Technische Universität München
Lehrstuhl für Kommunikationsnetze

Peer-to-Peer Networking in Heterogeneous Communication Environments

Stefan R. Zöls

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

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. sc. techn. Andreas Herkersdorf
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Jörg Eberspächer
	2.	Univ.-Prof. Dr.-Ing. Klaus Wehrle, Rheinisch-Westfälische Technische Hochschule Aachen

Die Dissertation wurde am 18.06.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 26.11.2008 angenommen.

Dedicated to my Wonderful Wife Susi

Abstract

This thesis analyzes requirements and develops solutions for the efficient use of the Peer-to-Peer (P2P) principle in heterogeneous communication environments. These are characterized by a large diversity in the resources of participating devices, in the applied network access technology, and in the user behavior. In this context we focus on structured P2P systems that are based on a distributed hash table (DHT).

We present measurements showing that state-of-the-art DHTs can hardly be applied in heterogeneous communication environments, and thus propose a hybrid design scheme for DHTs. In addition, we analyze different hierarchical DHT architectures. For this purpose, we develop an analytical cost model for system optimization. Based on this model, we compare different architectures and provide a full set of distributed algorithms to achieve and maintain an optimal system operation.

Zusammenfassung

Diese Arbeit analysiert Anforderungen und entwickelt Lösungen zur effizienten Nutzung des Peer-to-Peer (P2P) Prinzips in einem heterogenen Kommunikationsumfeld. Letzteres zeichnet sich durch Unterschiede in den Ressourcen der beteiligten Geräte, in der verwendeten Netzzugangstechnologie und im Nutzerverhalten aus. Dabei wird der Fokus auf strukturierte P2P-Systeme gelegt, die auf einer verteilten Hashtabelle (DHT) basieren.

Auf Grundlage von Messungen, die zeigen, dass herkömmliche DHTs kaum in einem heterogenen Kommunikationsumfeld verwendet werden können, wird ein hybrides Systemkonzept für DHTs vorgeschlagen. Ferner werden verschiedene hierarchische DHT-Architekturen analysiert und dazu ein analytisches Kostenmodell zur Systemoptimierung erstellt. Basierend auf diesem Modell werden verschiedene Architekturen verglichen und ein kompletter Satz verteilter Algorithmen entwickelt, um einen optimalen Systembetrieb zu gewährleisten.

Acknowledgements

When I started my doctoral studies in the area of Peer-to-Peer (P2P) systems in 2004, I quickly noticed that P2P networking means much more than just file sharing. The sophisticated algorithms, necessary to operate a continuously changing system without a central coordination entity, aroused my interest. At that time, the mainstream research on P2P systems focused on fixed IP networks and usually assumed functionally equal peers. Nevertheless, Prof. Dr. Jörg Eberspächer suggested to investigate the possibilities and challenges of extending the P2P paradigm to heterogeneous communication environments, characterized by a large diversity of the participating peers. Today, four years later, I hope this thesis can make a considerable contribution to this research field and propose suitable solutions for future communication networks.

During my research activities, I had the pleasure of working with many great people. First and foremost, I would like to thank my advisor Prof. Dr. Jörg Eberspächer for his continuous support, his great confidence and the fantastic working atmosphere at the Lehrstuhl für Kommunikationsnetze (LKN). The "Spirit of LKN" is not just a phrase, it really exists! I am also very pleased that Prof. Dr. Klaus Wehrle is the second examiner of this thesis. At several conferences and workshops, I got to know him not only as a great scientist but also as a very nice person.

The results presented in this thesis would not have been possible without the outstanding longtime cooperation between our institute and NTT DoCoMo Euro-Labs in Munich, Germany. During my whole time at LKN, I enjoyed the excellent collaboration with Dr. Wolfgang Kellerer, Dr. Zoran Despotovic, Anthony Tarlano and Maximilian Michel, our fruitful technical discussions and, not to forget, all our nice conference travels.

The working environment at LKN was very motivating and inspiring for me and made me always feel at home. Therefore, I thank all my current and former colleagues at LKN. In particular, I would like to thank Dr. Martin Maier, Sabine Strauß and Thomas Kurzhals for their great organizational and technical support. Furthermore, the technical discussions with Dr. Rüdiger Schollmeier, Dr. Ingo Gruber, Gerald Kunzmann, Quirin Hofstätter, Christoph Spleiß, Oliver Hanka, Michael Eichhorn, Stephan Eichler, Robert Nagel, Robert Vilzmann, Simon Schubert and many others were always highly interesting and resulted in numerous ideas for my work.

Last but not least, I would like to thank my beloved family and all my friends for their continuous and unconditional support. A very special thanks goes to my wonderful wife Susi. You are always there whenever I need you, you support me wherever possible, and you make – by far – the most delicate cakes in the world! Thank you so much!

Munich, Germany, June 2008 Stefan Zöls

“This machine is a server. DO NOT POWER IT DOWN!!”

Message from Tim Berners-Lee
on the front of the first
web server at CERN.

peer /pɪə(r) *noun*

“[usually pl.] a person who is the same age or who has the same social status as you”

Oxford Advanced Learner’s Dictionary.

Contents

1 Introduction and Motivation

Peer-to-Peer (P2P) systems have become an important research field in the area of Information and Communications Technology (ICT) in the last decade. Their distributed nature, reducing or completely avoiding the need of central servers, offers important benefits for operators and for customers of modern P2P-based communication systems. For example, the operator of a P2P-based communication system can reduce its expense and save infrastructure costs by exploiting the computing power of end-user terminals, while the user of a P2P system can easily share own content with and download content from other users. Many successful P2P applications such as Napster [NAP, Gre02], Gnutella [GNU], BitTorrent [BIT, Coh03], Skype [SKY, BS06, BD] or Groove Virtual Office [Cho] show the increasing importance of P2P-based communication systems.

In a P2P system, each participant ideally contributes to and benefits from the system in the same way. This equality of participants is reflected in the term "peer", and stands in contradiction to the clear division of tasks in classical Client-Server systems. In practice, each peer in a P2P system acts simultaneously as a server that shares own content with other peers, as a client that gains access to the content provided by other peers, and as a router that helps forwarding messages from a source peer to one or multiple destination peers.

P2P systems are distributed and self-organizing systems. Self-organization, the organization of the system without a central coordination entity, is achieved by local interactions among the peers and makes the system flexible and robust to so-called churn, i.e. continuous and frequent arrivals and departures of peers. P2P systems are built on top of an existing IP infrastructure, usually on top of the global Internet. Figure 1.1 shows this overlay principle. The connections among the peers emerge independently of the underlying physical network and the applied access networks. With reference to the ISO/OSI reference model, P2P systems are also characterized as "application layer networks", since their topology results from overlay connections among peers that are established by the P2P application.

A fundamental question in P2P systems is how to locate requested content, i.e. how to determine which peers share data items that match the querying peer's interests. This problem, to which we refer as "data lookup", arises because of the distributed nature of P2P systems. They typically avoid the presence of a central entity and thus require a completely self-organizing lookup strategy. Two basic approaches have evolved that try to solve the problem of self-organized data lookups: unstructured P2P systems and structured P2P systems.

