied, i.e., the fuzzy object *high* in the output fuzzy set *suitability* is cropped in height of the probability calculated in the aggregation.

④ **Accumulation**: If multiple fuzzy rules relate to the same output variable, the implication results will have to be combined using the maximum operator.

⑤ **Defuzzification**: Determine an exact output value (*crisp value*) from the output membership function. Most often, the x-value of the center of gravity of the output membership function is used as output value.

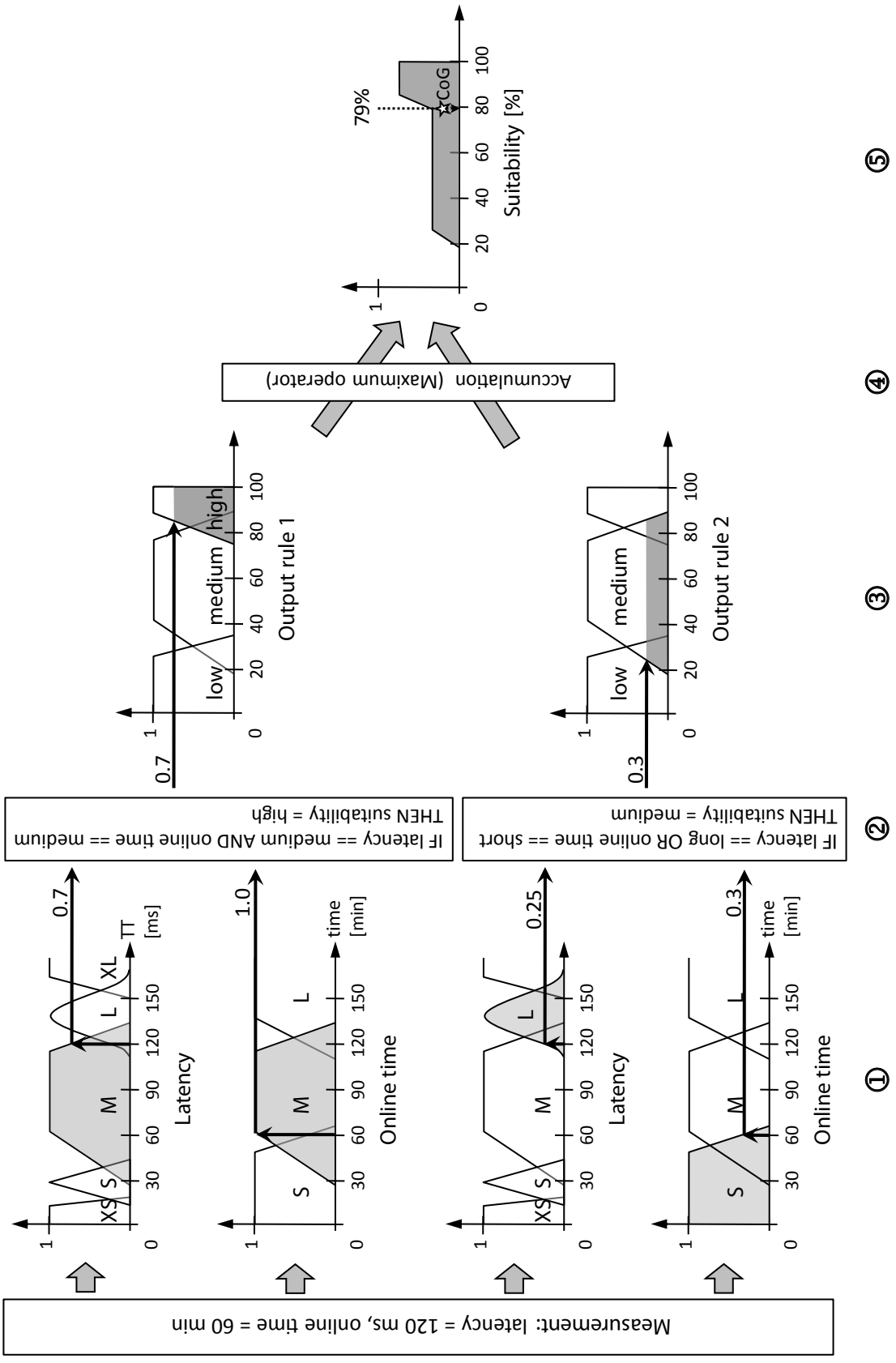


Figure 6.19.: ① Fuzzification, ② Aggregation, ③ Implication, ④ Accumulation, ⑤ Defuzzification

Applying fuzzy logic to DHT routing First, we determine the subset of fingers f_i that are closer to the destination in the ID space. Then, we calculate the suitability $S(f_i)$ for all nodes in the subset. Thereby, a high suitability indicates that the corresponding finger is a good choice for the next hop. Finally, the message is forwarded to the node with the maximum suitability. If the protocol uses parallel routes, the message will be forwarded to the β nodes with the highest suitability. Moreover, in the case of a routing failure, an alternative route, bypassing the failed node, can easily be determined by using another node with high suitability.
We worked out several criteria that might be useful for estimating optimal paths and classified these influencing variables in two categories:

Proximity

- **Distance to the destination:** Nodes that make much progress in the ID space are favorable for achieving short path lengths.
- **Remaining hops:** The estimated number of remaining hops (hops(f_i)) should be low.
- **RTT:** Close nodes in terms of latency should be preferred.

Reliability and availability

- **Online time:** A long online time means a high probability that the node is still available.
- **TLS:** If a packet from the neighbor was just received, the neighbor would be more likely to be available than if the stabilization message from it was overdue.
- **Node stress:** Nodes might signal a high load to their neighbors asking them to forward fewer packets to them.
- **Packet loss rate:** If a high packet loss was measured for that node, it should not be used for routing.

The designer of a specific P2P application should carefully select the influencing variables it considers most important for its application. Then, fuzzy sets and rules like the examples we presented can be easily defined. The main drawback of fuzzy logic is that is does not scale to complex problems. However, in this case only a relatively simple decision based on a few influencing parameters must be made. Moreover, only some of the established rules depend on the specific destination, whereas many rules deal with the general properties of a finger f_i. Thus, these rules can be calculated in advance, and intermediate results can be used for multiple routing decisions. As a result, the computation complexity of fuzzy-based routing is further reduced.

	Chord [SMK+01a]	**CAN** [RFH+01]	**Kademlia** [MM02]	**Pastry, Tapestry** [RD01] [ZHS+04]	**SkipGraph, SkipNet** [AS03] [HJS+03]
Improved stabilization	✓	★	★	★	★
Security considerations	✓	★	★	★	★
Advanced Chord	✓	×	×	×	×
Token Stabilization	✓	×	×	×	✓
Iterative/recursive routing	✓	✓	✓	✓	✓
PNS	[DKK+01]	★	✓	★	[HJS+03]
PRS	✓	★	✓	✓	✓
LNS	[ZY06]	✓	★	✓	✓
Parallel routing	[LLD04]	★	★	✓	✓
Symmetrical routing	[MCVR03]	★	★	★	✓
Chord#	[SSR05, SSR08]	×	×	×	✓
Hybrid routing	✓	✓	✓	✓	✓
Freebie Finger	✓	×	★	★	×
Fuzzy Route Selection (FRS)	✓	✓	✓	✓	✓

(✓: May be applied. ★: Introduced in original paper.
[...]: Introduced in cited paper. ×: Not applicable.)

Table 6.4.: Applicability of the presented improvements in various structured overlay networks

Conclusion In this section we introduced Fuzzy-based Route Selection (FRS). It is a promising solution for combining various heuristics for selecting optimal neighbors and routes, thus, minimizing lookup-latency in structured overlay systems. We believe considering reliability *and* availability is especially promising in scenarios with high churn rates or where wireless connections are involved. Using fuzzy linguistic variables, it is easy to define new rules, which are easy to understand and modify.
Contrary to existing solutions, FRS allows to base routing decisions on a linear value range where heuristics can be applied to select an optimal lookup path. We showed that the output of the fuzzy logic algorithm can be adjusted to the needs of the deployed system and the underlying network structure. We are aware that fuzzy logic is not suitable for larger problems due to the additional computation power needed. Routing decisions, however, are a simple problem with not too many influencing variables.

6.3. Conclusion

Basic structured P2P protocols already feature a scalable lookup path length and a robust overlay structure for low to moderate churn rates. However, simple modifications are able to significantly improve lookup performance and robustness. In this chapter, we analyzed related work and presented our own solutions at the example of Chord.
Thereby, some improvements were already suggested for other P2P protocols (e.g., PNS and PRS), other improvements are limited to Chord-like circular overlays (e.g., Token Stabilization), and the remainder of improvements can be easily translated to various structured P2P overlays (e.g., Fuzzy-based Route Selection). Table 6.4 gives an overview of the presented algorithms and their applicability. There, our own solutions are highlighted in bold font.
We presented an advanced Chord variant, which includes several modifications to the basic Chord stabilization and finger update. The Token Stabilization further improves the robustness of the overlay. We exploit Chord's circular structure and send token-like stabilization messages in both directions. Despite its lower bandwidth requirement, it outperforms both basic and advanced stabilization.
Moreover, we discussed disruptions of the overlay structure. By applying improved stabilization algorithms and calling periodic stabilization more frequently, the probability of a broken ring topology can be reduced. Yet, the probability will never be negligible, thus, one should design for it. Also, the overlay may split in multiple partitions due to, for example, a whole organization being disconnected from the Internet. We discussed several recovery mechanisms and gave design choices that further reduce the probability of disruptions. Also, we introduce a scalable algorithm for detecting other partitions and presented related work on efficient merging of partitions.
Furthermore, we introduced a combination of recursive and iterative routing, which features the advantages from both variants. We also presented Freebie Fingers; additional finger entries that come without additional signaling overhead by exploiting existing finger update traffic. As a result, the mean lookup path length is reduced by approximately 50%. Finally, we discussed neighbor and route selection. Our FRS algorithm uses fuzzy-logic to easily and comprehensively combine various different heuristics and parameters. In particular, considering the availability of nodes, we expect a significant performance gain.

CHAPTER 7

Application of structured P2P for Voice-over-IP

In this chapter we introduce a Voice-over-IP (VoIP) framework, which is based on a global (i.e., physical network boundaries-spanning) P2P overlay network. Therein, the P2P network provides multiple lookup services, like name services, directory services and discovery services.
Most current VoIP solutions (e.g., Session Initiation Protocol (SIP) [RSC+02]) still rely on centralized architectures to provide basic services like establishing calls, or billing and accounting. As each server represents a possible Single Point of Failure (SPoF) and does not scale well, we suggest replacing centralized structures with P2P mechanisms, resulting in *P2P-based VoIP* applications. P2P is highly scalable and does not require a central control. Moreover, it is independent of existing infrastructure, thus supporting ad hoc networks and ephemeral groups. Furthermore, P2P algorithms provide simple discovery and setup.
Skype [Skyb] is a popular example of a proprietary P2P-based VoIP solution. As of October 2008, Skype has been downloaded more than 1 billion times and more than 14 million users are online at the same time [Mer]. The architecture of Skype is based on a hierarchical unstructured P2P overlay network, similar to KaZaA [GDJ06].
From our point of view, *structured* P2P protocols could increase the efficiency of such a system due to their benefits mentioned in this thesis. In this section, we therefore evaluate the applicability of the Chord protocol for signaling purposes in VoIP systems. Establishing a VoIP call can be divided into three steps:

1. Looking up the person in the white pages (directory service) (optional).
2. Finding out the person's current IP address (name service).
3. Calling the person.

The connection itself (step 3) is established directly between the two parties using TCP and/or UDP, whereas step 1 and step 2 require lookups in centralized or decentralized

databases. Thereby, VoIP must compete with established Public Switched Telephone Networks (PSTNs). In order to provide a certain Quality of Service (QoS) these lookups must be answered quickly and reliable. This requirement can easily be met by central instances, but as stated earlier, they require expensive administration and maintenance. In contrast to that, decentralized systems save costs and provide higher resilience, as SPoFs are eliminated. To be able to perform fast lookups in structured P2P networks it is essential that the network structure is stable and as little timeouts as possible occur during queries.
From an operator's point of view, *supplementary services* are an important part of a communication network. Thereby, these services do not solely account for customer satisfaction, but yield a considerable profit, as can be seen by studying for example Skype. There is no basic fee, and Skype-to-Skype calls and instant messaging are free. Yet, Skype reported $143 million in revenue for the third quarter of 2008 [eBa08]. This revenue was mainly earned by additional services, which are subject to a fee, like calling and receiving calls from phones and mobiles, sending SMS messages, or using voicemail. Moreover, (P2P-based) VoIP systems should support existing and additional supplementary services, in order to compete with existing telephone networks.
In the following, we present common supplementary services and features of communication networks. Afterward, we introduce a framework that allows to easily implement new services. Finally, we discuss range, wildcard and complex queries and present a prefix-based query algorithm designed for user lookups.

7.1. Supplementary services and add-ons

Supplementary services in Public Switched Telephone Network (PSTN) In telephony, supplementary services are functionalities of end devices and the telecommunication network. These services simplify the operation of the system and make it easier to use. Moreover, the Intelligent Network (IN), a service-specific network architecture intended both for fixed as well as mobile telecommunication networks, allows operators to provide additional value-added services. Thereby, the network takes over certain functionalities. In the following we give a short overview on common supplementary services in telecommunication networks.

- **Calling Line Identification Presentation/Restriction (CLIP, CLIR)** The caller may define whether his telephone number should be transmitted to the called party or the transmission of the number should be blocked, respectively. In this service, the network must assure that the correct information is transmitted.
- **Completion of Calls to Busy Subscribers/on No Reply (CCBS, CCNR)** With CCBS the caller (A) may activate an automated call-back request at the callee (B)'s switching center in case the callee's line is busy. If B hangs up, his switching center signals to A's switching center, which in turn initiates a new call from A to B. Similarly, a call-back request at B's switching center might be activated if B does not reply to A's call. The next time B completes a call and hangs up the phone, a call-back to A is initiated automatically.

- **Call Waiting (CW)** This service acoustically or optically signals an incoming call despite a busy line. Thus, subscribers are available on the phone during ongoing calls.
- **Call forwarding (CF)** The subscriber is able to forward calls to another device using this service. There are three triggers for CF: The current device is busy, the call is not taken after a period of time, or the subscriber decided to temporarily forward all incoming calls to the other device. The settings for the CF service are stored at the subscriber's switching center.
- **Call Hold (CH)** If an incoming call is signaled during an ongoing call (CW), the subscriber might hold its current call and activate the new call. It may also switch between the two active calls, thereby alternately holding one call.
- **Three Party Service (3PTY)** 3PTY allows three parties to simultaneously talk to each other. This situation might occur if a third party C calls a busy line. The call is signaled to the callee A (CW). Instead of holding its ongoing call with party B (CH), A decides to establish a three party call with B and C. Also, one party in an ongoing call between A and B may invite a third party C to the talk by calling it. Thereby, the calls are mixed locally in the telephone of the conference leader, which establishes separate connections to both parties. Alternatively, operators provide a multiparty conferencing service based on the IN, making a switching center responsible for mixing the calls.
- **Answering machine** An answering machine either is a device that will record incoming calls if the line is busy or the call is not replied. It is installed locally at the subscriber, or it is a value-added service provided by the telephone company. We focus on the second case, where the recorded voice message is stored "in the network". Then, the message might be accessed from different end devices (by calling a certain phone number and authenticating by, e.g., dialing a PIN number).

A possible realization of selected features in a P2P-based VoIP system is introduced in Section 7.2.2.