Chronologically, unstructured P2P systems have been the first attempt to organize data lookups without a central entity. Their topology emerges randomly, by establishing connections from each peer to a number of randomly chosen other peers. To locate content, unstructured P2P systems typically flood query messages along these connections

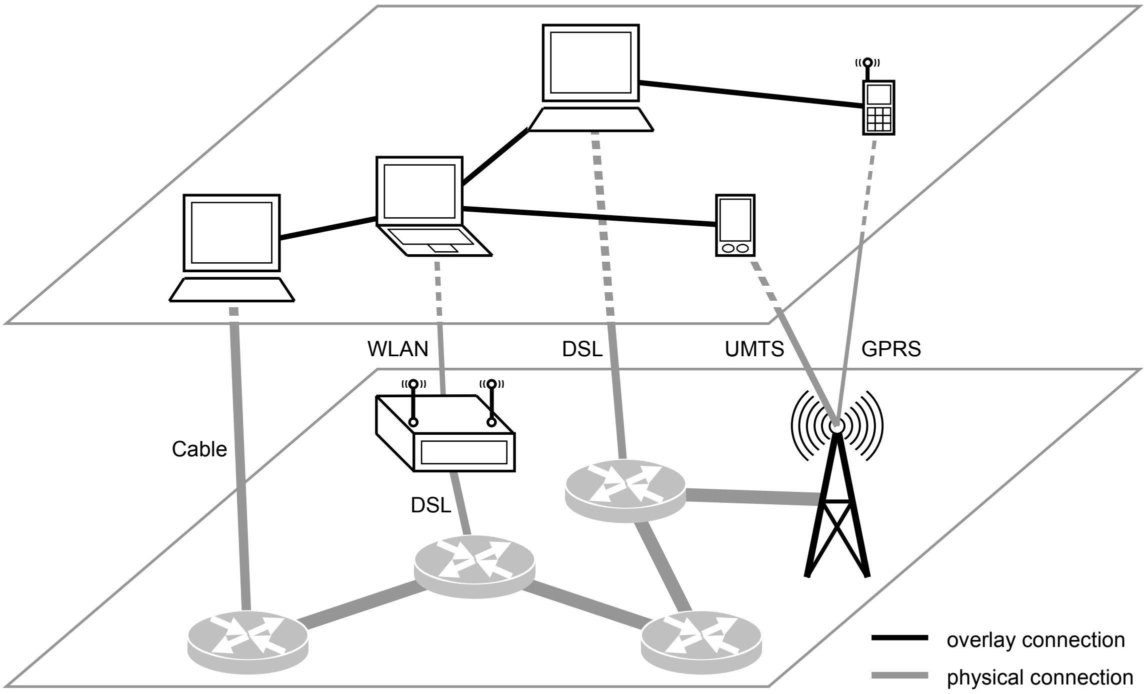


Figure 1.1: Overlay principle of P2P systems.

through the system. This search algorithm, however, cannot guarantee to find all peers in the system that share relevant content and usually results in a high signaling traffic.

To overcome these problems, structured P2P systems have been proposed that establish a predefined topology. Structured P2P systems make use of a Distributed Hash Table (DHT) to distribute index information about the shared data items across the participating peers, enabling their subsequent fast lookup. (Note that we use the terms "structured P2P system" and "DHT" synonymously in the following.) Although structured P2P systems offer significant advantages in comparison to unstructured P2P systems, they still pose many questions, e.g., how to deal with frequent peer arrivals and departures, with peer failures, or how to balance the load among the participating peers.

In this thesis, we address these challenges in the particular context of heterogeneous communication environments. With heterogeneous environments, we refer to communication systems that show a large diversity in the resources of participating devices, in the network access technology that connects participants to the system, and in the user behavior. This heterogeneity leads to special characteristics of the considered communication environments, which in turn poses additional challenges for the applied P2P system. The goal of this thesis is to identify these challenges, to analyze the resulting requirements of DHTs that are applied in heterogeneous communication environments, and to develop optimized solutions for their efficient use in such scenarios.

The main contributions of this thesis can be summarized as follows:

- We propose a novel hybrid design scheme for DHTs that are applied in heterogeneous communication environments.

- We present a thorough study of hierarchical two-tier DHT-based P2P systems. Their hierarchical system architecture allows distinction between the capacities of peers, thus building a solid foundation for P2P systems that are applied in heterogeneous environments. In this context, we develop a formal analytical cost model to determine the optimal operating point of the system in terms of a cost-optimal ratio between peers at the higher layer ("superpeers") and peers at the lower layer ("normal peers" or "leafnodes").

- Based on this cost model, we compare different hierarchical system architectures and show that a hierarchical system composed of a carefully chosen set of superpeers and normal peers attached to them is optimal.

- We propose a novel load balancing algorithm for hierarchical DHT-based P2P systems that is superior to state-of-the-art solutions and that is orthogonal to other existing load balancing algorithms for DHTs.

- We present a completely self-organizing hierarchical DHT-based P2P system. Its main design goal is cost-optimal operation, so we develop a set of distributed algorithms that achieve and maintain at any time the optimal operating point that is defined by our analytical model.

Throughout the thesis, we verify the efficiency of the proposed solutions by mathematical analyses and detailed computer-based simulations.

This thesis is organized as follows. Chapter 2 presents a survey of the basic principles of P2P systems and gives an overview of state-of-the-art solutions for data lookup in P2P systems. In particular, we discuss the special case of a centralized P2P system as well as the two basic approaches of unstructured and structured P2P systems. For each of these systems, we explain advantages and disadvantages and present well-known examples. We conclude with a detailed comparison of unstructured and structured P2P systems. Chapter 2 forms a tutorial on data lookup algorithms in P2P systems that provides a good overview particularly for those who are new to the topic.

A further preparatory chapter is Chapter 3, where we address heterogeneous communication environments. In detail, we explain the term "heterogeneous environments" and describe the main characteristics of such communication scenarios and the resulting challenges for the applied P2P system. Further, we present real-world measurements that show that state-of-the-art solutions can hardly be applied in heterogeneous environments.

Based on the findings of Chapter 3, we present a novel hybrid design scheme for DHTs in Chapter 4. It enables the efficient use of DHTs in the considered communication scenarios. After explaining how to set up such a hybrid DHT, we verify the improvements of our design scheme by mathematical analysis and simulations.

In a next step, we extend the ideas from Chapter 4 and analyze hierarchical DHT-based P2P systems that divide the participating peers in powerful superpeers and normal peers. In Chapter 5, we develop a formal analytical cost model that allows us to judge on the optimality of hierarchical DHTs, by determining the optimal ratio between superpeers and normal peers. Using this model we compare different hierarchical system architectures, and we show that a simple hierarchical system architecture composed of a DHT among a carefully chosen set of superpeers and normal peers attached to them is optimal in the sense that it minimizes the consumption of system resources, while maintaining system stability.

Chapter 6 presents a novel load balancing algorithm targeting hierarchical DHT-based P2P systems. Our algorithm balances the load among superpeers, which is a main prerequisite for the proper operation of hierarchical P2P systems. Further, it is orthogonal to other existing load balancing algorithms for DHTs. Chapter 6 explains the basic functionality of our algorithm, followed by detailed simulations that show a superior behavior (in terms of load balancing performance and generated overhead) of our algorithm compared to state-of-the-art solutions.

In Chapter 7, we make use of all previous results and develop a completely self-organizing hierarchical DHT-based P2P system whose main design goal is cost-optimal operation. We propose a set of distributed algorithms that achieve and maintain an optimal operating point of the system in terms of an optimal number of superpeers. In its optimal operating point, the system generates the minimal possible signaling traffic on condition that no superpeer is overloaded. Detailed simulations, run in a range of realistic settings, confirm the good performance of our solution and show significant advantages of a hierarchical system architecture compared to a flat one.

Chapter 8 provides an insight into our prototypical implementation of the proposed hierarchical DHT-based P2P system. In particular, we present an exemplary service – a distributed picture sharing service – that we have implemented for testing purposes.

Chapter 9 summarizes this work. Appendix A illustrates the software implementation of the simulator and the prototype. Appendix B presents a brief mathematical comparison of the routing performance of hierarchical and flat DHT-based P2P systems. Appendices C and D list the used abbreviations and mathematical variables.

Many contributions of this work are also documented as articles in journals [KKSZ06, ZDK08], as conference papers [ESZK04, ZSKT05, ZSH+05, ZDK06, ZDK07, ZHDK09], as workshop papers [ZSK05, ZETK06, ZSKD06b, KDM+07] and as poster presentation [ZSKD06a].