Features and add-ons for Voice-over-IP (VoIP) systems PSTN mainly relies on switching centers and Intelligent Networks (INs). In contrast to that, VoIP is designed in an end-to-end manner [SRC84], with the end devices taking over much functionality and tasks. Thereby, the core network is more and more reduced to an efficient "bit-pipe". Some services like Call Waiting (CW) may be fully implemented in the end devices. Yet, several services are not feasible without the interaction of the network. Other services can be enhanced if some functionality is offered by the network. For example, storing messages for offline participants is an enrichment for instant messaging. Also, requesting a call-back from another user that currently is not logged in requires an appropriate database in the system.
Moreover, storing information about missed calls and messages in the system enables the usage of multiple end devices that might be contacted with the same identifier. For example, a person A tries to contact a friend B, but the call is not answered. From A's point of view a timely call-back from B will be more likely if the missed call is signaled at all of B's devices. Similarly, B might be interested in receiving instant messages on

the device it currently uses. In this context, we must distinguish between identifiers describing an end device and identifiers describing a person or institution that might be assigned with multiple devices. Usually, someone wants to contact another person and not a specific device owned by that person (except for economic reasons).
Besides providing existing supplementary services, VoIP systems introduce new services and add-ons. In this context, an *add-on* is something added on top of the system that enhances the system. An add-on could be an accessory or feature, whereby a *feature* is an attractive aspect of the system. Common features that are already implemented in existing VoIP solutions include video telephony, instant messaging, collaboration tools, and gaming.
Considering the examples of gaming and collaboration tools, we can observe that these services can either be an add-on to a telephone system (like Unyte Application Sharing [Web] and various online games in Skype), or telephony is an add-on for massive multiplayer online role-playing games (MMORPG, e.g., World of Warcraft [Bli]) and collaboration tools (e.g., Microsoft Office Groove [Mic]). This mixture of different add-ons and services is referred to as *feature interaction*.
In PSTN various supplementary services require interworking, e.g., between the originating and the terminating network to provide Completion of Calls to Busy Subscribers (CCBS). The interaction of two services is defined by a protocol that specifies the required communication. The interactions get more complicated with Internet telephony [LS00]. First, users must be able to trust each other. Setting up a web of trust is much more difficult than compared with the PSTN. Second, the distributed nature leads to various implementations and systems that are controlled by entirely separate organizations. Feature interaction would require these organizations to co-operate. Third, many "Web 2.0" systems and platforms encourage their participants to add value to the application as they use it, e.g., by implementing new services. Tim O'Reilly calls it an 'architecture of participation' [Mus06, O'R05]. He states that Web 2.0 software 'gets better the more people use it', and he even claims that 'user contributions are the key to market dominance'. Thus, features are created by amateur developers that might not be considering interactions at all. Fourth, VoIP is an end-to-end service with packets traveling from user to user. Thus, intermediate servers can not intercept the connection and take appropriate actions. More detailed information on feature interworking and its challenges can be found in corresponding literature like [LS00].
The following list describes common add-ons for Internet telephone systems. A more extensive list of features can be found in [Whi].

- **Online "white pages" (Searching subscribers)** White pages is a common name for a telephone directory. In a communication service the directory lists all participants of the service. It can be used to find the contact information of other users. Due to privacy restrictions, such a directory usually lists only contact information necessary for the corresponding service, e.g., the telephone number in a telephone directory. Additional information like the user's address might be given to distinguish users with the same name. In contrast to printed white pages, their online counterparts offer higher flexibility for searching users. Searching in printed editions is limited to first selecting the town, second the last name and third (if

given) the first name or address. Digital editions often provide searching for any attributes as well as the use of wildcards and complex queries. In Section 7.3 we present solutions for complex queries in decentralized directories.

- **Buddy-Lists and Presence** A buddy-list is like a private phone directory. The user adds people that he (frequently) communicates with and thus wants to keep track of. Storing people in the buddy-list supersedes the lookup in a public phone directory for those people. Buddy-lists are usually stored at a central instance, such that its content is available at all devices a user logs on to. Additionally, buddy-lists often provide presence information for entries in the list. *Presence*, in the context of communication, refers to technology that allows people to see at a glance the status of availability of other persons. In Skype, a user's status might, e.g., be 'online', 'away', 'not available', 'offline', or 'do not disturb' [Skya].
- **Instant Messaging (IM)** Internet communication services often feature an instant messaging service. This service is functionally adequate to a real time chat and enables live text-based communication between two or more users. Various proprietary and open instant messaging protocols exist for fixed and mobile systems. IM systems often allow users to exchange files, e.g., photos and videos. Also, some systems will temporarily store messages if a user is not available. When the user logs on, missed messages are transmitted to its device.
- **Click-2-Dial** Click-2-dial is a feature that allows initiating a call by clicking on an appropriate sensitive area in another application. Often, the sensitive area shows the telephone number or nickname that will be called if the area is clicked. Yet, it might also be, for example, a logo, a button, or the other party's name. Then, the real telephone number or nickname is transparent to the user. Amongst others, the Click-2-Dial sensitive area might be displayed on a web page, on a web portal, in a document, or in an IPTV broadcast.

In VoIP systems users are often addressed by their nickname or SIP Uniform Resource Identifier (URI), whereas the client is contacted by its IP address. Thus, a user might be registered from various devices. In that case, events like incoming calls or instant messages are signaled at all devices. In Skype event logs and chat protocols are even synchronized between all clients. If a client was offline during an event, the missed event will be made available at that client the next time it is online.

VoIP systems often feature mobility, i.e., a user can log in the system from any device with Internet connectivity. On the one hand, this is a great benefit. Yet, on the other hand, mobility comes along with various critical problems. One sensitive issue are emergency calls, as it is no longer feasible to fast and easily locate the geographical position of the caller. A solution to this problem could be devices that know their geographical position. Then, in case of an emergency, the VoIP application could transmit the current location to the called party. Another problem is the ability to backtrack calls. A worldwide authentication of users and devices is crucial to prevent anonymous calls from advertising organizations or stalkers.

7.2. Realizing supplementary services in P2P-based VoIP

In this section we sketch the idea of a framework for realizing supplementary services in a VoIP system. The framework is based on top of a DHT, thus, all services might be realized in a fully decentralized manner. Thereby, all services are implemented basically by using *store* and *retrieve* operations [SK07]. In this thesis, we concentrate on the signaling that is necessary for the services, whereas the actual communication and related issues like audio/video coding are not addressed. The design of our framework offers an easy way to implement new supplementary services. Moreover, it is purely decentralized, supports existing telephony services, and may be implemented platform independent.
As we demonstrated in the previous section, in VoIP many services are no longer provided solely by the network, but clients themselves help realizing the services according to the end-to-end principle. Consider the service three party conference (3PTY) as an example. It is realized by setting up two calls from the owner of the conference to both communication partners, instead of involving a central exchange. Thereby, the conference's owner acts as the exchange, which mixes both calls [JSC+05].
The remainder of this chapter is organized as follows. After presenting related work, we explain the basic concept and functionality of our framework. In section 7.2.2 we present the implementation details of a distributed user directory, therewith exemplifying the challenges arising with decentralized applications.

7.2.1. Related Work

P2PSIP The conventional SIP architecture sets up a relatively fixed hierarchy of SIP routing proxies and SIP user agents. In order to find out the IP address of the User Agent Client (UAC), where a user can be contacted, a proxy/registrar server is used. The SIP specification distinguishes between the *Address of Record (AoR)* and the *Contact URI*. The AoR indicates a certain user, whereas the Contact URI maps to the UAC where the user can be contacted [RSC+02]. In the context of VoIP, SIP is used for initiating calls. The calls themselves are realized end-to-end by setting up direct UDP (or TCP) connections.
In P2PSIP, the hierarchical SIP architecture is replaced by a structured P2P overlay. Thereby, the P2P network takes over the role of the proxy servers, e.g., the mapping of an AoR into one or more Contact URIs. The details of the decentralized solution are currently worked out in the IETF P2PSIP Working Group [BR]. In the current draft, all basic P2P messages, like a JOIN request, are encoded in SIP messages.
The P2PSIP architecture uses SIP specifications, as SIP is a widely established protocol based on standards. Therefore, existing software components may be re-used and P2PSIP will be compatible with most existing equipment. Also, SIP already supports both Instant Messaging (IM) and VoIP.
The authors of the draft, distinguish between between node operations (i.e., join/leave the overlay and stabilization) and user operations (e.g., SIP registration). The P2P overlay is not limited to the mapping of AoRs to Contact URIs. For example, the overlay may

be used to transport SIP messages between any two nodes in the overlay. Moreover, nodes may also offer services to other peers, like a STUN server [RMMW08] or a voice mail (VoM) service. Thereby, additional information about the service may be stored in the P2P overlay, e.g., which peers offer a certain service or which requirements exist for using a service.

PeerThings Communication Platform The PeerThings project[5] is a real-live industrial application for large-scale structured overlay networks developed by Siemens AG [Wim06, Wim06]. Thereby, many conventional network server responsibilities are pushed over to the client side.
As part of the PeerThings project, Siemens developed a decentralized communication platform supporting voice communication, video communication, instant messaging, etc. It has a closed source and proprietary design. Yet, standard SIP is used for voice and video calls. Thus, it fully supports interworking with SIP and PSTN by using appropriate gateways [Rus06]. Siemens states that it is expandable to new application like audio/video streaming or payable transactions. It could also be integrated with mobile phones [Wim06].
PeerThings is based on the Resource Management Framework (RMF) [SR02, FFRS02] with an underlying Chord-like DHT [Rus06]. Thereby, the RMF provides mechanisms to manage resources. A metadata representation is used for describing resources and each resource is identified by a Unique Identifier (UID). Links between resources are realized with a *linkUID* property, which contains the UID of the linked resource. Using the RMF, users can register, search and retrieve resources from the network. Moreover, users may subscribe for certain resources, i.e., they are informed about changes to these resources [Sou06].

Skype Skype is a popular P2P VoIP client developed by KaZaA in 2003 [BS04]. Skype was acquired by eBay in September 2005 for $2.6 billion. On September 28, 2008 the Skype software was downloaded 1 billion times. Thereby, the speed of downloads was mainly linear in the last two years [Mer]. In its "third quarter 2008 results" eBay reported more than 370 million users registered with Skype and $143 million in revenue for the quarter (representing 46% year-over-year growth) [eBa08]. Around 14 million users are online at the same time. The popularity of Skype can be explained by the fact that Skype is easy to install and use, has better voice quality than MSN and Yahoo IM clients, and works almost seamlessly behind NAT and firewalls [BS04]. On the contrary, the Skype protocol is proprietary and almost everything is obfuscated [BD06]. Thus, it is not possible to distinguish normal behavior from information exfiltration. As a result, companies might risk a security gap if the Skype software is used by their employees.
Several researchers tried to analyze the Skype network and client software [BS04, GDJ06, BD06, Mer].
They found out that Skype applies an hierarchical P2P overlay similar to the FastTrack network [GDJ06]. Ordinary hosts connect to supernodes, which form an unstructured P2P overlay. Yet, due to large cache sizes, each supernode knows almost every other

[5] PeerThings was presented at the CeBIT trade show in 2006.

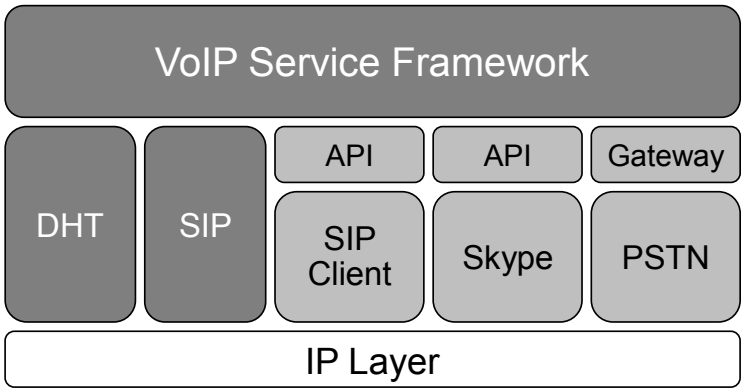


Figure 7.1.: Application stack of our VoIP Service Framework

supernode. In 2006, [BD06] estimated around 20,000 supernodes in the world. Any node with a public IP address having sufficient CPU, memory, and network bandwidth is a candidate to become a supernode.
Skype uses both TCP and UDP for communication. However, no default listening ports are used, but the port number is randomly chosen upon installation and local network conditions. [BD06] shows that Skype can work without UDP, but not without TCP connections. For login, Skype operates a central login server, which appears to be hosted by an Internet Service Provider (ISP) in Denmark [BS04]. The client and the login server have a shared secret: a hash of the password [BD06].
[GDJ06] performed a measurement study of the characteristics of the Skype network. They observe very little churn in the top level network. However, supernodes show diurnal behavior, which is correlated with normal working hours. As a result, median session time of several hours were measured. Session times are heavy tailed and are not exponentially distributed. Interestingly, despite the additional role of supernodes, no excessive bandwidth consumption was observed. 95% of the time, supernodes consume less than 1,000 bps.

7.2.2. VoIP Service Framework

In the following, we explain the functionality of the framework at some sample services and features. It is implemented around a DHT, which simply provides resilient resource discovery. Three types of data resources are used: *contact sheets*, *profiles*, and *info profiles*. Services are implemented by storing and retrieving these resources.
Our framework should provide interaction of various existing VoIP systems. Thus, we use an additional UID instead of existing SIP URIs or VoIP nicknames. The framework is implemented on top of a DHT (see Figure 7.1). In order to integrate an existing VoIP system/client and use its voice channel, it must either provide an API or open protocols like SIP must be used. Moreover, gateways may be used to connect to the PSTN. Thereby, an external DHT and/or SIP client may be used, or the functionality may be directly integrated in the implementation of the framework.
The framework makes a few requirements to the applied DHT. $\langle \text{key}; \text{value} \rangle$-pairs should be replicated in the DHT and should be removed if no update is received for a certain

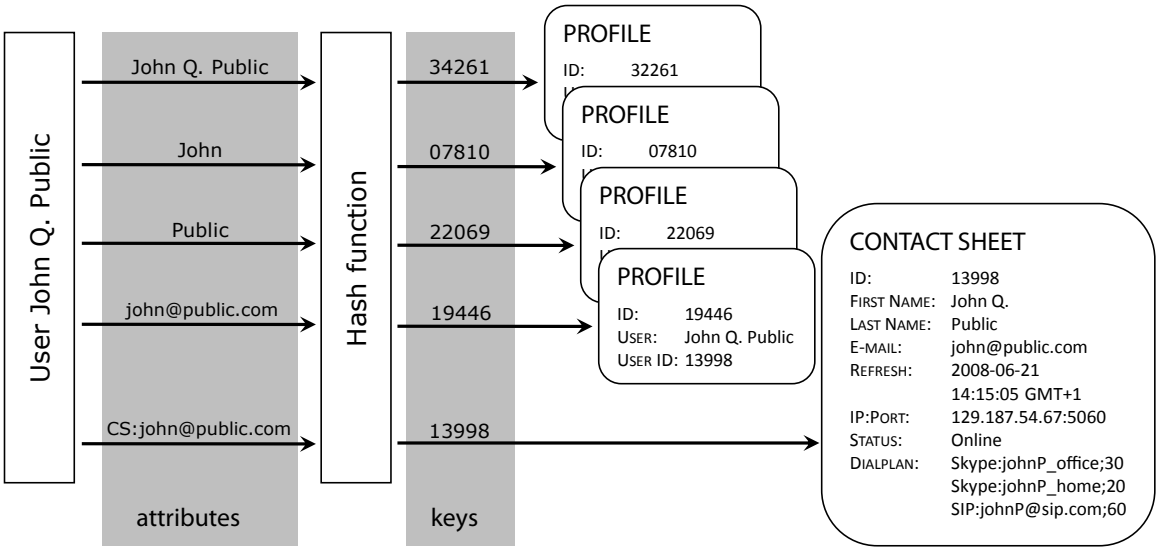


Figure 7.2.: Contact sheet of John Public

time interval. Thereby, contact sheets have long timeout values (in the order of days), whereas other information stored in the DHT may expire after a few minutes. Moreover, the DHT must allow for storing multiple resources with the same ID. If this ID is queried, all resources will be returned.