2 Basic Principles of P2P Systems

2.1 The P2P Paradigm

The great success of Napster, Gnutella and their successors was supported by three important developments in the area of communication networks. First of all, the end-users' communication devices, mainly personal computers with network access, have become powerful machines that are able to process large amounts of data very quickly. Second, the communication networks (particularly the access networks) have become fast enough to transmit huge data volumes from one end-user terminal to another. Finally, flat rates have more and more substituted time- or volume-based payment models, making it affordable for users to stay connected for a long time and thus resulting in a significantly increased mean session duration of participants.

All this has benefited the evolution of P2P as a new paradigm in networking and computing, whose fundamental idea is to shift the intelligence to the edges of the network. In other words, the content that is provided in a P2P system is generated, stored and demanded at the end-users' terminals. This is reflected in the three basic tasks of each peer in a P2P system: sharing own content with other peers, searching the system for content provided by other peers, and routing messages from a source peer to one or multiple destination peers.

Steinmetz and Wehrle characterize the P2P paradigm also as a perspective change in the view on Internet communication technology, "from coordination to cooperation, from centralization to decentralization, and from control to incentives" [SW05]. As a design principle for distributed systems, the P2P approach results in systems without central entities, making it essential for the participating peers to cooperate. To ensure this, appropriate incentive mechanisms ensure "a fair balance between give and take among peers" [SW05]. The bottom line is that the P2P paradigm differs heavily from the conventional Client-Server approach.

The distributed nature of P2P systems introduces a number of challenges. First of all, to join an existing P2P system, the joining peer has to know at least one peer that is already connected and serves as an entry point to the system. This bootstrapping problem is usually solved by the use of a bootstrap server, i.e. a permanently connected peer that is known by every joining peer, or by caching of addresses of peers that have been contacted in earlier sessions. Further, Steinmetz and Wehrle emphasize that an efficient lookup of data items is essential for the proper operation of a P2P system [SW05]. The challenge here is to find (preferably) all peers in the system that share relevant content, and at the same time to generate as little signaling traffic as possible on every query. Once the peers sharing relevant content have been found, the content has to be transferred from the providing peers to the requesting peer. In the literature, a number of strategies can be found how to do this efficiently in a distributed manner [HJ05]. Besides these main challenges, i.e. bootstrapping, data lookup and content dissemination, further aspects are important for the efficient and reliable operation of P2P systems. Typical examples which are also subject to ongoing research are trust and reputation management or digital rights management (DRM).

In this thesis, we concentrate on the problem of looking up data items in a P2P system. According to the organization of data lookups, P2P systems can be classified into three categories:

- Centralized P2P systems (as a special case of P2P)
- Unstructured P2P systems
- Structured P2P systems

In the following, we introduce these lookup concepts, explain their advantages and disadvantages and give well-known examples for each category.

2.2 Centralized P2P Systems

A centralized P2P system can be regarded as a special case of P2P systems, as it represents a mixture between a conventional Client-Server system and a P2P system. In a centralized P2P system (cf. Figure 2.1), every peer registers its shared data items at a central index server (step 0). Hence, this central database is scanned whenever a peer sends a query for a data item (steps 1 and 2). The subsequent data transfer is then performed in a decentralized manner, i.e. directly from peer to peer (steps 3 and 4) and without any interaction from the server.

A centralized P2P system offers two important advantages. First, the central entity has complete knowledge of all participating peers. This allows for an easy maintenance of the system. Second, every query can be resolved immediately and with a low signaling traffic, as the central entity also has complete knowledge of all shared (and thus registered) data items. This also ensures a guaranteed query resolution, i.e. the guarantee that a query for a specific data item returns all peers in the system that share this data item, and includes the possibility of negative answers, i.e. the information that the requested data item is currently not available in the system.

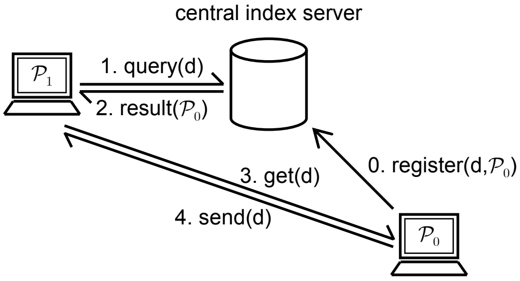


Figure 2.1: Data lookup and transfer in a centralized P2P system.

However, there are also disadvantages that outweigh the above benefits. The central index server forms a bottleneck in the system and a single point of failure. Thus, it must be equipped with extensive hardware resources to be able to process all incoming query requests, and it is vulnerable to attacks such as Denial-of-Service attacks. A further disadvantage of a centralized P2P system can be seen at the example of Napster [NAP, Gre02], the first popular P2P file-sharing system. In 2001, Napster had to shut down their index server due to a court decision. As a result, there was no possibility any more for Napster users to find the shared data items of other users, and Napster disappeared.

2.3 Unstructured P2P Systems

Shortly after the shutdown of Napster, unstructured P2P systems evolved. Due to their decentralized system architecture, they avoid a single point of failure. The term “unstructured” arises from the fact that the topology, i.e. the connections among the participating peers, emerges randomly and does not follow a predefined structure. The reason is that participating peers establish connections with a set of randomly chosen neighboring peers. Usually, these connections are maintained by periodically exchanged keep-alive messages.

In an unstructured P2P system, the shared data items are neither registered on a central index database like in a centralized P2P system (cf. Section 2.2) nor referenced on other peers in the system like in a structured P2P system (cf. Section 2.4). As a result, a flooding-based search algorithm is used that tries to discover all peers in the system that share relevant content. Certainly, such a search algorithm generates a high amount of signaling traffic which may overload the participating peers. In the following section, we discuss these aspects by taking a closer look on the Gnutella 0.4 protocol as a typical example of an unstructured P2P system.

2.3.1 The Gnutella 0.4 Protocol

The Gnutella 0.4 protocol specification [G04] is the groundwork of the popular Gnutella file-sharing application that is based on an unstructured P2P system. The specification includes four basic functionalities that any implementation must provide:

- Establishing a connection to the Gnutella system
- Probing the Gnutella system for other peers
- Searching the Gnutella system
- Appropriate routing of incoming messages

Connecting to the Gnutella system. Every joining peer connects to the Gnutella system by setting up a connection to another, already connected peer. The connection is initiated by a GNUTELLA CONNECT message from the joining peer. In case the contacted peer accepts the connection, it responds with a GNUTELLA OK message. The contacted peer

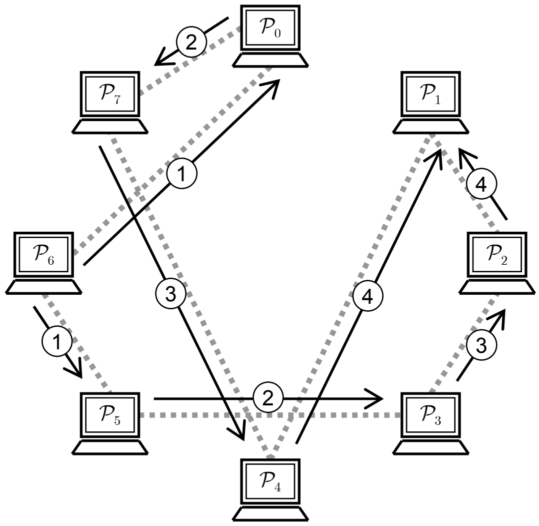


Figure 2.2: Flooding of a Query message through an unstructured P2P system.

also has the possibility to reject the connection request by sending any other response, e.g. if it already has a high number of neighbors.