In our framework, each user publishes a *contact sheet* in the DHT. The contact sheet contains various information about the user, like connection data ((e.g., IP address:port), address, and dialplan (see Figure 7.2). Note that in the course of this thesis we neglect security aspects, like authentication or the visibility and privacy of user data. Contact sheets are stored in the DHT using, for example, the user's email address with the prefix 'CS:' as input for the hash function. For example, the key of John Public's (JP) contact sheet is $\mathcal{H}$(CS:john@public.com) = 13998. This key is also referred to as UIDs. It addresses a certain user, not a specific device.

Profiles are used to provide a directory service, whereas *info profiles* store additional information for users.

In order to establish a communication channel to a user, its UID must be looked up in the DHT to learn how the user wants to be contacted. Then, the application tries to establish a connection to the user with respect to the dialplan.

White pages (Searching subscribers) and Buddy lists Usually, users do not know the ID of the person to be contacted. Therefore, it must be possible to look up the ID of the contact sheet in a phone-book-like user directory. Like in a printed phone book, the user directory should contain sufficient information in order to clearly find a certain person. Thus, information like the user's postal or email address should be stored in the contact sheet. Moreover, it is necessary to store additional *profile sheets*. The profile sheet is a small resource that connects a certain attribute, like the user's last name, to the corresponding contact sheet. In Figures 7.2 and 7.3, a profile sheet with ID $\mathcal{H}(john@public.com) = 19446$ stores the full name of user JP as well as the ID of JP's

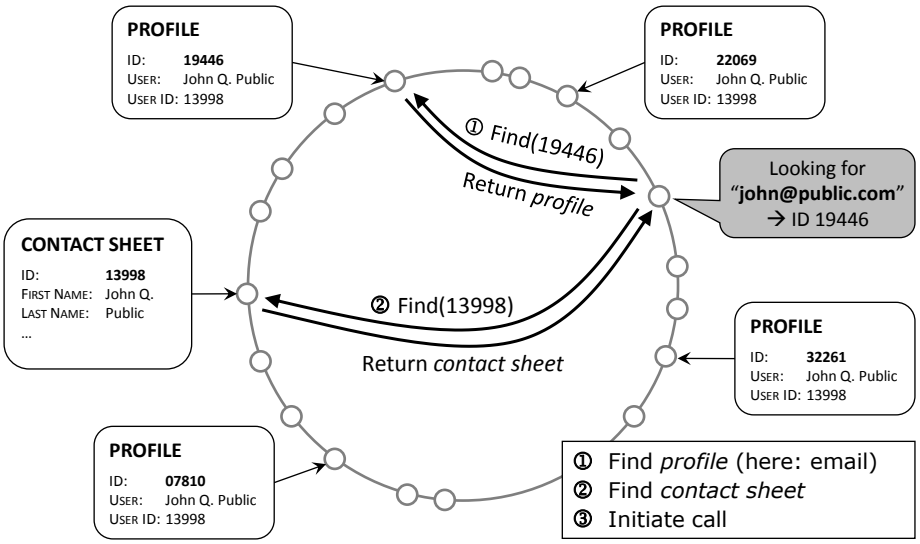


Figure 7.3.: Sample lookup for user John Public

contact sheet. If another user A knows JP's email address, he will be able to calculate the ID by hashing the email address. Then, A searches the DHT for this ID (Figure 7.3 ①). He receives the corresponding info profile and learns about the ID of JP's contact sheet. Finally, it is able to retrieve the contact sheet ② and call JP ③.
Storing additional profile sheets will be necessary to find a user if its ID is not known. For each attribute, one should be able to find in the directory, a separate profile sheet is required (see Figure 7.2). Yet, there are several problems with this approach. First, many users with the same attribute values exist, for example, millions of people are called *John*. As shown in Section 3.3, names follow a Zipf-like distribution. There are a few very common names, and many names that appear very infrequently. Thus, a node, which is responsible for a frequent name would have to store a huge amount of data. Also, many queries for that name must be answered and nodes might not be able to handle this extremely high load. Load balancing techniques (as introduced in Section 3.3) are no feasible solution to this problem, as each name is hashed to one particular ID. Second, due to the applied consistent hashing, a DHT resolves only exact query matches. Mistyped entries, entries containing wildcards, and complex queries are not supported innately by DHTs. In Section 7.3 we evaluate related work on alternative search indexes for DHTs and present PriMA KeyS, a prefix-based multi-attribute keyword search algorithm.
Buddy lists are a common add-on for IP-based telephony. Using the DHT, buddy lists may be stored in the network. Thus, buddy lists for an user are available at all clients, where the user is logged in. That way, changes to the buddy list are also kept synchronized on all clients.

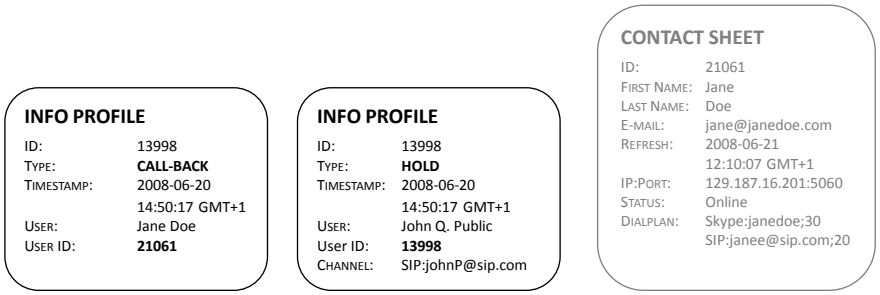


Figure 7.4.: Info profiles CALL-BACK (left) and HOLD (center) and contact sheet of Jane Doe (right)

Dialplan and call forwarding The *dialplan* of a user B contains rules for establishing a connection to B. It defines the kind and order of services that might be used to contact the user. In our example, John Public can be contacted using Skype or SIP. The dialplan is a linked list with a variable number of entries and it is processed in the given order of the list. If the call is not answered within the specified timeout value, the next entry will be used to establish a connection.
User B may alter its dialplan at any time by updating the replicas stored in the DHT. Prior to calling B, a user A retrieves B's contact sheet. Thus, it is guaranteed that the latest dialplan is used and that changes in the dialplan immediately take effect. *Call forwarding* (CF) can be enabled by removing an entry or setting its timeout value to zero, thus, skipping this rule.

Completion of calls on no reply (CCNR), missed calls This supplementary service is realized by using *info profiles* of type CALL-BACK. If no rule in the dialplan was successful, a call-back request would be generated and stored in the DHT. As the callee must be able to find appropriate info profiles, the info profile is stored with the same ID as the ID of the callee's contact sheet. Thus, profiles are stored on the same peer as the corresponding user's contact sheet. The profile contains a timestamp, the name of the caller A, and his UID (see Figure 7.4 (left)).
Imagine user A tried to contact B, but the call is not answered, thus A generates a call-back request. Each client will periodically query the DHT for the UID of the registered user, in order to find out if appropriate info profiles exist. If a new info profile of type call-back exists, user B might decide whether he wants to ignore the request or if he wants to establish a connection. In the second case, B's client queries the network to find out A's current dialplan and tries to set up a connection with A. After successfully calling A, the info profile is either marked done or is removed from the network.
There is a main difference of our realization compared with the PSTN. The call-back is not initiated from a telephone switch, but from the end device. Moreover, in our solution the call-back is not limited to the called device, as the call-back is signaled at all devices of user B.

Putting calls on hold *Info profiles* are also used to put calls on hold and resume the call from another location or device. A sample profile of type HOLD is shown in Figure 7.4 (center). If one party A decides to use this service, it will generate an info profile that contains a timestamp, the name and UID of the other party B, and the currently used voice channel. Again, the profile is stored with the UID of A's contact sheet. Then, the call is disconnected.
Similar to CCNR, all devices of user A periodically query the DHT for info profiles of A. Thus, the information about the call on hold will be available at all devices. User A then may try to re-establish the call using the former voice channel. In case of no reply, it might also lookup B's latest dialplan and try another communication channel.

Answering machine, voice mail (VoM) An answering machine may either be installed locally on an end device or as a service provided by the network. The first case is trivial and not considered in this work. A voice mail (VoM) is an example for a service that may be provided by a third party. Thereby, two aspects are interesting to us.
First, a discovery service may be queried for an appropriate service provider. The discovery service allows services to register themselves in spontaneous networks for later lookup. [SSR08] lists some typical examples that provide standard interfaces and schemas for registering and finding services: Jini [Jin], Globus MDS-2 [Glo], and the UDDI web service [UDD].
In our realization, the service discovery and our framework could use the same DHT overlay. Users search the DHT for a voice mail service. After registering with the service, its provider returns relevant data for setting up the service. For example, users receive an additional VoM UID from the provider. The service is activated by integrating the service along with the UID in user B's dialplan. If user A tries to call B and the VoM entry in the dialplan is executed, A will search the DHT for B's VoM UID. As a result, user A receives the connection details necessary for contacting the VoM service. After A established the connection and the voice mail is recorded, the VoM service uploads an info profile to the DHT using B's UID. This profile contains connection details about the host that recorded the voice message. Thus, after retrieving the info profile, B might connect to this host and listen to the recorded message. The recorded voice mail might also be directly stored in the DHT along with the info profile. Then, the P2P overlay is responsible for maintaining a copy of the message.
In this context, a service provider might be any node participating in the system. Thereby, incentives, likes virtual money, might motivate users to provide processing capacity and storage required for setting up services [UB03].

7.3. Realizing range, wildcard, and complex queries

DHTs establish a structured P2P overlay network by applying proactive routing algorithms. Due to their well defined structure, this class of P2P protocols is able to locate any content in the system within a limited number of hops. However, basic DHT algorithms are limited to queries for content that exactly matches the search term, as they

provide only hash table functionalities. In contrast to unstructured P2P overlays, advanced queries, like range, wildcard and complex queries are not inherently supported by structured P2P protocols.
Wildcard characters are used to substitute for any other character(s) in a term. Some common wildcards that match zero or more characters are the asterisk sign (*) and the percent sign (%). The question mark (?), the period (.), or an underscore (_) usually substitute as a wildcard character for a single character [MRS08].
Prefix-based queries (or trailing wildcard queries) are a special case of wildcard queries, which search for all entries that share a common prefix, i.e., the * query symbol occurs only once at the end of the search string. Similarly, using a *range query* all entries of a database with a value between an upper and lower boundary may be retrieved. The difficulty with these queries is that it is not generally known in advance how many entries a query will return. *Wildcard queries* will be used, for example, if the user is aware of multiple variants of spelling a term or if he is unsure about the correct spelling of the term (e.g., Sydney vs. Sidney, color vs. colour) [MRS08].
A *complex query* is a query that includes more than one type of operator. It may be created by using the UNION operator, which takes the union of all rows returned by several queries. Also subqueries (or nested queries) and special predicates (e.g., ANY, EXISTS, or IN) may be used to create a complex query.
We present a Prefix-based Multi-Attribute Keyword Search (*PriMA KeyS*) that is specially designed for DHT-based community services, like searching persons in a distributed phone book. Our architecture is fully distributed and pays special attention to a balanced storage load distribution as well as low network traffic. Hierarchical identifiers generated from multiple keywords help to reduce the load on nodes that host common keywords. Additionally, a locality preserving hash function enables prefix-based queries. An extensive linguistic analysis of search keywords is carried out to select optimum design parameters.
We discuss related work in Section 7.3.1 and define our *Prefix-based Multi-Attribute Keyword Search (PriMA KeyS)* algorithm in Section 7.3.2 Using sample queries, we present experimental performance results in Section 7.3.3. We show that our system can efficiently handle both detailed and unspecific queries.

7.3.1. Related Work

Keys and Zipf distribution In database terminology an *entity* is an existing *object*. Information about this object is stored in the *database*. An object can, for example, be a thing, transaction, event, or person. A collection of single entities that share similar or comparable properties is called an *entity-set*. Typical properties of an entity are called *attributes*. Entities are characterized by their attributes. A *key* is the minimum combination of attributes that identify an entity out of an entity-set. In many cases not all available attributes are necessary to identify an entity. Objects that have a unique serial number can be identified by this number. Persons can, e.g., be identified by their last and first name, and date of birth. If there are several objects that share the same key, the database will return all matching entities.

Most current DHT protocols generate an the key of an entity by hashing one or more attributes. As a result, contacts will only be found if the querying person knows all attribute values that are part of the key. As certain attribute values like George, Smith, or Fairview are very common (see Section 3.3), it is not feasible to map each of these keywords to a single ID. To avoid common keywords in DHTs, an ID is generated using several attributes as input parameters. The basic idea is that each resource (except for clones), like a person in our example, will be distinguishable from other resources if enough attributes are considered. This raises two questions: What attributes should be considered? How should the key be generated from the attributes of the ressource? The answer to the first question mainly depends on the considered application. In our telephone book, we identify users by four attributes: first and last name, street, and town or zip code. We assume that the zip code can be translated into the corresponding town, for example, by using a distributed hash table and vice versa. Additional attributes store things like the user's phone numbers and email addresses. In section 7.3.3 we show how to determine how many attributes should be used for generating keys with the algorithm we propose.

Keyword Search Most search algorithms in DHTs are based on an *inverted distributed hash table* explained in detail in [RV03]. Information about available resources is made locatable by storing ⟨identifier; list of locations⟩-pairs in the P2P system. If multiple keywords are stated in a query each keyword might be searched separately. Results from different nodes are then merged at the initiator of the query. A more efficient solution is to include all keywords in each query. Then, nodes hosting one of the keywords should return only resources that match all keywords. Thereby, the number of responses will be reduced, especially, if common keywords are used in the query. Another efficient approach is based on *chained processing* [LSS02]. At first, the keywords stated in the query are put in ascending order according to the size of their index. Then, the query is routed to the node that hosts the first keyword. This node writes matching content to the list of results and forwards it to the node hosting the next keyword in the chain. Each subsequent node successively intersects its index with the previous results. Eventually, the last node in the chain constructs the final answer and returns it to the initiator. Throughout the process the size of the list of results can be no larger than the index at the first node in the chain. By sorting the nodes in advance, it is ensured that the amount of transferred data is minimized.
MAAN [CFCS03] provides range queries by extending Chord with locality-preserving hashing for numerical attributes. Multi-attribute queries are performed either by searching for all queried attributes separately and intersecting the results at the query originator, or by piggy-backing all sub-queries in each search request. Then, nodes can locally select resources that are valid for all query parameters.
The *Squid* approach [SP04] uses all available attributes to span a multi-dimensional attribute space. Provided there are enough different attributes each resource corresponds to a unique point in the space. Then, a space filling curve (SFC) (e.g., a Hilbert or z-order curve) is applied to the attribute space. SFCs are continuous curves that pass through every point of the unit-space. Most SFCs possess a good locality-preserving property as

they are constructed iteratively. This property ensures that similar resources are mapped to keys that are close to each other, thereby enabling range queries. Finally, the SFC is mapped to the DHT's ID space. Related approaches are *SCRAP* [GYGM04] and *ZNet* [SOTZ05]. Users specify certain attributes when they search for specific resources, thus selecting a subspace of all available attributes. The more attributes are specified, the smaller is the resulting subspace. Wildcards and range queries are realized in a similar way. Next, all reaches of the curve that are situated within this subspace are determined. Finally, the DHT is queried for all nodes that are responsible for these reaches. In our telephone book scenario we face several drawbacks of the Squid approach. Users looking up data tend to specify only a small amount of attributes. Then, the resulting subspace is very large and a lot of nodes must be contacted in order to resolve the query. Also, all attributes that span the attribute space must be known when inserting a resource. Due to protection of their privacy, quite a few users prefer to publish only their name and phone number without specifying their address. Additionally, IDs are usually limited to 128 or 256 bits. Mapping a huge attribute space linearly to the ID space, many adjacent points in the attribute space are mapped to the same ID. Densely populated areas in the attribute space may then result in only a few different IDs. Consequently, load balancing the system is not feasible any more.