Probing the Gnutella system for other peers. Every peer periodically sends Ping messages to its neighbors, which in turn forward the incoming Ping messages to their neighbors, and so forth. Additionally, every peer that receives a Ping message can respond with a Pong message. The Pong message is transmitted along the same path that carried the incoming Ping message, thereby informing all peers along this path about the responding peer's presence. As a result, every peer is able to establish more than one connection to other peers in the Gnutella system, and every peer is aware of other active peers that can be contacted in case that a neighboring peer leaves the system.

Searching the Gnutella system. As already mentioned, a flooding-based search algorithm is used in unstructured P2P systems. Whenever a peer in the Gnutella system initiates a query for a particular data item, it generates a Query message containing all relevant search criteria. Then it sends this Query message to all its neighbors. Like Ping messages, every peer forwards incoming Query messages to all its neighbors (except the one that has sent the message), so the Query message is flooded through the system. In addition to forwarding the Query message, every peer checks if it shares data items that meet the specified search criteria, and if this is the case it responds with a Query Hit message.

Figure 2.2 shows an example where every peer has two direct neighbors. Peer $\mathcal{P}_6$ is searching the Gnutella system, so in a first step it sends the Query message to its direct neighbors $\mathcal{P}_0$ and $\mathcal{P}_5$. In a second step, $\mathcal{P}_0$ and $\mathcal{P}_5$ forward the Query message to their neighbors $\mathcal{P}_7$ and $\mathcal{P}_3$, respectively. Note that they do not transmit the Query message back to $\mathcal{P}_6$, as $\mathcal{P}_6$ is the peer that has sent the Query message to them. The Query message is further flooded through the Gnutella system and reaches $\mathcal{P}_4$ and $\mathcal{P}_2$ after three hops. Finally, after four hops, it arrives and terminates at $\mathcal{P}_1$.

Gnutella 0.4 uses two mechanisms to prevent messages from being forwarded infinitely through the system. First, every message contains a unique identifier. Whenever a peer receives a message it checks by means of the identifier if it has already received the same message earlier, and if so, it discards the message and does not forward it to its neighbors. Second, every message contains a time-to-live (TTL) counter that is initially set to 7 and decremented every time the message is forwarded. As soon as the TTL counter reaches 0, the message is discarded and not forwarded any longer.

There are two disadvantages resulting from a flooding-based search algorithm. If the system size is large or the system is widely ramified, it will not be guaranteed that a Query message reaches all peers in the system. For example, an initial TTL counter of 3 for Query messages in the Gnutella system from Figure 2.2 would result in not reaching peer $\mathcal{P}_1$. Thus, the resolution of a query, i.e. the exact information about all peers in the system that share relevant content, can never be guaranteed. This includes the inability to provide a negative query answer, i.e. the information that no peer is currently online sharing relevant content. In contrast, only positive answers in form of Query Hit messages are possible. An even more problematic disadvantage is the high amount of signaling traffic that is generated by a flooding-based search algorithm. In Section 2.3.2, we deepen this finding and present some state-of-the-art techniques that can be used to reduce the signaling traffic.

Appropriate routing of incoming messages. Due to the decentralized nature of unstructured P2P systems, every participating peer needs to cooperatively route messages that are initiated by other peers. Above, we have already seen how this is done in Gnutella 0.4. Every peer forwards incoming Ping and Query messages to all of its neighbors except the one that has delivered the message. Furthermore, Pong and Query Hit messages are transmitted along the same path and contain the same unique identifier as the associated Ping and Query messages.

2.3.2 The Problem of Flooding

The high signaling traffic generated in unstructured P2P systems is caused by the so-called snowball effect. Every query starts with an initial Query message that is continuously copied and forwarded by every peer that receives the message. The total number of generated messages depends on the connectivity d, i.e. the average number of active connections that a peer maintains to other peers, and the initial TTL counter. For the following analysis (see also [SS02]), we model the unstructured P2P system as a tree, i.e. we assume that no loops occur, and we assume the system size to be sufficiently large so that a query does not end before the TTL counter reaches 0.

The initial transmission of the Query message results in d messages that are sent from the initiating peer to its d neighbors. In a second step, these d neighbors forward the message to their $d-1$ neighbors, excluding the initiating peer. Thus, $d \cdot (d-1)$ messages

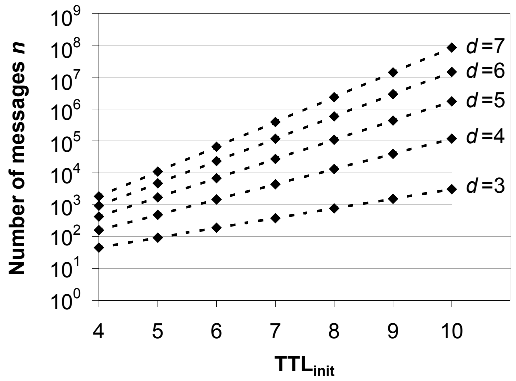


Figure 2.3: Number of messages generated by a single query.

are generated in step 2. In general, $d \cdot (d-1)^{\mathrm{i}-1}$ messages are generated in the i^{th} step, so the total number of messages n generated by a single query is given by

$$n = \sum_{\mathrm{i}=1}^{\mathrm{TTL_{init}}} d \cdot (d-1)^{\mathrm{i}-1} = d \cdot \frac{1-(d-1)^{\mathrm{TTL_{init}}}}{2-d} \tag{2.1}$$

Figure 2.3 shows n as a function of d and $\mathrm{TTL_{init}}$. For example, typical Gnutella 0.4 settings of $d = 4$ and $\mathrm{TTL_{init}} = 7$ result in 4,372 messages generated by every single query. In particular, we notice that n grows exponentially with an increasing initial TTL counter. For the designer of an unstructured P2P system this represents a trade-off between query success and signaling traffic. If he specifies a low initial TTL counter, the signaling traffic will be comparatively low, but the query might end before it reaches all peers in the system. On the other hand, if he specifies a high initial TTL counter, the query will comprise (ideally) all participating peers, but will thereby generate a high amount of signaling traffic.

In the above analysis, we have made the simplistic assumption that no loops occur in the P2P system. Clearly, this assumption does not hold in a real-world system like Gnutella 0.4. In fact, loops in an unstructured P2P system diminish the exponential growth of the signaling traffic. However, it still remains very high. For example, measurements of the Gnutella system by Ripeanu *et al.* in 2001 have shown that "the network generates about 330 TB/month simply to remain connected and to broadcast user queries" [RIF02].

Multiple solutions can be found in the literature (e.g. [G06, LCC$^+$02, SBR04, SMZ03, YGM02]) that aim at reducing the high amount of signaling traffic in unstructured P2P systems. In the following, we briefly present the most important ones, including the setup of a hierarchical system architecture, the expanding ring search and random walks. Structured P2P systems, as another possibility to avoid flooding and the corresponding high signaling traffic, are discussed in detail in Section 2.4.

Hierarchical system architecture. Shortly after the start of Gnutella 0.4, the increasing number of users lead to a strong increase in the signaling traffic. As a consequence, the Gnutella developers redesigned the system. The main improvement of the next Gnutella version, Gnutella 0.6, was to establish a hierarchical system architecture that divides the participating peers into so-called “ultrapeers” or “superpeers” and so-called “leafnodes”. The hierarchical system architecture does not consider all peers as equal, but promotes powerful peers to superpeers that set up an unstructured P2P system just as Gnutella 0.4 and that are responsible for routing all Query messages. All other peers, in contrast, act as leafnodes that maintain only a connection to their (randomly selected) superpeer and are not involved in routing Query messages. After its release, Gnutella 0.6 lead to a greatly improved efficiency and scalability of the Gnutella system in terms of reduced signaling traffic and higher query success. Nevertheless, Gnutella 0.6 still uses a flooding-based search algorithm to distribute Query messages among superpeers.