Mercury [BAS04] builds a separate attribute hub for each attribute. Within each hub the resources are ordered by their attribute values. In the first step an intelligent query algorithm selects the hub that leads to success with highest probability. Histograms that show the density distribution in all hubs support the selection. In the second step all resources that possess the queried attribute value are returned. The main disadvantage in our kind of application is that very frequent attribute values like Smith are still mapped to a single node in the hub. Also, combining different attributes to reduce the query range is not possible.

Other approaches try to arrange all resources in a hierarchical scheme. In our scenario the topmost level could be the country followed by state, town, street, and finally last name. [GEFB+04], for example, creates a hierarchical indexing scheme. Content is identified by XML descriptors and can by queried using XPath expressions. Indexes contain query-to-query mappings, i.e., for a given query q a list of more specific queries, which are covered by q, are returned. Matching queries are processed recursively until the desired content is found.

Keyword fusion [LL04] tries to reduce storage and network consumption by exploiting the fact that many keyword sets follow a Zipf-like distribution of the keyword popularity. The algorithm is based on an inverted distributed hash table. Common keywords pose a problem for DHTs. For example, the most common first name in our white pages dataset is *Peter* with about 150,000 entries. Storing all resources and answering all queries for this keyword would be an obvious burden for a single peer. Therefore, common keywords and their corresponding lists of values are removed from the table and the keywords (without their values) are stored in a distributed *Fusion Dictionary* instead. These keywords can no longer be queried separately, but must be combined with additional keywords. Thereby, queries for frequent keywords that would return thousands of results are prohibited. Also, nodes that formerly were responsible for hosting these keywords are unburdened. For example, combining first and last names results in 5,300 *Mueller, Peter*,

which is the most common combination of first and last names in our dataset. A more even load balance can be achieved by adding more attributes, for example, 485 *Mueller, Peter* live in a city with the initial letter *A*. As expected, the number of matching entries is significantly reduced when considering more attributes.

If additional keywords are specified, which are not part of the Fusion Dictionary, the chained query process will start at one of these keywords. Keywords that are contained in the dictionary are added to the query in a so-called *partial keyword list*. In order to be able to publish and query resources that are only characterized by common keywords, *Keyword Fusion* is applied. New synthetic keywords are generated by concatenating two or more common keywords in alphabetical order. Thereby, the frequency of the synthetic keywords is reduced compared with the individual frequency of the original keywords. The Fusion Dictionary is cached at all peers in order to further decrease network traffic. Using this approach, storage consumption of the top 5% most loaded nodes can be reduced by 50% and the overall search traffic can be decreased by up to 80%. The main problem with this approach is that synthetic keywords which result from concatenating common keywords will only be retrievable if both keywords are exactly known. In our algorithm, we take up the idea of a distributed dictionary and extend it in such a way that range queries, as well as wildcard searches, become possible.

[WMB99, MRS08] split each keyword to be indexed into *n-grams*, i.e., distinct n-length substrings. For example, the keyword `Peer-to-Peer` could be split into 10 trigrams: `Pee`, `eer`, `er-`, `r-t`, `-to`, `to-`, `o-P`, `-Pe`, `Pee`, `eer`. Instead of inserting the keyword into the DHT, for each n-gram g_i the pair $\langle \mathcal{H}(g_i); \text{value} \rangle$ is indexed in the DHT. Thereby, an index over n-grams for various values of n (typically $n = 2; 3$) can be built. Similarly, lookups for keywords are also split into n-grams. For example, the queried string `Peer` is also split into multiple n-grams g_i (e.g., the trigrams `Pee` and `eer`), and a separate lookup is done for their hash values $\mathcal{H}(g_i)$. The results are grouped by resource and sorted by the number of occurrences, with resources matching in many n-grams being displayed at the top of the list.

Extended Prefix Hash Trees [SKM07] are an indexing infrastructure that supports range queries on top of DHTs. Multiple keywords are concatenated to a unique ID according to a given priority. This ID is subsequently arranged in an n-ary prefix tree (trie). Thus, all nodes can be accessed efficiently. Range query functionality is provided by connecting all non-empty leaf nodes through a linked list. Nodes in the trie are stored in a DHT in order to achieve load balance among all participating peers. Caching frequently accessed prefixes reduces network traffic and stress on the responsible nodes. Due to the proposed construction of IDs, a wildcard in an attribute will supersede all given attributes with lower priority. Sometimes in a telephone book application, the usage of wildcards will be helpful if the correct spelling of an attribute value with high priority is not known exactly. For example, the German last names Maier, Mayer, and Meier are pronounced the same way. In our approach, further attributes reduce the number of results even if `M??er` is queried and last names have top priority. Using the same query with EPHT leads to countless results as the ID would diminish to `M*`.

Further information about search methods in recent P2P networks can be found in [RM06].

7.3.2. Prefix-based Multi-Attribute Keyword Search (PriMA KeyS)

In this section, we describe the design of our query algorithm. Our approach is based on some of the ideas summarized in Section 7.3.1 but mainly differs in the way we construct the IDs used in the overlay [SKM07].

Composition of keys The overlay IDs in our approach consist of several sections that contain different attributes. Figure 7.5 shows an ID that is constructed using up to three attributes. Each section contains the locality aware hash of the corresponding attribute value. A key could then be composed of the first five characters of a person's last name, three characters of its first name, and the first letter of the town name (5L 3F 1T). We call IDs consisting of characters *names*. The more attributes are considered, the more unique the resulting name is. However, as we assume a constant ID size, the more attributes we consider the fewer characters can be used per attribute and the more entries result in the same name. Additionally, if there are many entities that cannot be distinguished by their attributes, the l least significant bits of an ID might be set to a random value. This results in up to 2^l different IDs which will be suitable to be hosted by different nodes if load balancing algorithms are applied.

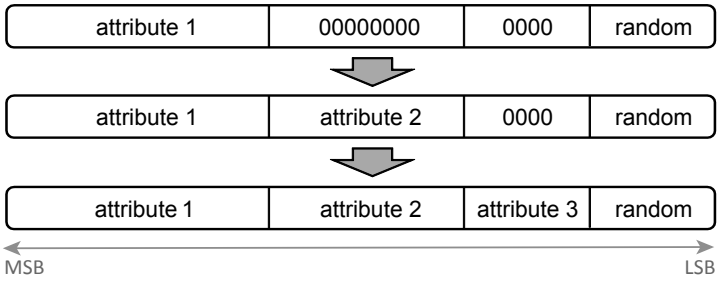


Figure 7.5.: Keys consist of multiple sections.

Names can be transformed to binary IDs (and vice versa) by applying a simple basis transformation: Any d-digit word from an alphabet with b different characters (base b) can be converted to a decimal number by multiplying the index of each character in the alphabet with $b^{(d-i-1)}$ and building the sum of all resulting numbers:

$$\sum_{i=1}^{n} j \cdot b^{(n-i-1)} \tag{7.1}$$

where i is the index of the corresponding character in the d-digit word and j is the index of the corresponding character in the alphabet starting with $j = 0$.

Using 26 characters ($b = 26$) from A to Z the keyword `GERALD` relates to the following decimal value:

$$\begin{array}{lrcl} \texttt{G} & 6 \cdot 26^5 & = & 71288256 \\ \texttt{E} & 4 \cdot 26^4 & = & 1827904 \\ \texttt{R} & 17 \cdot 26^3 & = & 298792 \\ \texttt{A} & 0 \cdot 26^2 & = & 0 \\ \texttt{L} & 11 \cdot 26^1 & = & 286 \\ \texttt{D} & 3 \cdot 26^0 & = & 3 \\ \hline & \sum & = & 73415241 \end{array}$$

In an analogous manner names can be converted to a binary ID. Additional coding of the keywords can further improve our system. By replacing common character strings in the keywords with other symbols, much like in a Huffman Code, keywords can be distinguished with fewer characters. Many German last names, for example, share the prefix `SCH`. If the total number of characters assigned to this attribute is four, only one additional character will be available to differentiate between these names, whereas replacing `SCH` by a single symbol leaves three characters available to distinguish the last names. We also recommend shifting all letters to upper case as it is not necessary to distinguish between lower and upper case. The size of the applied alphabet will also be kept small if letters with special characters like umlauts or accents are dropped or translated to the corresponding letter without umlaut or accent.

Fusion Dictionary In general, users query phone book entries by stating different attributes. Printed editions necessitate selecting the town in the first step. Then, the person will be searched by its last name and finally, if several entries exist, by its first name. Sometimes entries can also be distinguished by additional information about the person's address. Digital editions allow for further search capabilities, for example, finding a person by its phone number. Online communication services like Skype even offer their users more flexible queries for other users in the system.
Even if our IDs were built with a fixed structure like in printed phone books, the resulting system would not be limited to finding only persons in a known town. Using a wildcard for the town attribute, it would still be possible to find a person solely by the person's name. However, this would require immense query costs, because processing the query would be similar to browsing all printed phone books. We therefore suggest generating several IDs for each resource according to the most common query patterns. However, storing multiple IDs per entry manifolds storage consumption as well as network traffic caused by republish and replication events.
Exploiting the fact that phone book attributes follow a Zipf-like distribution we suggest applying a *Fusion Dictionary*-like approach. The underlying DHT can efficiently look up values of rare occurrence. There is no real benefit in generating IDs built from several attributes for these values. However, for the small set of frequently occurring values the number of matching resources will be significantly reduced if multiple attributes are considered. Nodes that are responsible for hosting a common keyword add this value to

the dictionary. They also generate new IDs consisting of two or more attribute values and move the corresponding entries to nodes, which are responsible for the new IDs. Up from this moment, queries for these keywords can only be performed in combination with other attributes.
The system can either globally specify all utilized keyword combinations in advance or store available combinations with the keyword in the dictionary. In the second case, nodes can choose optimal attribute sets at runtime by evaluating the resources they host as well as all corresponding queries they answer. The dictionary itself is stored in the DHT. Before a node searches for a resource, it contacts the dictionary and verifies whether it contains one or more of the keywords. Since the dictionary entries are not expected to change significantly over time, the complete dictionary is cached at all nodes in order to reduce the total number of dictionary queries.

Locality preserving hashing The use of locality preserving hash functions will be advisable if range query functionality is required. Only locality can assure that similar entries are mapped into a small range of the ID space. On the contrary, cryptographic hash functions try to remove locality in order to achieve good load balancing. A perfect cryptographic hash function, which receives two input values that differ in only one bit, returns two IDs that differ in half of their bits. In our evaluation we apply a very simple locality preserving hash function [CFCS03]:

$$\mathcal{H}(v) = (v - v_{\min}) \cdot (2^m - 1)/(v_{\max} - v_{\min}) \tag{7.2}$$

where $v \in [v_{\min}, v_{\max}]$. According to our identifier space $v_{\min}$ and $v_{\max}$ are set to 0 and $2^m - 1$, respectively, resulting in $\mathcal{H}(v) = v$. This function corresponds to trimming keywords to a fixed number of characters and then sorting the resulting IDs in an alphabetic sequence. In order to deal with the load imbalance, which results from locality preserving hashing, load balancing mechanisms have to be applied (see Section 3.3).

Searching for IDs that contain wildcards In the first step, we define the ID range that is to be queried. The start ID ID_{start} is constructed by replacing all digits that contain a wildcard with the first character of the used alphabet. The stop ID ID_{stop} is constructed analogously by replacing the wildcards with the last letter of the alphabet. Then the corresponding binary IDs are calculated and node n_1, which is responsible for ID_{start}, is queried using the DHTs native *findpeer* algorithm. Additionally, the query packet contains the complete list of search keywords. The queried peer will check whether entries in its local database fit the keywords and returns matching content. The initiating node can already display these intermediate results to satisfy the user. If the ID of node n_1 is larger than ID_{stop}, the query is completed. Otherwise, the closest neighbor of the node, n_2, is contacted resulting in further matching content. This procedure is repeated until the node n_i at the end of the query range is reached.
In order to reduce network traffic and overload on the nodes, we introduce a slightly more complex query algorithm (see Algorithm 1). After receiving data from node n_1, the initiating node transforms the node ID of node n_1 to its character representation. This ID is then increased by one while taking fixed and wildcard characters into account.

```
ID_start = Replace( '?', 'a', ID );
ID_stop = Replace( '?', 'z', ID );
while ID_start ≤ ID_stop do
    node = findPeer( ID_start );
    ReceiveQueryResults();
    ID_start = IncID( node.ID );
end
DisplayResults ;
```

Algorithm 1: Pseudocode of the query algorithm.

Figure 7.6 gives an example of our increase function IncID. Different letter case is only shown for demonstrative purpose, but not taken into account for queries.

Search keywords	Last name: Boy
	First name: George
	Street: Broadway
	Town: *
Key (5L 3F 1S 1T)	`BOY??GEOB?`
ID_{start}	`BOYaaGEOBa`
ID_{stop}	`BOYzzGEOBz`
IncID(`BOYaaGEOBa`)	`BOYaaGEOBb`
IncID(`BOYaaGEOBb`)	`BOYaaGEOBc`
IncID(`BOYaaGEOBz`)	`BOYabGEOBa`
IncID(`BOYabGEOBa`)	`BOYabGEOBb`
IncID(`BOYgzGEOBz`)	`BOYhaGEOBa`

Figure 7.6.: Exemplification of the IncID() function

Incremental Results Most current online (communication) services like Google or Skype display search results incrementally. We also use this mechanism in order to reduce unnecessary overload in the system. Users who recognize that their search is too unspecific and produces too many results will cancel their query and add further or more specific search attributes. Unless there are no more additional attributes left, a user is not willing to browse through lots of query results. In addition, searches will appear to be faster, if the first few results are displayed as soon as possible, while the rest of the results is collected in the background. The underlying DHT is able to route to the first node in the query range in about $O(\log_2 n)$ (e.g., Chord [SMK+01a]), $O(\sqrt{n})$ (e.g., CAN [RFH+01]), or $O(1)$ (one-hop routing, e.g., [GLR04]) hops. In our system the query range that is spanned by unspecific queries, is often hosted by several nodes which are queried sequentially. Thus, incremental results can be easily generated by our system.

In order to improve the distribution of the query load to a query range (especially for ID ranges corresponding to frequent keywords) an extended version of our system does not route to the first node in the range but to a random node within it. This can be easily done by initially setting wildcards to random characters instead of the first letter in the alphabet. From the node that is selected by random, the search then proceeds in both directions.

7.3.3. Evaluation

Simulation environment We make some simplifying assumptions in our simulation environment. To capture the performance of the search mechanism itself and to exclude side-effects caused by the maintenance routines of the underlying DHT, we study a stable system without churn. We also assume that efficient load balancing mechanisms are applied and that thus storage load is almost evenly distributed to all participating nodes. In order to keep simulation complexity low, we implement a very simple global load balancing algorithm. In particular, we sort all resources by their IDs. The optimal number of resources num_{res} each node should store, is the number of resources divided by the number of participating nodes. We then assign the first num_{res} resources to the first node. As resources with the same keyword must be stored at the same node, we check whether there are further resources that will have to be assigned to this node. Then, we go on to the next node and repeat the procedure until all resources are allocated.
In a realistic environment, a significant part of all existing users is offline at any given time due to switched off devices, connection problems, discharged batteries or dead spots of mobile users, etc. To capture this effect in our simulations, we assume that only about 4% of all nodes are online and must store the entire database, i.e., each node hosts about 25 resources on average.