Also the FastTrack protocol establishes a hierarchical unstructured system architecture, similar to that of Gnutella 0.6. Popular applications that are based on this P2P protocol are the file-sharing application KaZaA [KZA] and the VoIP application Skype [SKY, BS06, BD].

Later in this thesis, we come back in detail to the idea of a hierarchical system architecture when we develop efficient solutions for P2P systems that are applied in heterogeneous communication environments.

Expanding ring search. A further solution to reduce the high signaling traffic in unstructured P2P systems is the “expanding ring search”, proposed in [LCC+02]. The authors try to solve the conflict between a high query success and a high signaling traffic (cf. the above discussion of setting the initial TTL counter) by starting a query with an initially low TTL counter. The initiating peer then waits for corresponding Query Hit messages. If the searched data item is highly replicated in the system, i.e. a lot of peers share the data item, it is likely that the query will already reach enough providing peers even when the initial TTL counter is low. If this is not the case, the initiating peer will increase the initial TTL counter and restart another query. This procedure continues until the searched data item is found or the initial TTL counter reaches an upper bound.

The expanding ring search certainly performs well when the searched data item is highly replicated. However, queries, especially for rare data items, take longer with this approach, because they may have to be started multiple times by the initiating peer.

Random walks. In [Spi76], Spitzer formally defines a random walk as a transition function $P(x, y)$ that, for all pairs x and y in a d-dimensional space of lattice points R, possesses the property

$$0 \leq P(x, y) = P(0, y - x), \quad \sum_{\mathrm{x \in R}} P(0, x) = 1 \tag{2.2}$$

Lv *et al.* exploit this technique in unstructured P2P systems to significantly reduce the signaling traffic [LCC+02]. Instead of flooding, they propose to initiate a query by sending a Query message to n selected neighbors of the initiating peer. Henceforth, every Query message makes its random walk through the system. Every peer that receives a Query message forwards it to only one randomly selected neighbor. To achieve a satisfying query success, i.e. to ensure that the query reaches as many peers in the system as possible, the initial TTL counter of a random walk must be significantly higher than when flooding the query through the system.

The simulations in [LCC+02] show that the random walk method cuts down the signaling traffic significantly compared to a flooding-based search. However, this traffic reduction is clearly bought at the expense of an increased query latency, as random walks may need significantly more time to discover peers that share relevant data items.

Although the above solutions can reduce the high amount of generated signaling traffic, another weakness of unstructured P2P systems still remains: the inability to guarantee the resolution of a query. Unstructured P2P systems cannot guarantee to find all peers in the system that share relevant content. Instead, only positive answers in form of Query Hit messages are possible. To solve this problem, structured P2P systems have been proposed.

2.4 Structured P2P Systems

Structured P2P systems are the latest generation of P2P systems. In contrast to unstructured P2P systems, they establish a predefined topology among the peers, e.g. a ring, a tree, or a multi-dimensional hypercube. This structure is used to distribute index information about all shared data items in the system over all participating peers. Usually, this is done by means of a Distributed Hash Table (DHT).

2.4.1 The Concept of Distributed Hash Tables

The basic problem that a DHT addresses is a self-organized distribution of a set of data items among a set of peers, enabling their subsequent fast lookup. In a DHT, peers collaboratively manage specific subsets of data items, identified by keys from an identifier space I, which depend on the set of all peers and the set of all data items available in the system. This is done by assigning each peer a unique identifier (ID) taken from I and by associating with this ID a partition (in some DHTs multiple partitions) of the identifier space I, so that the peer becomes responsible to manage all data items identified by keys from the associated partition. The partition typically consists of all keys that are closest to the peer's ID in a suitable metric. Thus, the identifier space I is equipped with a distance function s. To forward lookup requests, peers set up a routing graph by taking into account the knowledge on the association of peers with partitions.

Following [AAG+05], we define four constitutive functions that any DHT is equipped with:

- a function $id : P \rightarrow I$ that assigns an identifier taken from the identifier space I to each peer in P, where P is the set of all peers in the system (we say the identifier of a peer is the peer's "ID"),
- a function $key : D \rightarrow I$ that assigns an identifier taken from the identifier space I to each data item in D, where D is the set of all shared data items in the system (we say the identifier of a data item is the data item's "key"),

- a function *partition* : $P \rightarrow 2^I$ that associates each peer in P with one or more partitions of the identifier space I, and
- a function *neighbors* : $P \rightarrow 2^P$ that associates each peer in P with a subset of other peers in P, thus building a directed graph.

The function *id* is a collision-resistant hash function (e.g. SHA-1 [Pub02] or MD5 [Riv92]) that maps a peer onto a hash value. A typical input parameter of function *id* is a peer's IP address or a randomly chosen string. A side goal of the hash function *id* is balancing the load distribution: All peers should be distributed equally over the identifier space, so that each peer is responsible for approximately the same number of keys. The function *key* is also a collision-resistant hash function (possibly the same as function *id*) that maps a shared data item onto a hash value. A typical input parameter of function *key* is a data item's name or a keyword describing its content. The set of all participating peers can at any time be considered as input parameter of the function *partition*. The interpretation is that the complete identifier space must be partitioned among all peers that are currently present in the system. The function *neighbors* is responsible for building the DHT's routing graph. Using the metric of the identifier space, it normally enables peers to maintain short-range links to all peers with neighboring IDs and, in addition, a small number of long-range links to some selected peers. Using this established routing graph, peers forward lookup requests in a directed manner to other peers from their routing tables, trying to greedily reduce the distance to the key that is being looked up. As a result, most DHTs achieve a lookup path length, i.e. a number of peers contacted during a lookup, that is logarithmic[1] in the number of participating peers N, by using routing tables whose size is also logarithmic in N. However, in recent years there have been also some works, e.g. [MNR02] and [KK03], that achieve a constant outdegree graph and thus constant sized routing tables while retaining a logarithmic lookup path length.

To sum up, the specific design of a DHT depends on the choice of the identifier space, the distance function, the partitioning of the identifier space, and the linking strategy. They have been a subject of intensive research over the recent years and resulted in numerous designs of structured P2P systems (cf. Sections 2.4.2 to 2.4.4).

System maintenance. The good properties related to the efficiency of routing lookup requests do not come for free. For constructing and maintaining a DHT-based P2P system, peers have to deal in particular with the problem of churn, i.e. continuously arriving and departing peers. Since the freedom to choose neighbors in a DHT is constrained by the conditions imposed by the function *neighbors*, maintenance algorithms are required to ensure the consistency of routing tables in the presence of churn. The same applies for the consistent association of all partitions of the identifier space (and thus of the shared data items) with the responsible peers according to the function *partition*, i.e. the correctness of the DHT's mapping rule.

Depending on the type of guarantees given by the DHT-based P2P system, different deterministic and probabilistic maintenance algorithms have been developed. Maintenance actions can be triggered by various events, such as peer arrivals and departures or

[1] Note that, throughout this thesis, the term "logarithmic" refers to the binary logarithm to base 2, and that the used notation $\log x$ represents the binary logarithm of x, i.e. $\log_2 x$.

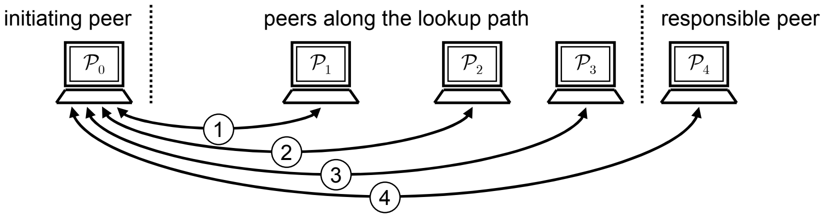


Figure 2.4: Iterative routing in a DHT-based P2P system.