Generating IDs Our query system will not be able to distinguish whether the user just typed in the initial letters of the last name or if the last name is actually that short. If the query string is shorter than the number of characters allowed for an attribute the query string will thus automatically be filled up with the wildcard character (`?`) like in the example in Figure 7.6.
The exact size and number of the ID ranges which must be queried mainly depend on the position of the first wildcard symbol in the ID. The further to the left the symbol appears, the more nodes have to be accessed during the lookup. If we use an attribute structure containing 6 characters for town, 4 characters for last name, and 2 characters for first name (6T 4L 2F) and a user searches for town (Munich) and last name (Kunzmann), the resulting ID will be `MUNICHKUNZ??`. The corresponding ID range is very small and likely to be hosted by only a few nodes (Figure 7.7). In contrast to this, if we use the following structure (7T 4L 1F), the ID will be `MUNICH?KUNZ?`. This time the ID range is split into multiple parts which are highly likely hosted by multiple nodes.
In order to determine the optimal number of characters for the main attribute, we perform a linguistic analysis. Figure 7.8 shows the Cumulative Distribution Function (CDF) of the length n of names in the United States (source: [U.S]) on the left y-axis (solid lines). The mean length of male first names, female first names, and last names is 5.72, 6.03,

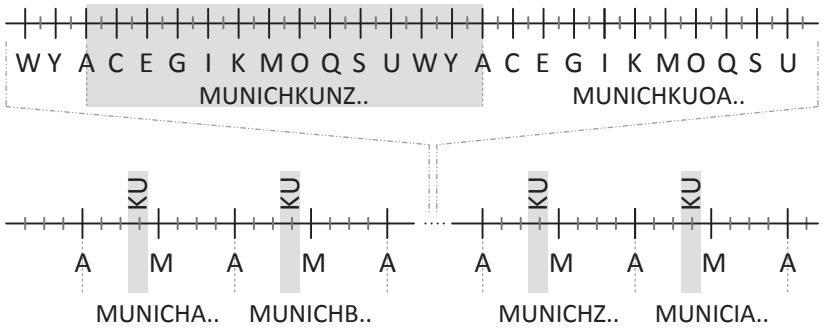


Figure 7.7.: The query `MUNICHKUNZ??` corresponds to a small continuous range (top), whereas a query for `MUNICH?KUNZ?` results in many disjoint ranges (bottom).

and 6.12, respectively. The right y-axis depicts the frequency of the most common string with the last names trimmed to n characters (dotted lines).
On the one hand, the number of resources that match a certain prefix will be large if only a few characters are used in the ID. Figure 7.8 further shows that more than 3 characters should be used in order to reduce the maximum number of matching resources to a reasonable value. On the other hand, if we use many characters for the most significant attribute there will be a lot of queries for shorter values. For example, about 90% of US last names are shorter than 8 characters. When using an attribute with 8 characters for the participant's last name, 90% of all queries would contain at least one wildcard in the first 8 characters, resulting in a large ID range which would have to be queried. In addition, the number of characters that can be stored in a 128 or 256-bit ID is limited. Table 7.1 shows the number of characters that can be stored in an ID depending on the size of the alphabet used.
Our initial simulations showed that 5 characters is a good trade-off for the main attribute when searching for random persons. Further attributes can be stored with fewer characters as the combination of several attributes already reduces the number of matching content. Spending about 50%, 30%, 15%, and 5% on the first, second, third, and fourth attribute resulted in a good search performance.

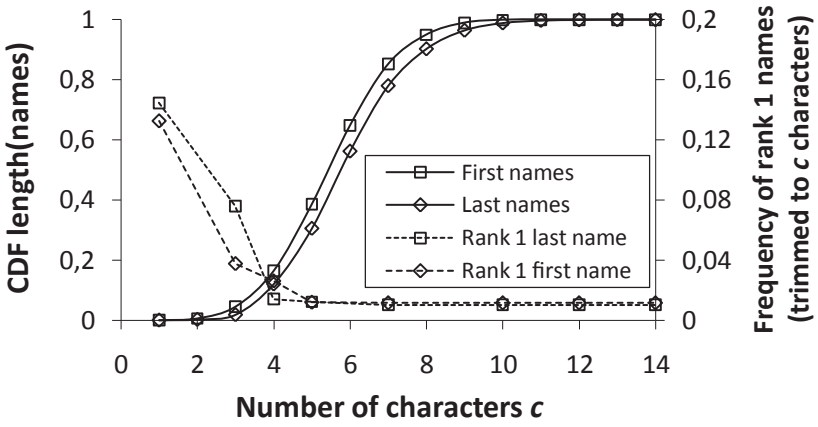


Figure 7.8.: CDF of length of first/last name

	0..9	A..Z	ASCII	Unicode
128 bit	40	25	16	8
256 bit	80	50	32	16

Table 7.1.: Number of letters fitting in a 128 or 256-bit ID

Storage Load Assuming a perfect load balancing mechanism, the number of resources which need to be stored by a single node is limited by the number of occurrences of the most common ID. IDs will be less frequent if more attributes with more characters are used to generate the ID. In the optimum case every entity is unique. However, queries would lead to large and disjoint ID ranges. For attribute compositions presenting a good trade-off, the most frequent entry occurs about one thousand times, like the ID (5L 3F 2T).

Search performance To analyze the search performance, we measure the number of nodes which are contacted during a query. We thereby neglect those nodes which are initially contacted in order to locate ID_{start}, as this number depends on the underlying DHT and not on our algorithm. Figure 7.9 shows the CDF of the number of nodes that will have to be queried during a search if only one attribute (5L), two attributes (5L 3F), or three attributes (5L 3F 2T) are used. Solid lines represent queries for all kind of attribute values, whereas dotted lines are obtained by queries without wildcard characters. In this scenario wildcards occur due to attribute values that are shorter than the number of used characters for the corresponding attribute. All curves were obtained using the results of 100,000 lookups for random persons.

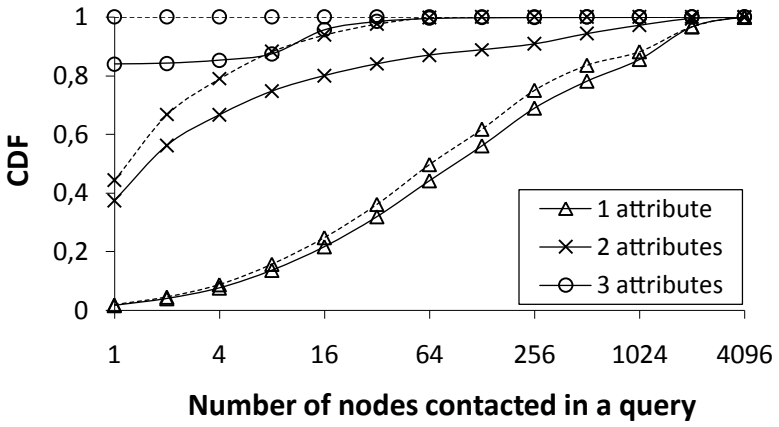


Figure 7.9.: Number of nodes contacted in a query

The topmost curve shows that one single node has to be contacted to answer a detailed and well specified query. If wildcards are allowed and a wildcard happens to appear in the main attribute, up to 64 nodes might have to be queried to cover the resulting ID ranges. However, in more than 80% of all queries only a single node needs to be

contacted. The number of nodes, which need to be contacted, will increase if only two or one attribute are specified in the query. In the latter case, up to 4096 nodes out of one million might have to be contacted in order to complete the query. Moreover, in the worst case it might be possible, that only one out of these 4096 nodes actually holds matching content.
To find an optimal number of attributes, we look at the average number of resources on a peer which match a specific query. Figure 7.10 shows the corresponding Probability Density Function (PDF) for queries consisting of one (5L), two (5L 3F), and three (5L 3F 2T) attributes using bins of size five. If only one single attribute is used in a query, the query will be too imprecise and most peers return between 25 and 30 matching resources. Also note that in addition the query will be send to quite many peers as indicated in the discussion of the last figure. Specifying more attributes leads to more precise queries. Thus, fewer matching resources per queried peer exist and fewer peers have to be queried in total. However, if too many attributes are specified, it will become more likely that queried peers do not have any matching resources and will therefore be contacted unnecessarily.

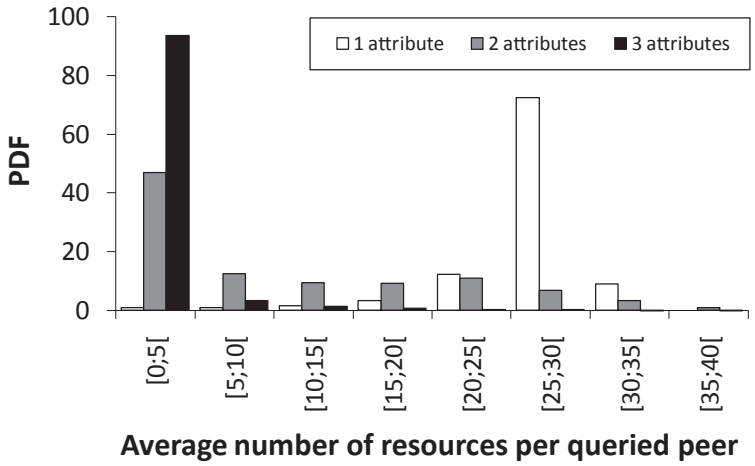


Figure 7.10.: Average number of matching resources per queried peer

7.3.4. Conclusion

One of the main drawbacks of DHT-based overlay networks is that they inherently support only hash table functionality, i.e., queries for exact keywords. More advanced queries, like wildcard, range and complex queries, have to be implemented on top of the basic DHT protocol. There are different approaches in the literature of how to realize such complex queries in a relatively general context. In this section we therefore introduced a novel keyword-based query algorithm, which was designed to enable advanced lookups for persons in community services.
Thereby, the specific composition of our multi-attribute keys allows range and wildcard queries. That way a user does no longer have to know the exact search term, but may issue fuzzy queries covering several possible keywords. Due to the prefix based approach, the wildcard functionality is constrained. It is, for example, not possible to solely search

for the last three characters of a person's name. However, such queries are very unlikely and can be neglected in this kind of application. By combining our algorithm with other well known approaches from literature, like fusion dictionaries, we were able to avoid the problem of very common keywords.
We evaluated our algorithm by means of simulation, using the example of a German phone book with 27 million entries. In particular, we derived optimal input parameters for the given scenario and showed that our algorithm works well in a realistic environment. Including the phone books of other countries into our distributed index is expected to improve the uniformity of the stored keywords, as the most common family names differ significantly in individual countries. Asian countries, in particular, use an entirely different set of Unicode characters and thus do not increase the peaks observed for common Latin names. Also, for most applications, stating the country a person lives in and using a country code as the main attribute in the ID will distribute different countries to different parts of the ID space.

7.4. Conclusion

VoIP-based communication systems are becoming more and more popular as they offer cheap (or even free) calls, especially to/from foreign countries. Thereby, services are important features which account for user satisfaction and revenue of operators. Yet, the realization of many services is currently based on central instances, e.g., SIP proxy/registrar servers.
We introduced a framework for realizing services in a decentralized DHT-based architecture. Thereby, contact information of users (*contact sheets*) is stored in the DHT. Also, services are realized by inserting *info profiles* into the DHT. Due to the DHT, the framework is independent of a specific platform or existing infrastructure, thus supporting ad hoc networks and ephemeral groups. The framework also provides an easy way to implement and launch new services. By using gateways, existing APIs, or open protocols, the framework may integrate any communication system and extend these systems with services, which are available in the framework.
The most critical part of our framework is the underlying DHT. Yet, we are quite confident that operators can establish a carrier-grade system by using a DHT, which implements improvements introduced in this thesis and in related work.
One basic part of each communication system is the lookup for other participants. In order to realize a decentralized version of such a user lookup, the DHT has to be extended to advanced query functionality. In the previous section we discussed related work and showed that no satisfying solution is proposed. By combining and extending these solutions, we introduce PriMA KeyS, a novel prefix-based multi-attribute keyword search algorithm. The algorithm is fully distributed and pays special attention to low network traffic and balanced query and storage load distributions.

CHAPTER 8

Conclusion, Discussion and Outlook

The ongoing shift from client-server architectures to decentralized Peer-to-Peer architectures is fostered by the enormous potential available at the edges of the network. Moreover, the inherent structure of P2P networks naturally resembles the connections between communicating groups. Thereby, these networks are self-organizing—although each entity solely bases its behavior on a few simple rules (the protocol) and its local observations (communication with a few selected entities), a powerful, scalable, robust and flexible organization (the network) is created at a macroscopic level.
In this thesis, we focused on structured P2P networks, which apply proactive routing, i.e., a deterministic overlay topology is maintained in order to provide guarantees on the lookup performance. DHTs are the most common approach to realize the necessary overlay. In particular, we evaluated the Chord protocol, a well-known representative of DHTs. Chord is based on basic algorithms, which establish a plain, circular overlay topology. Yet, this structure is highly flexible and its performance in terms of lookup path length, robustness, and signaling overhead is similar to other DHT protocols.
Ideally, P2P networks require no additional operator interaction, as the system is adapting automatically to the current state of the system, thus, offering important benefits for operators and customers. Service providers, for example, may save infrastructure costs by exploiting free resources available at the end devices. As a consequence, customers, e.g., profit from reduced prices and novel services. Another characteristic of self-organizing systems is that even though rules are simple and may be well understood, describing and predicting the behavior of the whole system in detail is very difficult. In this thesis we chose both a simulative and a analytical approach to evaluate the performance of these systems.
We verified that structured P2P overlay networks scale to extremely large numbers of users. Moreover, churn (i.e., joining, leaving and failing of users) is expected and not critical for the system, except for extreme churn rates. A high fluctuation of peers, however, will degrade the performance of the system and, eventually, the system will break down. We also showed, that it is important to adapt the configuration of the

design parameters to the current behavior of the system, whereas, the influence of the size of the overlay is rather small.
In Chapter 6, we evaluated existing modifications and extensions to structured overlay networks. Based on these results, we developed algorithms for improving the robustness and performance of structured P2P networks. We showed that the correctness and stability of the overlay can be increased by sending token-like stabilization messages, which exploit the circular structure of the Chord protocol. Furthermore, transmitted messages in our approach experience less timeouts, thus, increasing the lookup performance as a side-effect.
We also pointed out that in spite of a more stable overlay, the probability of disruptions and fragmentations of the overlay must not be neglected. We discussed several design choices, which reduce the probability of such disruptions. As a consequence, we introduced mechanisms, which are able to detect multiple partitions and recover the overall network topology.
Additionally, we evaluated state-of-the-art algorithms for further improving the lookup performance of structured overlay networks. By combining recursive with iterative routing, we managed to defy the individual weak points of both approaches. Additional freebie finger entries were used to exploit existing finger update traffic. Thus, we were able to reduce the mean lookup path length by approximately 50%. In order to allow an easy and comprehensive combination of various routing heuristics, we introduced a route selection method based on fuzzy logic.
As a final point, we presented a Voice-over-IP (VoIP) service framework as a show-case of the benefits of structured P2P networks. By means of integrating both legacy as well as innovative services, we demonstrated that structured P2P networks are capable of providing fully-distributed carrier-grade applications. In particular, we discussed the realization of distributed white pages and presented a novel prefix-based multi-attribute user lookup.
Summarizing, we demonstrated that structured overlay networks provide the technical feasibility of fully distributed (P2P-based) carrier-grade applications. However, in order to be suitable for an even wider field of applications, especially security remains an open issue and is thus an area of ongoing research [SM02, CDG+02b, Wal03, See06]. Most deployed systems disregard security at all or rely on centralized solutions. Currently, a main focus in research is on trust and reputation management [BM06]. However, further research on fully decentralized security mechanisms, which may be deployed in spontaneous, infrastructure-less ad hoc networks and ephemeral groups, is required.
Moreover, novel concepts for the Next Generation Internet (NGI) evolve, which integrate fundamental support for P2P applications [FDKC06], replace the current Internet name resolution with DHT-based alternatives [PMTZ06], provide an alternative to the IP layer [EFK03], or even think about a complete P2P-based clean-slate design [HSKE09].