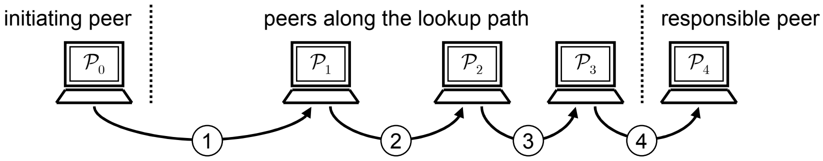


Figure 2.5: Recursive routing in a DHT-based P2P system.

routing failures due to inconsistent routing tables, or being run periodically. In general, any maintenance algorithm faces a trade-off between signaling traffic versus degree of consistency and thus resilience and reliability of the system.

Iterative and recursive routing. There are two different possibilities of resolving a lookup of the responsible peer for key k in a DHT-based P2P system: an iterative routing scheme and a recursive routing scheme. With an iterative scheme, the peer that initiates the lookup also initiates the complete subsequent communication for resolving the lookup, by continuously contacting peers along the lookup path and asking them for the peer in their routing table that is closest to k. With a recursive scheme, the peer that initiates the lookup sends the lookup request to the peer in its routing table that is closest to k. Henceforth, each intermediate peer along the lookup path forwards the request in the same way, until it arrives at the responsible peer. Figures 2.4 and 2.5 depict both routing schemes.

At first glance a recursive routing scheme seems to be preferable, as it is faster (peers can immediately forward the lookup request to the next peer along the lookup path) and generates less signaling traffic (only one message is generated per hop). There is a further advantage of a recursive routing scheme. As peers may fail, each message should be monitored by a timer that will initiate a retransmission or another appropriate failure recovery mechanism if no acknowledgement is received. When the DHT uses a recursive routing scheme, each peer communicates mostly with the peers in its routing table, so it can easily maintain a past history of its neighbors' response times and thus calculate a reasonable timer value for each neighbor. This is hardly possible with an iterative routing scheme, where a peer communicates potentially with any other peer in the system [RGRK04].

However, there are also important advantages offered by an iterative routing scheme. As the peer that initiates the lookup controls the complete communication during the lookup, it can immediately react to peer failures and, e.g., contact another peer in the proximity of the failed peer. In contrast, if a recursive lookup does not provide a response, the initiating peer will have no information about what went wrong and will usually have to restart the whole lookup procedure from the beginning [DLS$^+$04]. Iterative routing also offers the possibility of following multiple different lookup paths in parallel, thus preventing the lookup procedure from being delayed by a single timeout [RGRK04]. Furthermore, with regard to system security aspects, an iterative routing scheme is also preferable in the sense of a lower vulnerability to attacks. In a DHT-based P2P system that applies a recursive routing scheme, an attacker can trigger multiple subsequent messages with every initiated lookup request. The attacker can exploit this possibility by generating a huge amount of lookup traffic, e.g. to perform a DoS attack. This is not possible with an iterative routing scheme. Here, the initiating peer can trigger only one subsequent message per routing hop.

In recent years, the advantages and disadvantages of iterative and recursive routing have been controversially discussed in the research community. In general, the designer of a DHT-based P2P system has to decide between a better performance (recursive routing) and a higher reliability (iterative routing). For example, Microsoft has chosen the latter option and uses an iterative routing scheme in its Peer Name Resolution Protocol (PNRP). PNRP is part of the P2P platform that is shipped with Windows® XP and Windows Vista™ [Mic]. In this thesis, we adopt this view and concentrate on iterative routing. However, where appropriate, we also discuss recursive routing.

Data storage. The function *partition* of a DHT associates each peer in the system with one or more partitions of the identifier space. As a result, each peer is responsible for a subset of all shared data items in the system. Responsibility in this context means that each peer either stores the associated data items by copying them from the providing peers ("direct storage"), or maintains only a reference to each associated data item ("indirect storage"). A reference to a shared data item contains a keyword (usually the filename), a key (the hash value of the keyword), and contact information about the providing peer (usually its IP address and port number). According to Wehrle *et al.* [WGR05], both possibilities have advantages and disadvantages.

In a DHT that uses direct storage, a shared data item is copied to and permanently stored on the responsible peer. When the data item falls into the responsibility of another peer, e.g. upon a peer arrival, the data is copied from the previously responsible peer to the new peer. Consequently, the data item is permanently available in the P2P system, even when the providing peer leaves the system. However, the permanent availability of data items does not come for free. It certainly generates an immense amount of overhead in terms of storage space consumption and bandwidth consumption. This is especially true when large data items such as movies or CD images are shared.

With regard to this overhead we abandon from a direct storage and focus on an indirect storage of data items. Each peer stores only a reference to each data item it is responsible for, and the data item itself remains on the providing peer. Although with this possibility a shared data item is available only as long as the providing peer is available, the storage and traffic load in the DHT is significantly lower.

Note that in the following chapters, we often refer to a reference to a shared data item also as a "key-value-pair" that is stored in the DHT. The term "key" here corresponds to the keyword and the associated hash value, respectively, and the term “value” reflects the contact information of the providing peer.

Numerous different DHT-based P2P systems have been designed in recent years. Basically, they all rely on the DHT concept that we have just presented. Nevertheless, they show a large diversity in the key design aspects, i.e. choice of the identifier space, distance function, partitioning of the identifier space, and linking strategy. In the following, we discuss in detail the two DHT-based P2P systems that, from our point of view, have drawn the most attention in the research community: Chord (Section 2.4.2) and Kademlia (Section 2.4.3). In Section 2.4.4, we briefly present further important DHT-based P2P systems. Section 2.4.5 summarizes with a comparison of the presented DHTs.

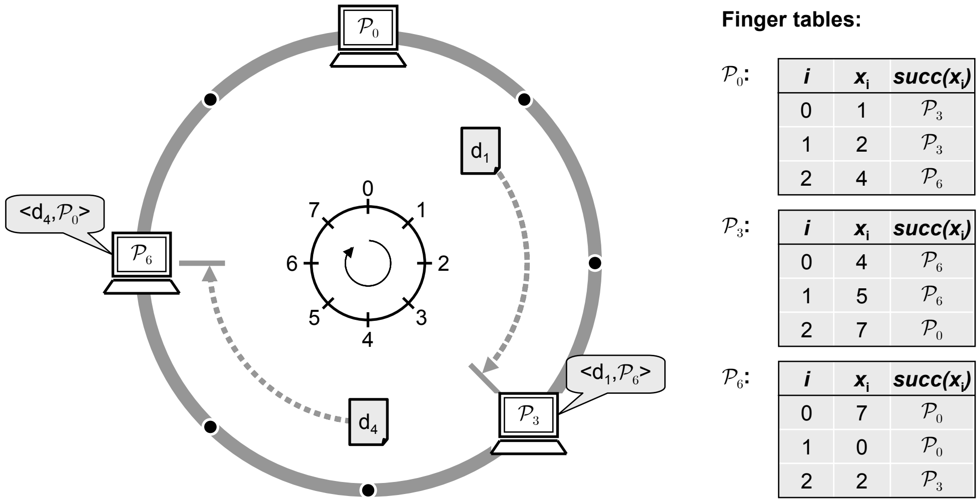
2.4.2 Chord

The Chord protocol, proposed by Stoica *et al.* in [SMK+01], uses a b-bit hash function to derive an ID for every peer and a key for every shared data item in the system. The identifier space is ring-shaped, ranging from 0 to $2^{\mathrm{b}} - 1$. Figure 2.6 shows an exemplary Chord ring. We set $b = 3$ and put three peers on the ring: peer $\mathcal{P}_0$ with ID 0, peer $\mathcal{P}_3$ with ID 3, and peer $\mathcal{P}_6$ with ID 6.

The distance function in Chord is simple and intuitive. The distance $s(x, y)$ from an identifier x to an identifier y is

$$0 \leq s(x,y) = \begin{cases} y - x & \text{if } y \geq x \\ y - x + 2^{\mathrm{b}} & \text{if } y < x \end{cases} \tag{2.3}$$

or, in other words, the absolute distance from x to y in clockwise direction. This distance function is directly used to associate data items with peers. A data item with key k is mapped onto the peer whose ID has the smallest distance to k, i.e. the first peer on the



Finger tables:

$\mathcal{P}_0$:

i	x_i	$succ(x_i)$
0	1	$\mathcal{P}_3$
1	2	$\mathcal{P}_3$
2	4	$\mathcal{P}_6$

$\mathcal{P}_3$:

i	x_i	$succ(x_i)$
0	4	$\mathcal{P}_6$
1	5	$\mathcal{P}_6$
2	7	$\mathcal{P}_0$

$\mathcal{P}_6$:

i	x_i	$succ(x_i)$
0	7	$\mathcal{P}_0$
1	0	$\mathcal{P}_0$
2	2	$\mathcal{P}_3$

Figure 2.6: Exemplary Chord ring with 3 peers: Peer $\mathcal{P}_0$ provides data item d$_4$, peer $\mathcal{P}_6$ provides data item d$_1$. The according references are stored on the responsible peers $\mathcal{P}_6$ and $\mathcal{P}_3$, respectively.

Chord ring clockwise from k. This peer is called the “successor” of k. In the example from Figure 2.6, we assume that two data items are available in the system. Peer $\mathcal{P}_0$ provides data item d_4 whose key is 4, and peer $\mathcal{P}_6$ provides data item d_1 whose key is 1. According to Chord’s mapping rule, d_4 is mapped onto the successor of identifier 4, i.e. peer $\mathcal{P}_6$. The same applies for d_1, whose successor is peer $\mathcal{P}_3$. As a result, $\mathcal{P}_6$ stores a key-value-pair $\langle \mathrm{d}_4, \mathcal{P}_0 \rangle$ that refers to $\mathcal{P}_0$ as provider of d_4, and $\mathcal{P}_3$ stores a key-value-pair $\langle \mathrm{d}_1, \mathcal{P}_6 \rangle$ that refers to $\mathcal{P}_6$ as provider of d_1.

Chord’s mapping rule, which assigns each shared data item to its direct successor on the ring, also implies the partitioning of the identifier space. The section of the identifier space that peer $\mathcal{P}$ is responsible for ranges from the ID of $\mathcal{P}$’s predecessor $pred(\mathcal{P})$ [2] on the ring to $\mathcal{P}$’s ID:

$$partition(\mathcal{P}) =]id(pred(\mathcal{P})); id(\mathcal{P})] \tag{2.4}$$

In our exemplary Chord ring, peer $\mathcal{P}_0$ is responsible for the interval $[7; 0]$, peer $\mathcal{P}_3$ is responsible for the interval $[1; 3]$, and peer $\mathcal{P}_6$ is responsible for the interval $[4; 6]$.

Routing. To route a lookup request for key k to the responsible peer, i.e. to the successor of k on the ring, each peer maintains a pointer to its succeeding peer on the ring. As long as these pointers are correct (we will see shortly how this can be achieved in case of churn) a lookup request can be forwarded from each peer to its successor, until the lookup request reaches the responsible peer which then can answer the request. Certainly, this routing scheme is not scalable, as it requires $1/2 N$ hops on average to resolve the successor of k, where N is the number of peers in the system. Thus, each peer maintains a routing table, called “finger table” in Chord, with up to b additional pointers to other peers in the system. Note that $N \leq 2^{\mathrm{b}}$, so the size of the finger table is logarithmic in N.

The finger table of peer $\mathcal{P}$ is constructed as follows. Each entry, usually called “finger”, consists of the finger index i, the identifier x_{i} of a predefined distance from $\mathcal{P}$’s ID on the identifier ring (= the finger ID), and the responsible peer for this identifier, i.e. the successor of x_{i} $succ(x_{\mathrm{i}})$ (= the finger peer). For the i^{th} finger of $\mathcal{P}$, the finger ID x_{i} is calculated by

$$x_{\mathrm{i}} = \left(id(\mathcal{P}) + 2^{\mathrm{i}}\right) \text{ modulo } 2^{\mathrm{b}} \quad \forall\, i \in [0; b-1] \tag{2.5}$$

The right side of Figure 2.6 lists the finger tables of the three peers in our exemplary Chord ring. Note that the first finger peer in the finger table is always the direct successor on the ring.

The distances that fingers bypass on the Chord ring increase exponentially with the finger index. Thus, each peer is aware of multiple peers that are closely following it on the identifier ring and knows a few peers that are farther away. The maximum distance that can be bypassed with the $(b-1)^{\mathrm{th}}$ finger is $2^{\mathrm{b}-1}$ which is half of the identifier space. As a result, the number of hops to resolve the successor of a given key k is reduced to $1/2 \log N$ on average (see [SMK+01] for proof).

Chord makes an explicit differentiation between the reliability of lookups and their duration. As long as each peer has correct information about its succeeding peer on

[2] Note that we use the symbols $pred()$ (= predecessor) and $succ()$ (= successor) synonymously for peers and identifiers, as the meaning will be clear from the context. In either case, the symbols refer to the preceding and succeeding peer on the ring, respectively.

the ring, a lookup will always reach the responsible peer, even if the finger tables have missing or incorrect entries. In this case, the lookup only takes longer. This differentiation is reflected in two different algorithms to maintain neighbor information in Chord, the Stabilize algorithm for ensuring the correctness of successor pointers, and the FixFingers algorithm for updating fingers in the finger table of peers (see below for details). Also the routing of a lookup request reflects this differentiation: In a first step, the lookup request is routed to the immediate predecessor $pred(k)$ of the desired key k. Provided that this peer has correct information about its successor, $succ(pred(k))$ is the responsible peer for k, so $pred(k)$ sends this information back to the searching peer. As an example, we assume that peer $\mathcal{P}_6$ in Figure 2.6 initiates a lookup to find the responsible peer for its data item d_1, in order to store the according reference on this peer (we also say that $\mathcal{P}_6$ "inserts" its data item d_1 into the system). $\mathcal{P}_6$ first checks its finger with the highest index, i.e. finger 2, pointing to $\mathcal{P}_3$. Because $\mathcal{P}_3$ does not precede 1 (the key of d_1), $\mathcal{P}_6$ takes one step back and checks the next finger in its finger table, i.e. finger 1. This finger points to $\mathcal{P}_0$, which does precede 1. Therefore, the lookup request is sent to $\mathcal{P}_0$. In turn, $\mathcal{P}_0$ finds that key 1 is located between itself and its successor $\mathcal{P}_3$, so $\mathcal{P}_3$ is the responsible peer. This information is sent back to $\mathcal{P}_6$, so that the insertion of d_1 can take place.