APPENDIX A

Abbreviations and Symbols

List of Abbreviations

CDF Cumulative Distribution Function

DHT distributed hash table

DoS Denial-of-Service

FRS Fuzzy-based Route Selection

GNP Global Network Positioning

GUI Graphical User Interface

GUID Globally Unique Identifier

ID Identifier

IN Intelligent Network

ISP Internet Service Provider

LNS Long Lifetime Node Selection

MTBJ Mean Time Between Two Joins

MTTL Mean Time To Leave

NAT Network Address Translation

NED Negative Exponential Distribution

NGI	Next Generation Internet
P2P	Peer-to-Peer
PDF	Probability Density Function
PNS	Proximity Neighbor Selection
PRS	Proximity Route Selection
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RPC	Remote Procedure Call
RTT	Round-Trip Time
SIP	Session Initiation Protocol
SPoF	Single Point of Failure
TLS	Time Last Seen
TT	Transmission Time
TTL	Time-to-Live
UAC	User Agent Client
URI	Uniform Resource Identifier
UID	Unique Identifier
VoIP	Voice-over-IP

List of Symbols

m	Size [in bit] of the ID space
d	Dimensionality of the ID space
b	Base of the ID space
$\mathcal{K}$	Set of keys
k	Key, $k \in \mathcal{K}$
K	Number of keys: $\lvert\mathcal{K}\rvert$
$\mathcal{N}$	Set of nodes
n	Node, $n \in \mathcal{N}$, with ID n
N_{total}	Number of nodes: $\lvert\mathcal{N}\rvert$
N	Number of live nodes
n_{new}	New node
$n_{\text{bootstrap}}$	Bootstrap node (used for bootstrapping)
p, s	Predecessor, Successor of node n
$\mathcal{F}$	Set of (Freebie) Fingers
f	(Freebie) finger, $f \in \mathcal{F}$
F	Number of (freebie) fingers: $\lvert\mathcal{F}\rvert$
C	Size of the cache
c_n, c_m	Coordinate of a node n, monitor m
c	Constant
$\mathcal{L}$	List (of successors)
L	Size of list: $\lvert\mathcal{L}\rvert$
$\mathcal{R}$	Replication group
R	Size of Replication group: $\lvert\mathcal{R}\rvert$
i	Index, e.g., of a finger entry
α	Number of replies from $\mathcal{R}$, which must be received in order for a query to be counted successful
β	Number of parallel queries
κ	Bucket size in Kademlia
t_{stab}	Neighbor stabilization period
t_{fu}	Finger update period
t_{TLS}	Timeout value of entries in Freebie Finger cache
t_{hop}	Hop timeout value
t_{search}	Search timeout value

TTL	TTL counter
r_{join}	Join rate
$\mathcal{H}$	Hash function
$\Delta(x, y)$	Distance function (according to some specific distance metric)
hops	Estimate of remaining number of hops
ones(x)	Number of ones in the bitwise representation of x
E$[X]$	Expected value of a random variable X
$T_{\text{on}}, T_{\text{off}}$	Random variables describing the duration of online sessions and offline periods

APPENDIX B

Simulation environment

File syntax

Source file syntax (scenario description)

- `peers <uMax> <assign>` – Determine the maximum number of peers
- `fullchord <num> <E(T_avg)>` – Generate a chord ring by joining *num* nodes and resources at a mean rate of $E[T_{avg}]$
- `newchord <num> <E(T_avg)>` – Generate a new chord ring by joining *num* nodes at a mean rate of $E[T_{avg}]$, but inserting no resources
- `user <duration> <E(T_on)> <E(T_off)> <E(T_search)>` – Simulate user behavior for *duration* seconds with the parameters $E[T_{on}]$, $E[T_{off}]$, and $E[T_{search}]$
- `expsearch <duration> <E(T_search)>` – Simulate every node searching for a random resource for *duration* seconds with a mean inter-search period of $E[T_{search}]$ ms
- `fullsearch <esearch> <random>` – Simulate every node searching for every present resource with a mean inter-search period of $E[T_{search}]$ ms
- `randomsearch <number> <esearch>` – Simulate random nodes searching for random resources for a *number* of times with a mean inter-search period of $E[T_{search}]$ ms
- `joinpeer <number> <E(T_join)> <random>` – Join *number* of nodes with a given ($random = 0$) or a random time $E[T_{join}]$ between joins ($random = 1$)
- `randomleave <number> <eleave> <random>` – Fail *number* of nodes with a given ($random = 0$) or a random time $E[T_{leave}]$ between fails ($random = 1$)
- `wait <time>` – Waiting phase of *time* ms

Event file syntax

At `<time>` a basic operation for node `<n>` is executed.

- `<time> join <n>` – Node n joins
- `<time> assign <n> <k_1> ... <k_x>` – Multiple resources with keys $k_1..k_x$ are assigned to node n
- `<time> leave <n>` – Node n fails
- `<time> insert <n> <k>` – Node n insert a resource with key k
- `<time> find <n> <k>` – Node n initiates a query for key k
- `<time> exit` – Exit the simulation

DIF file syntax

Remove existing events These commands remove existing events within the period t_{start} to t_{end}. If t_{end} is set to 0 the command will be valid until the end of the simulation. If t_{start} and t_{end} are not defined, their value will be assumed to be 0, thus events are removed from the complete simulation.

- `NOTDOCUMENT <k> <t_start> <t_end>` – Remove all events for key k
- `NOTNODE <n> <t_start> <t_end>` – Remove all events for node n
- `NOTJOIN <n> <t_start> <t_end>` – Remove all join events for node n
- `NOTLEAVE <n> <t_start> <t_end>` – Remove all leave events for node n
- `NOTSEARCH <k> <t_start> <t_end>` – Remove all search events for key k

Add new events These commands add new events to the simulation at the specified `<time>`. They have a higher priority than the remove commands, e.g., if a `NOTNODE` and a `DIFFJOIN` command are defined, the node will be joined, although all other events for this node will be removed. Thus, all existing events can be removed and new events can be defined for certain nodes.

- `DIFFJOIN <n> <time>` – Node n joins
- `DIFFLEAVE <n> <time>` – Node n fails
- `DIFFINSERT <n> <time> <DocID>` – Node n inserts a resource with key k
- `DIFFFIND <n> <time> <DocID>` – Node n initiates a query for key k

INIT file syntax

- `DEBUG` – Switch DEBUG on (1)/off (0). If DEBUG is switched on (DEBUG = 1) the following parameters and their values will be printed to the screen.

Search-related parameters

- `ONEHOPTIMEOUT` – Hop timeout in ms
- `SEARCHTIMEOUT` – Search timeout in ms
- `TTL` – Maximum number of hops in order to avoid loops
- `ALPHA` – All nodes of the appropriate replication group r are expected to answer queries. A query will be counted as successful if the initiator receives at least `ALPHA` ($\alpha \leq R$) answers.

Routing-related parameters

- `PACKET_LOSS` – Percentage of lost packets
- `AVG_PKT_DELAY` – Mean Transmission Time (TT) (Negative Exponential Distribution (NED))

Queue-related parameters These parameters adapt the event queue. Different queues may be selected in `def.h`.

- `MAX_NUM_ENTRIES` – Maximum number of entries
- `ENTRY_TUNIT` – Duration (in milliseconds) spanned by one entry
- `REALLOC_INCREMENT` – Amount of incremental reallocation

Node-related parameters Parameters describing the behavior of nodes.

- `NUM_DOCS` – Number of resources per node (size of local database)
- `REPLICATION_GRADE` – Size of the replication group
- `NUM_NEIGHBOURS` – Number of successors/predecessors (= $1/2$ number of neighbors)

Parameters related with stabilization, finger update, and republish

- `STABILIZE_PERIOD` – Time in ms between two stabilization calls
- `REPUBLISH_PERIOD` – Time in ms between two republish calls
- `FU_PERIOD` – Time in ms between two finger update calls
- `NUM_FINGERS` – Maximum number of fingers stored for each finger interval

Parameters related to the statistic function These parameters control the output of the statistic function. Defining the parameter `PRINTSTAT` is mandatory.

- `PRINTSTAT` – Time in 100 ms between two calls of the statistic function. This parameter must be specified to be able to use the following statistics. Sums and mean values are calculated for the last `PRINTSTAT` · 100 ms. A cumulated value is the sum of that value for all live nodes.
- `PNBL` – Print the mean number of errors in the complete neighbor lists
- `PRGR` – Print the mean number of errors in the replication groups
- `PSUL` – Print the mean number of errors in the successor lists
- `PPRL` – Print the mean number of errors in the predecessor lists
- `PCHU` – Print the number of join and leave events
- `CPCHU` – Print the cumulated number of join and leave events
- `PACK` – Print the total number of packets, sent packets, and lost packets
- `CPACK` – Print the cumulated number of packets, sent packets and lost packets
- `PBYT` – For all types of packets print the number of sent packets
- `CPBYT` – For all types of packets print the number of cumulated packets
- `SRCH` – Print the number of queries, successful queries, and erroneous queries (3 different types)
- `SRCH` – Print the cumulated number of queries, successful queries, and erroneous queries (3 different types)
- `SEARCHTIME` – Print the mean search duration
- `CSEARCHTIME` – Print the cumulated mean search duration
- `JOINTIME` – Print the mean time required for a node to successfully join the network
- `CJOINTIME` – Print the cumulated times required for nodes to successfully join the network
- `TIMEOUT` – Print the consumed USER and SYSTEM time since the simulation start
- `MEMOUT` – Print the current main storage consumption

Additional parameters

- `SEED` – [Optional], seed value for initiating the simulation. The seed parameter can also be specified as a command line parameter, thereby overwriting the INIT-file setting.

List of Figures

List of Tables

Bibliography

Publications by the author

[BHKE07] Andreas Binzenhöfer, Tobias Hossfeld, Gerald Kunzmann, and Kolja Eger, *Efficient simulation of large-scale p2p networks: Compact data structures*, Proceedings of the Workshop on Modeling, Simulation and Optimization of Peer-to-peer environments (MSOP2P) in conjunction with Euromicro (PDP '07), Feb 2007.

[BHKE09] ———, *Efficient simulation of large-scale p2p networks*, Accepted for International Journal of Computational Science and Engineering - Special Issue on 'Parallel, Distributed and Network-Based Processing' (2009).

[BKH07] Andreas Binzenhöfer, Gerald Kunzmann, and Robert 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 **20** (2007), no. 6, 625–641.

[EHBK07] Kolja Eger, Tobias Hossfeld, Andreas Binzenhöfer, and Gerald Kunzmann, *Efficient simulation of large-scale p2p networks: Packet-level vs. flow-level simulations*, Proceedings of the 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, Jun 2007.

[ESZK04] Jörg Eberspächer, Rüdiger Schollmeier, Stefan Zöls, and Gerald Kunzmann, *Structured p2p networks in mobile and fixed environments*, Proceedings of HET-NETs '04 International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks, Jul 2004.

[FKSK06] Michael Finkenzeller, Gerald Kunzmann, Rüdiger Schollmeier, and Andreas Kirstädter, *Critical-mass of a distributed end-system monitoring service*, Proceedings of the 2006 World Congress in Computer Science Computer Engineering, and Applied Computing (ICOMP '06), Jun 2006.

[HBS+05] Tobias Hoßfeld, Andreas Binzenhoefer, Danial Schlosser, Kolja Eger, Jens Oberender, Ivan Dedinski, and Gerald Kunzmann, *Towards efficient simulation of large scale p2p networks*, Tech. Report 371, University of Wuerzburg, 2005.

[HSKE09] Oliver Hanka, Christoph Spleiss, Gerald Kunzmann, and Jörg Eberspächer, *A novel DHT-based network architecture for the next generation Internet*, Proceedings of the International Conference on Networks (ICN '09), Mar 2009.

[KB06] Gerald Kunzmann and Andreas Binzenhoefer, *Autonomically improving the security and robustness of structured p2p overlays*, Proceedings of the International Conference on Systems and Networks Communications (ICSNC '06), Nov 2006.

[KBH05] Gerald Kunzmann, Andreas Binzenhöfer, and Robert Henjes, *Analyzing and modifying Chord's stabilization algorithm to handle high churn rates*, Proceedings of the 6th Malaysia International Conference on Communications (MICC) in conjunction with International Conference on Networks (ICON), Nov 2005.

[KBS08] Gerald Kunzmann, Andreas Binzenhöfer, and Fabian Stäber, *Structured overlay networks as an enabler for future internet services*, it - Information Technology **50** (2008), no. 6, 376–382.

[KKSZ06] Wolfgang Kellerer, Gerald Kunzmann, Rüdiger Schollmeier, and Stefan Zöls, *Structured peer-to-peer systems for telecommunications and mobile environments*, AEÜ - International Journal of Electronics and Communications **60** (2006), no. 1, 25–29.

[KNE05] Gerald Kunzmann, Robert Nagel, and Jörg Eberspächer, *Increasing the reliability of structured p2p networks*, Proceedings of the 5th International Workshop on Design of Reliable Communication Networks (DRCN '05), Oct 2005.

[KNH+07] Gerald Kunzmann, Robert Nagel, Tobias Hossfeld, Andreas Binzenhöfer, and Kolja Eger, *Efficient simulation of large-scale p2p networks: Modeling network transmission times*, Proceedings of the Workshop on Modeling, Simulation and Optimization of Peer-to-peer environments (MSOP2P) in conjunction with Euromicro (PDP '07), Feb 2007.

[KS06] Gerald Kunzmann and Rüdiger Schollmeier, *EUNICE 2005: Networks and applications towards a ubiquitously connected world*, vol. 196, ch. Exploiting the overhead in a DHT to improve lookup latency, pp. 247–254, Springer, 2006.

[Kun05] ———, *Iterative or recursive routing? hybrid!*, KiVS Kurzbeiträge und Workshop, Lecture Notes in Informatics, vol. 61, GI, Mar 2005, pp. 189–192.

[SBD+06] Amardeo Sarma, Christian Bettstetter, Sudhir Dixit, Gerald Kunzmann, Rüdiger Schollmeier, J. Nielsen, P. Santi, R. Schmitz, M. Stiemerling, Dirk Westhoff, and A. Timm-Giel, *Self-organization in communication networks*, pp. 423–451, Wiley, 2006.

[SK03] Rüdiger Schollmeier and Gerald Kunzmann, *GnuViz - mapping the Gnutella network to its geographical locations*, Praxis der Informationsverarbeitung und Kommunikation (PIK) **26** (2003), no. 2, 74–79.

[SK07] Christoph Spleiss and Gerald Kunzmann, *Decentralized supplementary services for Voice-over-IP telephony*, Proceedings of EUNICE 2007 (Enschede, Netherlands), Lecture Notes in Computer Science, vol. 4606, Jul 2007, pp. 62–69.

[SKM07] Fabian Stäber, Gerald Kunzmann, and Jörg P. Müller, *Extended prefix hash trees for a distributed phone book application*, Proceedings of the 13th International Conference on Parallel and Distributed Systems (ICPADS '07), Dec 2007.

General publications

[ABFW04] Vinay Aggarwal, Stefan Bender, Anja Feldmann, and Arne Wichmann, *Methodology for estimating network distances of Gnutella neighbors*, GI Jahrestagung (2) (Peter Dadam and Manfred Reichert, eds.), LNI, vol. 51, 2004, pp. 219–223.

[ACMDH03] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, and Manfred Hauswirth, *PIX-Grid: A platform for p2p photo exchange*, Proceedings of Ubiquitous Mobile Information and Collaboration Systems (UMICS '03), 2003.

[Ada00] Lada A. Adamic, *Zipf, power-laws and Pareto - a ranking tutorial*, Tech. report, Information Dynamics Lab, HP Labs, 2000.

[AH04] Karl Aberer and Manfred Hauswirth, *Peer-to-peer systems*, Practical Handbook of Internet Computing (Munindar P. Singh, ed.), Chapman Hall & CRC Press, Baton Rouge, 2004.

[ALPH01] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman, *Search in power-law networks*, Physical Review E **64** (2001), no. 4, 46135–46143.

[AS03] James Aspnes and Gauri Shah, *Skip graphs*, Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '03), Jan 2003, pp. 384–393.

[BAS04] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan, *Mercury: supporting scalable multi-attribute range queries*, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04), ACM Press, 2004, pp. 353–366.

[BC98] Paul Barford and Mark Crovella, *Generating representative web workloads for network and server performance evaluation*, Proceedings of the ACM Symposium on Parallel and Distributed Tools (SIGMETRICS '98), Jul 1998, pp. 151–160.

[BD06] Philippe Biondi and Fabrice Desclaux, *Silver needle in the Skype*, Black Hat Europe 2006, 2006.

[Bin08] Andreas Binzenhöfer, *Performance analysis of structured overlay networks*, Doctoral thesis, University of Würzburg, Feb 2008.

[BM06] Raouf Boutaba and Alan Marshall, *Special issue - management in peer-to-peer systems*, Computer Networks **50** (2006), no. 4, 469–596.

[BS04] Salman A. Baset and Henning Schulzrinne, *An analysis of the Skype peer-to-peer Internet telephony protocol*, Tech. report, Columbia University, New York, Sep 2004.

[BSH05a] Andreas Binzenhöfer, Dirk Staehle, and Robert Henjes, *On the fly estimation of the peer population in a Chord-based p2p system*, Proceedings of the 19th International Teletraffic Congress (ITC19), Sep 2005.

[BSH05b] Andreas Binzenhöfer, Dirk Staehle, and Robert Henjes, *On the stability of Chord-based p2p systems*, Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '05), Nov 2005.

[BSV03] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker, *Understanding availability*, Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Feb 2003.

[BTG04] Andreas Binzenhöfer and Phuoc Tran-Gia, *Delay analysis of a Chord-based peer-to-peer file-sharing system*, Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC '04), Dec 2004.

[CDG+02a] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan Wallach, *Secure routing for structured peer-to-peer overlay networks*, Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI '02), Dec 2002, pp. 299–314.

[CDG+02b] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach, *Security for structured peer-to-peer overlay networks*, Proceedings of the 5th Symposium on Operating Systems Design and Implementaion (OSDI '02), 2002.

[CDHR02] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron, *Exploiting network proximity in distributed hash tables*, Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo '02), Jun 2002.

[CFCS03] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely, *MAAN: A multi-attribute addressable network for grid information services*, Proceedings of the 4th International Workshop on Grid Computing (GRID '03), IEEE Computer Society, 2003, p. 184.

[CHHC06] Jerry C.-Y. Chou, Tai-Yi Huang, Kuang-Li Huang, and Tsung-Yen Chen, *SCALLOP: A scalable and load-balanced peer-to-peer lookup protocol*, IEEE Transactions on Parallel and Distributed Systems **17** (2006), no. 5, 419–433.

[CL99] Hyoung-Kee Choi and John O. Limb, *A behavioral model of web traffic*, Proceedings of the 7th Annual International Conference on Network Protocols (ICNP '99), IEEE Computer Society, 1999, p. 327.

[CMM02] Russ Cox, Athicha Muthitacharoen, and Robert T. Morris, *Serving DNS using a peer-to-peer lookup service*, Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Lecture Notes in Computer Science, no. 2429, Springer, Mar 2002.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, *Freenet: A distributed anonymous information storage and retrieval system*, Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Lecture Notes in Computer Science, vol. 2009, 2001, pp. 46–66.

[CZK05] Byung-gon Chun, Ben Y. Zhao, and John D. Kubiatowicz, *Impact of neighbor selection on performance and resilience of structured p2p networks*, Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS '05), 2005.

[DA06] Anwitaman Datta and Karl Aberer, *The challenges of merging two similar structured overlays: A tale of two networks*, Proceedings of the International Workshop on Self-Organizing Systems (IWSOS '06), 2006.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris, *Vivaldi: A decentralized network coordinate system*, Proceedings of the ACM SIGCOMM '04, Aug 2004.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, *Wide-area cooperative storage with CFS*, ACM SIGOPS Operating Systems Review **35** (2001), no. 5, 202–215.

[DLS+04] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris, *Designing a DHT for low latency and high throughput*, Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04), Mar 2004.

[DP80] D. Dubois and H. Prade, *Fuzzy sets and systems - Theory and applications*, Academic press, 1980.

[EA05] Sameh El-Ansary, *Designs and analyses in structured peer-to-peer systems*, Doctoral thesis, Royal Institute of Technology, Stockholm, Sweden, 2005.

[EAKAH04] Sameh El-Ansary, Supriya Krishnamurthy, Erik Aurell, and Seif Haridi, *An analytical study of consistency and performance of dhts under churn*, Technical Report SICS T2004:12, Swedish Institute of Computer Science, Oct 2004.

[EFK03] Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurthy, *PeerNet: Pushing peer-to-peer down the stack*, Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003.

[ES00] Tony Eyers and Henning Schulzrinne, *Predicting Internet telephony call setup delay*, Proceedings of the 1st IP-Telephony Workshop (IPTel 2000), Apr 2000.

[ES05] Jörg Eberspächer and Rüdiger Schollmeier, *Peer-to-peer systems and applications*, ch. First and Second Generation of Peer-to-Peer Systems, pp. 35–56, Springer, 2005.

[FDKC06] Thomas Fuhrmann, Pengfei Di, Kendy Kutzner, and Curt Cramer, *Pushing Chord into the underlay: Scalable routing for hybrid MANETs*, Interner Bericht 2006-12, Fakultät für Informatik, Universität Karlsruhe, 2006, `http://i30www.ira.uka.de/research/publications/p2p/`.

[FFRS02] Thomas Friese, Bernd Freisleben, Steffen Rusitschka, and Alan Southall, *A framework for resource management in peer-to-peer networks*, Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, Lecture Notes In Computer Science, vol. 2591, Springer, 2002, pp. 4–21.

[FJ92] Christos Faloutsos and H. V. Jagadish, *On B-tree indices for skewed distributions*, Proceedings of the 18th International Conference on Very Large Data Bases (VLDB '92), Morgan Kaufmann Publishers Inc., 1992, pp. 363–374.

[FKS05] Bryan Ford, Dan Kegel, and Pyda Srisuresh, *Peer-to-peer communication across network address translators*, Proceedings of the USENIX Annual Technical Conference (USENIX '05), 2005.

[GBL+03] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse, *Kelips: Building an efficient and stable p2p DHT through increased memory and background overhead*, Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003.

[GDJ06] Saikat Guha, Neil Daswani, and Ravi Jain, *An experimental study of the Skype peer-to-peer VoIP system*, Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS '06), 2006.

[GEBR+03] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller, *Hierarchical p2p systems*, Proceedings of the ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par '03), 2003.

[GEFB+04] L. Garcés-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross, *Data indexing in peer-to-peer DHT networks*, Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS '04), IEEE Computer Society, 2004, pp. 200–208.

[GGG+03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, *The impact of DHT routing geometry on resilience and proximity*, Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03), ACM Press, 2003, pp. 381–394.

[GLR04] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues, *Efficient routing for peer-to-peer overlays*, Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NDSI '04), Mar 2004.

[GMB03] Danilo Gligoroski, Smile Markovski, and Verica Bakeva, *On infinite class of strongly collision resistant hash functions "EDON-F" with variable length of output*, Proceedings of the 1st International Conference On Mathematics and Informatics for Industry, 2003.

[GSG02] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble, *A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems*, SIGCOMM Comput. Commun. Rev. **32** (2002), no. 1, 82–82.

[GSK06] Ingo Gruber, Rüdiger Schollmeier, and Wolfgang Kellerer, *Peer-to-peer communication in mobile ad hoc networks*, Ad Hoc & Sensor Wireless Networks (OCP Science Journals) **2** (2006), no. 3.

[GT06] G. Ghinita and Yong Meng Teo, *An adaptive stabilization framework for distributed hash tables*, Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS '06), IEEE, 2006.

[GYGM04] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina, *One torus to rule them all: multi-dimensional queries in p2p systems*, Proceedings of the 7th International Workshop on the Web and Databases (WebDB '04), ACM Press, 2004, pp. 19–24.

[HJS+03] Nicholas Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, *Skipnet: A scalable overlay network with practical locality properties*, Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), Mar 2003.

[HJTW03] Nicholas J. A. Harvey, Michael B. Jones, Marvin Theimer, and Alec Wolman, *Efficient recovery from organizational disconnects in skipnet*, Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003, pp. 183–196.

[HMT+05] Tobias Hoßfeld, Andreas Mäder, Kurt Tutschku, Phuoc Tran-Gia, Frank-Uwe Andersen, Hermann de Meer, and Ivan Dedinski, *Comparison of crawling strategies for an optimized mobile p2p architecture*, Tech. Report 356, University of Würzburg, Apr 2005.

[JMW03] Sushant Jain, Ratul Mahajan, and David Wetherall, *A study of the performance potential of DHT-based overlays*, Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems (USITS '03), 2003.

[Joh08] Wolfgang John, *On measurement and analysis of Internet backbone traffic*, Master's thesis, Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, 2008.

[KCC+07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye H. Kim, Scott Shenker, and Ion Stoica, *A data-oriented (and beyond) network architecture*, Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '07), ACM Press, 2007, pp. 181–192.

[KLKP08] Sebastian Kaune, Tobias Lauinger, Aleksandra Kovacevic, and Konstantin Pussep, *Embracing the peer next door: Proximity in Kademlia*, Eighth International Conference on Peer-to-Peer Computing (P2P '08), Sep 2008, pp. 343–350.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin, *Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web*, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC '97), ACM Press, 1997, pp. 654–663.

[KLVW04] Alexander Klemm, Christoph Lindemann, Mary K. Vernon, and Oliver P. Waldhorst, *Characterizing the query behavior in peer-to-peer file sharing systems*, Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04), ACM Press, 2004, pp. 55–67.

[KR88] Brian W. Kernighan and Dennis M. Ritchie, *The C programming languag*, Prentice Hall, Inc, 1988.

[KR04] David R. Karger and Matthias Ruhl, *Simple efficient load balancing algorithms for peer-to-peer systems*, Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures (SPAA '04), ACM Press, 2004, pp. 36–43.

[KRP05] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki, *Should Internet service providers fear peer-assisted content distribution?*, Proceedings of the ACM/USENIX Internet Measurement Conference (IMC '05), 2005, pp. 63–76.

[KSS05] Praveen Kumar, G. Sridhar, and V. Sridhar, *Bandwidth and latency model for DHT based peer-to-peer networks under variable churn*, Proceedings of the 2005 Systems Communications (ICW '05), IEEE Computer Society, 2005, pp. 320–325.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker, *Search and replication in unstructured peer-to-peer networks*, Proceedings of the 16th ACM International Conference on Supercomputing (ICS '07), Jun 2002.

[LCP+05] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim, *A survey and comparison of peer-to-peer overlay network schemes*, IEEE Communications Surveys and Tutorials **7** (2005), no. 2, 72–93.

[LH08] Jure Leskovec and Eric Horvitz, *Planetary-scale views on a large instant-messaging network*, Proceedings of the 17th international conference on World Wide Web (WWW '08), ACM Press, 2008, pp. 915–924.

[Li06] Jinyang Li, *Routing tradeoffs in dynamic peer-to-peer networks*, Ph.D. thesis, Massachusetts Institute of Technology, 2006.

[Li08] Jin Li, *On peer-to-peer (p2p) content delivery*, Peer-to-Peer Networking and Applications **1** (2008), no. 1, 45–63.

[LL04] Lintao Liu and Kang-Won Lee, *Keyword fusion to support efficient keyword-based search in peer-to-peer file sharing*, Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04), IEEE Computer Society, 2004, pp. 269–276.

[LLD04] Ben Leong, Barbara Liskov, and Erik D. Demaine, *EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management*, Proceedings of the 12th International Conference on Networks (ICON '04), Nov 2004.

[LMW98] Robert D. Love, M.D. Siegel Michael, and Kenneth T. Wilson, *Understanding token ring protocols and standards*, Artech House Publishers, Oct 1998.

[LS00] Jonathan Lennox and Henning Schulzrinne, *Feature interaction in Internet telephony*, Feature Interactions in Telecommunications and Software Systems VI, IOS Press, May 2000, pp. 38–50.

[LSG+04] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and Frans Kaashoek, *Comparing the performance of distributed hash tables under churn*, Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS '04), Feb 2004.

[LSM+05] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil, *A performance vs. cost framework for evaluating DHT design tradeoffs under churn*, Proceedings of the 24th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05), 2005, pp. 225–236.

[LSS02] Tim Lu, Shan Sinha, and Ajay Sudam, *Panache: A scalable distributed index for keyword search*, Tech. report, Massachusetts Institute of Technology, 2002.

[MA06] Luis R. Monnerat and Claudio L. Amorim, *D1HT: A distributed one hop hash table*, Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS '06), IEEE Computer Society, 2006, pp. 21–31.

[MCR03] Ratul Mahajan, Miguel Castro, and Antony Rowstron, *Controlling the cost of reliability in peer-to-peer overlays*, Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Lecture Notes in Computer Science, vol. 2735, Springer, 2003, pp. 21–32.

[MCVR03] Valentin A. Mesaros, Bruno Carton, and Peter Van Roy, *S-Chord: Using symmetry to improve lookup efficiency in Chord*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '03), Jun 2003.

[Mil67] Stanley Milgram, *The small world problem*, Psychology Today **1** (1967), 61–67.

[MKL+02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu, *Peer-to-peer computing*, Technical Report HPL-2002-57, HP Labs, Palo Alto, CA, USA, Mar 2002.

[MM02] Petar Maymounkov and David Mazières, *Kademlia: A peer-to-peer information system based on the XOR metric*, Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '01), Lecture Notes In Computer Science, vol. 2429, Springer, 2002, pp. 53–65.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval*, Cambridge University Press, 2008.

[MUG05] Tatsuya Mori, Masato Uchida, and Shigeki Goto, *Flow analysis of internet traffic: World Wide Web versus peer-to-peer*, Systems and Computers in Japan **36** (2005), no. 11, 70–81.

[Mus06] John Musser, *Web 2.0 principles and best practices - an O'Reilly Radar Report*, O'Reilly Media, Nov 2006.

[NBLR06] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai, *A survey of peer-to-peer network simulators*, Proceedings of the 7th Annual Postgraduate Symposium (PGNet '06), EPSRC, 2006.

[NM65] J.A. Nelder and R. Mead, *A simplex method for function minimization*, The Computer Journal **7** (1965), no. 4, 308–313.

[NW96] Hung T. Nguyen and Elbert A. Walker, *A first course in fuzzy logic*, CRC Press, Inc., 1996.

[NZ01] Eugene T. S. Ng and Hui Zhang, *Towards global network positioning*, Proceedings of the ACM SIGCOMM Internet Measurement Workshop, Nov 2001.

[PDGM06] Marcell Perényi, Trang Dinh Dang, András Gefferth, and Sándor Molnár, *Identification and analysis of peer-to-peer traffic*, Journal of Communications **1** (2006), no. 7, 36–46.

[PMTZ06] Vasileios Pappas, Dan Massey, Andreas Terzis, and Lixia Zhang, *A comparative study of the DNS design with DHT-based alternatives*, Proceedings of the IEEE Conference on Computer Communications (INFOCOM '06), 2006.

[PP01] Athanasios Papoulis and Unnikrishna S. Pillai, *Probability, random variables and stochastic processes*, McGraw-Hill Science/Engineering/Math, 2001.

[Pug90] William Pugh, *Skip lists: A probabilistic alternative to balanced trees*, Communications of the ACM **33** (1990), no. 6, 668–676.

[RD01] Antony Rowstron and Peter Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms, Lecture Notes in Computer Science, vol. 2218, 2001, pp. 329–350.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker, *A scalable content-addressable network*, Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01), ACM Press, Oct 2001, pp. 161–172.

[RGRK03] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz, *Handling churn in a DHT*, Tech. Report UCB/CSD-03-1299, EECS Department, University of California, Berkeley, 2003.

[RLS+03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica, *Load balancing in structured p2p systems*, Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003, pp. 68–79.

[RM06] John Risson and Tim Moors, *Survey of research towards robust peer-to-peer networks: Search methods*, Computer Networks **50** (2006), no. 17, 3485–3521.

[RPW04] Simon Rieche, Leo Petrak, and Klaus Wehrle, *A thermal-dissipation-based approach for balancing data load in distributed hash tables*, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN '04), IEEE Computer Society, 2004, pp. 15–23.

[RS05] Dario Rossi and Ion Stoica, *Gambling heuristics on a Chord ring*, Proceedings of the Global Telecommunications Conference (GLOBECOM '05), vol. 2, Nov 2005.

[RV03] Patrick Reynolds and Amin Vahdat, *Efficient peer-to-peer keyword searching*, Proceedings of International Middleware Conference, Lecture Notes in Computer Science, vol. 2672, Springer, Jun 2003, pp. 21–40.

[SBR03] Nima Sarshar, P. Oscar Boykin, and Vwani Roychowdury, *Percolation search in power law networks: Making unstructured peer-to-peer networks scalable*, Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P '04), IEEE Computer Society, 2003, pp. 2–9.

[Sch05] Rüdiger Schollmeier, *Signaling and networking in unstructured peer-to-peer networks*, Doctoral thesis, Technische Universität München, Germany, 2005.

[SD03] Rüdiger Schollmeier and Antoine Dumanois, *Peer-to-peer traffic characteristics*, Proceedings of the 9th EUNICE Open European Summer School (EUNICE '03), Sep 2003.

[See06] Jan Seedorf, *Security challenges for peer-to-peer SIP*, IEEE Network **20** (2006), no. 5, 38–45.

[SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, *A measurement study of peer-to-peer file sharing systems*, Proceedings of the Multimedia Computing and Networking Conference (MMCN '02), Jan 2002.

[SGG03] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble, *Measuring and analyzing the characteristics of Napster and Gnutella hosts*, Multimedia Systems **9** (2003), no. 2, 170–184.

[SM02] Emil Sit and Robert Morris, *Security considerations for peer-to-peer distributed hash tables*, Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Mar 2002.

[SMK+01a] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, *Chord: A scalable peer-to-peer lookup service for Internet applications*, Proceedings of the 2001 ACM SIGCOMM Conference, Aug 2001, pp. 149–160.

[SMK+01b] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, *Chord: A scalable peer-to-peer lookup service for Internet applications*, Tech. Report TR-819, MIT, Mar 2001.

[SOTZ05] Yanfeng Shu, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou, *Supporting multi-dimensional range queries in peer-to-peer systems*, Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P '05), IEEE Computer Society, 2005, pp. 173–180.

[SP04] Cristina Schmidt and Manish Parashar, *Enabling flexible queries with guarantees in p2p systems*, IEEE Internet Computing **8** (2004), no. 3, 19–26.

[SR02] Alan Southall and Steffen Rusitschka, *The resource management framework: A system for managing metadata in decentralized networks using peer-to-peer technology*, Agents and Peer-to-Peer Computing, First International Workshop (AP2PC '02), Lecture Notes in Computer Science, vol. 2530, Springer, 2002, pp. 144–149.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark, *End-to-end arguments in system design*, ACM Transactions on Computer Systems **2** (1984), no. 4, 277–288.

[Sri01] Kunwadee Sripanidkulchai, *The popularity of Gnutella queries and its implications on scaling*, Tech. report, Carnegie Mellon University, Feb 2001.

[SSDN02] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl, *HyperCuP–hypercubes, ontologies and efficient search on p2p networks*, Proceedings of the 1st Workshop on Agents and P2P Computing (AP2PC '02), 2002.

[SSR05] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld, *Chord#: Structured overlay network for non-uniform load distribution*, Tech. report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Aug 2005.

[SSR08] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld, *Range queries on structured overlay network*, Computer Communications **31** (2008), no. 2, 280–291.

[Sto01] Damien Stolarz, *Peer-to-peer streaming media delivery*, Proceedings of the 1st International Conference on Peer-to-Peer Computing (P2P '01), IEEE Computer Society, Aug 2001, p. 48.

[SW05] Ralf Steinmetz and Klaus Wehrle (eds.), *Peer-to-peer systems and applications*, Lecture Notes in Computer Science, vol. 3485, Springer, 2005.

[TB04] Andreas Tasch and Oliver Brakel, *Location based community services*, Proceedings of the IADIS Conference on Web Based Communities (WBC '04), Mar 2004.

[TC03] Liying Tang and Mark Crovella, *Virtual landmarks for the Internet*, Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, Oct 2003, pp. 143–152.

[TC04] Liying Tang and Mark Crovella, *Geometric exploration of the landmark selection problem*, Passive and Active Network Measurement, 5th International Workshop, PAM 2004, Lecture Notes in Computer Science, vol. 3015, May 2004, pp. 63–72.

[TJ07] Guang Tan and Stephen A. Jarvis, *Stochastic analysis and improvement of the reliability of DHT-based multicast*, Proceedings of the 26th Annual IEEE Conference on Computer Communications (INFOCOM '07), IEEE, 2007, pp. 2198–2206.

[TSG+01] Kelly Truelove, Clay Shirky, Lucas Gonze, Rael Dornfest, and Dale Dougherty, *2001 p2p networking overview: the emergent p2p platform of presence, identity, and edge resources*, O'Reilly, 2001.

[TTG05] Kurt Tutschku and Phuoc Tran-Gia, *Peer-to-peer-systems and applications*, ch. Traffic Characteristics and Performance Evaluation of Peer-to-Peer Systems, pp. 383–397, Springer, 2005.

[UB03] Herwig Unger and Thomas Böhme, *A probabilistic money system for the use in p2p network communities*, Virtual Goods Summit 2007, 2003, pp. 60–69.

[Wal03] Dan s. Wallach, *A survey of peer-to-peer security issues*, Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS 2002, Lecture Notes in Computer Science, vol. 2609, 2003, pp. 253–258.

[Wim06] Wilhelm Wimmreuter, *Einsatz netzwerkunabhängiger dienste*, e & i Elektrotechnik und Informationstechnik **123** (2006), no. 7, 283–287.

[WJ02] Jared Winick and Sugih Jamin, *Inet-3.0: Internet topology generator*, Tech. Report CSE-TR-456-02, Department of EECS, University of Michigan Ann Arbor, 2002.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing gigabytes: Compressing and indexing documents and images*, Morgan Kaufmann Publishing, 1999.

[WPL+06] Jui-Chieh Wub, Kuan-Jen Pengac, Meng-Ting Luac, Chang-Kuan Linac, Yu-Hsuan Chengac, Polly Huangade, Jason Yaoac, and Homer H. Chenacd, *HotStreaming: Enabling scalable and quality IPTV services*, Proceedings of the IPTV Workshop at the 15th International World Wide Web Conference (WWW '06), May 2006.

[ZDK06] Stefan Zöls, Zoran Despotovic, and Wolfgang Kellerer, *Cost-based analysis of hierarchical DHT design*, Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P '06), IEEE Computer Society, 2006, pp. 233–239.

[ZDK07] Stefan Zöls, Zoran Despotovic, and Wolfgang Kellerer, *Load balancing in a hierarchical DHT-based p2p system*, 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '07), Nov 2007.

[ZH05] Yingwu Zhu and Yiming Hu, *Efficient, proximity-aware load balancing for DHT-based p2p systems*, IEEE Transactions on Parallel and Distributed Systems **16** (2005), no. 4, 349–361.

[ZHS+04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz, *Tapestry: A resilient global-scale overlay*

for service deployment, IEEE Journal on Selected Areas in Communications **22** (2004), 41–53.

[Zim96] H.-J. Zimmermann, *Fuzzy set theory—and its applications (3rd ed.)*, Kluwer Academic Publishers, 1996.

[Zip32] G. K. Zipf, *Selective studies and the principle of relative frequency in language*, Harvard University Press, 1932.

[Zün07] Maximilian Zündt, *A distributed community-based location service architecture*, Doctoral thesis, Technische Universität München, 2007.

[ZY06] Yingwu Zhu and Xiaoyu Yang, *Implications of neighbor selection on DHT overlays*, Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS '06), IEEE Computer Society, 2006, pp. 197–206.

Online documents

Note: These sources are documents in the Internet. We last accessed them in June 2008. In the meantime these sources or their URLs might have been changed, or they might even be removed by the corresponding provider. These articles or web pages exist only in electronic form. However, they provide important information and further reading. Also, part of this thesis is based on them. Thus, I could not forbear from citing these sources.

[Bli] Blizzard Entertainment, *World of Warcraft*, `http://www.worldofwarcraft.com/`.

[Boc] Thomas Bocek, *A distributed DNS prototype*, `http://distributeddns.sourceforge.net/`.

[BR] David Bryan and Brian Rosen, *IETF WG: Peer-to-peer session initiation protocol*, `http://tools.ietf.org/wg/p2psip/`.

[bri] *BRITE: Boston university representative internet topology generator*, `http://www.cs.bu.edu/brite/index.html`.

[cai] *Cooperative association for internet data analysis (CAIDA)*, `http://www.caida.org`.

[Cia] Ciao GmbH, *Ciao*, `www.ciao.de`.

[Cli00] Clip2, *The Gnutella 0.4 protocol specification*, http://dss.clip2.com/GnutellaProtocol04.pdf, 2000.

[Coh] Bram Cohen, *The BitTorrent protocol specification*, www.bittorrent.com/protocol.html.

[cro] *Croquet project*, http://croquetconsortium.org.

[eBa] eBay Inc., *eBay*, www.ebay.com.

[eBa08] eBay Inc., *eBay Inc. reports third quarter 2008*, http://ebayinkblog.com/wp-content/themes/ink/news-docs/q3-2008-release.pdf, Oct 2008.

[EJ01] Donald E. Eastlake and Paul E. Jones, *RFC 3174 - US secure hash algorithm 1 (SHA1)*, http://tools.ietf.org/html/rfc3174, Sep 2001.

[Fra] France Télécom - R&D Division, *Solipsis*, http://solipsis.netofpeers.net/.

[fre08] *FreePastry*, http://freepastry.org/FreePastry/, 2008.

[giF] giFT-FastTrack, BerliOS Developer, *Documentation of the known parts of the FastTrack protocol*, http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=HEAD&content-type=text/vnd.viewcvs-markup.

[GKL+] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling, *p2psim: a simulator for peer-to-peer protocols*, http://pdos.csail.mit.edu/p2psim/.

[Glo] Globus.org, *GT information services: Monitoring & discovery system (MDS)*, http://www.globus.org/toolkit/mds/.

[Goo] Google Inc., *Google*, www.google.com.

[Inf81] Information Sciences Institute, University of Southern California, *RFC 793 - transmission control protocol*, http://tools.ietf.org/html/rfc793, Sep 1981.

[ipo] ipoque GmbH, *Internet study 2007: P2p file sharing still dominates the worldwide Internet*, http://www.ipoque.com/media/internet_studies/internet_study_2007.

[Jin] Jini.org, *Jini lookup discovery service specification*, http://www.jini.org/wiki/Discovery.

[JSC+05] A. Johnston, R. Sparks, C. Cunningham, S. Donovan, and K. Summers, *Session initiation protocol service examples 08*, http://tools.ietf.org/html/draft-ietf-sipping-service-examples-08, 2005.

[Jup01] Jupiter Media Metrix, *Global Napster usage plummets, but new file-sharing alternatives gaining ground*, Press Release. `http://www.comscore.com/press/release.asp?id=249`, Jul 2001.

[Kli] Alexey Klimkin, **unofficial* eDonkey protocol specification v0.6.2*, `http://kent.dl.sourceforge.net/pdonkey/eDonkey-protocol-0.6.2.html`.

[KM02] Tor Klingberg and Raphael Manfredi, *Gnutella 0.6 RFC*, `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`, Jun 2002.

[Lon04] Matt Loney, *The magic that makes Google tick*, `http://www.zdnet.com.au/insight/software/soa/The-magic-that-makes-Google-tick/0,139023769,139168647,00.htm`, Dec 2004, ZDNet UK.

[Mac07] Colm MacCarthaigh, *Joost network architecture*, Presentation to the The United Kingdom Network Operators' Forum. `http://www.joost.com/forums/p/2007/04/joost-p2p-networking-presentation-by-colm-maccarthaigh-joosts-network-architect/`, Apr 2007.

[Mel04] Jorma Mellin, *Peer-to-peer networking - phenomenon and impacts to carriers*, Presentation at Telekom Forum, Sept 2004.

[Men05] Thomas Mennecke, *DSL broadband providers perform balancing act*, `http://www.slyck.com/news.php?story=973`, Nov 2005.

[Mer] Jean Mercier, *Skype numerology*, `http://skypenumerology.blogspot.com/`.

[Mic] Microsoft, *Microsoft Office Groove*, `http://office.microsoft.com/en-us/groove/default.aspx`.

[Mic07] Microsoft XNA Creators Club Online, *Network architecture: Peer-to-peer*, `http://creators.xna.com/en-us/sample/networkp2p`, Dec 2007.

[Mir] Mirabilis, *ICQ*, `www.icq.com`.

[Nap] Napster, LLC, *Napster*, `www.napster.com`.

[O'R05] Tim O'Reilly, *What is Web 2.0 - design patterns and business models for the next generation of software*, `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`, Sep 2005.

[RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, *RFC 3489 - session traversal utilities for NAT (STUN)*, `http://tools.ietf.org/html/rfc3489`, Oct 2008.

[RSC+02] J. Rosenberg, Henning Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *RFC 3261 - SIP: Session initiation protocol*, http://www.ietf.org/rfc/rfc3261.txt, Jun 2002.

[Rus06] Steffen Rusitschka, *CeBIT 2006 - Siemens peer-to-peer technology*, Video interview http://video.google.com/videoplay?docid=-2085361973487925857, 2006.

[Sha] Sharman Networks Ltd., *KaZaA*, www.kazaa.com.

[Sit02] Emil Sit, *Detecting malicious nodes in Chord*, http://sow.csail.mit.edu/2002/proceedings/sit.pdf, 2002.

[Skya] Skype Limited, *Online presence in Skype*, http://www.skype.com/business/allfeatures/presence/.

[Skyb] Skype Limited, *Skype*, www.skype.com.

[Sop] Sopcast, *Sopcast*, www.sopcast.com.

[Sou06] Alan Southall, *Siemens peer-to-peer technologies and their industrial application*, Talk at Dagstuhl Seminar 06131, http://www.peer-to-peer.info/seminar/schedule/dagstuhl-06131-southall.pdf, Mar 2006.

[Top] TopWare Austria, *D-Info 97*, Published on CD., TopWare Austria, 'D-Info 97'.

[UDD] *Universal description, discovery and integration (uddi).*

[U.S] U.S. Census Bureau, *1990 census name files*, http://www.census.gov/, accessed on April 2007.

[Web] Webdialogs, Inc., *Lotus sametime Unyte share for Skype*, http://www.webdialogs.com/unyte/.

[Whi] Whichvoip.com, *Guide to VoIP phone service features and terms*, http://www.whichvoip.com/voip/info/voip_features.htm.

[Wim06] Wilhelm Wimmreuter, *Elements for transition: ENUM & P2P complement the beasts*, Talk at 44. DFN-Betriebstagung, http://www.dfn.de/content/fileadmin/3Beratung/Betriebstagungen/bt44/forum_voip_siemens.pdf, Feb 2006.