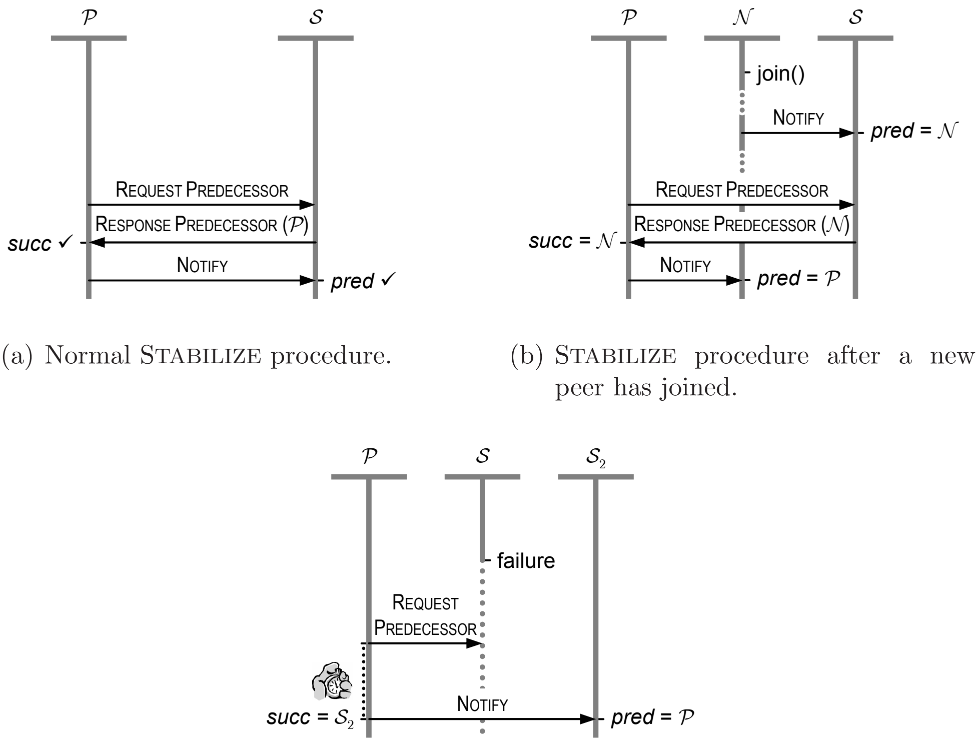
Joining the Chord DHT. When connecting to the system, a peer sets up its successor pointer and its finger table entries, and takes over all references to shared data items that fall into its responsibility.

To set up its successor pointer, a joining peer $\mathcal{P}$ contacts another peer that is already participating in the Chord DHT and asks this peer to look up the successor of $\mathcal{P}$ $succ(\mathcal{P})$. After receiving the response, $\mathcal{P}$ sets $succ(\mathcal{P})$ as its successor and notifies $succ(\mathcal{P})$ about its presence. As a result, $succ(\mathcal{P})$ changes its predecessor pointer to $\mathcal{P}$. Below, we will see that this predecessor pointer is a necessary prerequisite for the Stabilize algorithm. Once the successor pointer of $\mathcal{P}$ is set, $\mathcal{P}$ populates its finger table in the same way, i.e. by asking the other peer to look up the responsible finger peer for each of $\mathcal{P}$'s finger IDs. Finally, $\mathcal{P}$ contacts its successor in order to transfer from it all references whose keys lie in $\mathcal{P}$'s responsibility now.

Maintaining correct routing information. To ensure a correct and efficient routing of lookup requests in the presence of joining and leaving peers, the successor pointers and finger table entries of each peer in a Chord DHT are updated periodically. The according algorithms are called Stabilize and FixFingers.

The Stabilize algorithm is used to periodically probe a peer's successor on the ring. As already mentioned, a prerequisite of this algorithm is that each peer maintains an additional pointer to its predecessor on the ring. This pointer is also updated during a Stabilize procedure. Peer $\mathcal{P}$ initiates a Stabilize procedure by sending a Request Predecessor message to its successor $\mathcal{S}$. In the following, three situations can occur:

1. $\mathcal{S}$ still is the correct successor of $\mathcal{P}$ on the ring.
2. A new peer $\mathcal{N}$ has joined the Chord DHT and is located between $\mathcal{P}$ and $\mathcal{S}$, thus being the new successor of $\mathcal{P}$.
3. $\mathcal{S}$ has failed so that $\mathcal{P}$ has an invalid successor pointer.



(a) Normal STABILIZE procedure.

(b) STABILIZE procedure after a new peer has joined.

(c) STABILIZE procedure after a peer failure.

Figure 2.7: STABILIZE algorithm.

In case 1 (cf. Figure 2.7(a)), $\mathcal{S}$ replies with a RESPONSE PREDECESSOR message verifying $\mathcal{P}$ as predecessor of $\mathcal{S}$. Hence, peer $\mathcal{P}$ finds that $\mathcal{S}$ still is its correct successor, and sends a NOTIFY message to $\mathcal{S}$ that confirms that $\mathcal{P}$ still is the correct predecessor of $\mathcal{S}$. This is the normal mode of operation. In case 2 (cf. Figure 2.7(b)), $\mathcal{S}$ has already set its predecessor pointer to the new peer $\mathcal{N}$ during the join procedure of $\mathcal{N}$. Thus, the RESPONSE PREDECESSOR message contains $\mathcal{N}$ as predecessor of $\mathcal{S}$, so $\mathcal{P}$ learns about $\mathcal{N}$'s presence and sets its successor pointer to $\mathcal{N}$. Then, it sends a NOTIFY message to $\mathcal{N}$, and $\mathcal{N}$ sets its predecessor pointer to $\mathcal{P}$, thus ensuring the consistency of all predecessor and successor pointers of $\mathcal{P}$, $\mathcal{N}$ and $\mathcal{S}$. In case 3 (cf. Figure 2.7(c)), $\mathcal{S}$ has failed without notifying its neighboring peers on the ring. Thus, $\mathcal{P}$ has an outdated successor pointer and the Chord ring is broken between $\mathcal{P}$ and its new successor $\mathcal{S}_2$, which still considers $\mathcal{S}$ as its predecessor. To be able to handle such peer failures, each peer maintains additional pointers to its n closest successors on the Chord ring. These pointers are called the "successor list". If the direct successor of $\mathcal{P}$ fails, $\mathcal{P}$ will not receive a RESPONSE PREDECESSOR message from $\mathcal{S}$, so a timeout will occur. Hereupon, $\mathcal{P}$ notices the failure of $\mathcal{S}$ and sets the next entry in its successor list, i.e. $\mathcal{S}_2$, as its new successor. Additionally, $\mathcal{P}$ sends a NOTIFY message to $\mathcal{S}_2$, $\mathcal{S}_2$ sets its predecessor pointer to $\mathcal{P}$, and the ring break is repaired. Note that [SMK$^+$01] proposes a successor list length of $n = O(\log N)$.

Although the ring structure can be re-established after a peer failure by the Stabilize algorithm, all references that have been stored on the failed peer are lost. Hence, the corresponding data items cannot be looked up any more. This problem is usually solved by replication and republishing strategies. More details on these strategies are presented in the following chapters.

As long as the Stabilize algorithm ensures correct successor pointers at each peer, a lookup request always reaches the responsible peer. In case of missing or incorrect finger table entries, any peer along the lookup path can fall back on its successor pointer to route a lookup request closer to its destination. However, to answer lookup requests as fast as possible in the presence of churn, also the finger table entries must be updated regularly. For this reason, each peer implements the FixFingers algorithm. This algorithm periodically initiates a lookup of the currently responsible finger peer for each finger ID, and modifies the finger table entry if necessary. Thus, the FixFingers algorithm ensures that peer arrivals and departures are reflected in each peer's finger table.

Leaving the Chord DHT. The Stabilize and FixFingers algorithms continuously update the information about neighboring peers, thereby detecting peers that have failed and are not available any more. As a result, Chord is resistant to peer failures to a certain extent, and every peer that leaves the Chord DHT could simply disconnect without notifying any other peer. However, to increase the system's stability and performance, it is reasonable to implement the following leave procedure. Each peer that leaves the system informs its predecessor and successor about its departure (to avoid a temporary ring break), and transfers the references it stores to its successor on the ring (to avoid the unavailability of shared data items).

Summary. Chord is a DHT that defines a ring-shaped identifier space ranging from 0 to $2^{\mathrm{b}} - 1$. Each data item is associated with the peer whose ID succeeds the data item's key on the identifier ring. In order to route lookup requests towards their destination, each peer maintains a successor pointer and a routing table of size $O(\log N)$. The routing table is called finger table and contains peers in exponentially increasing distance on the identifier ring, thus ensuring that lookups can be answered after $O(\log N)$ routing hops.

2.4.3 Kademlia

In [MM02], Maymounkov and Mazières present a DHT called Kademlia. Like Chord, Kademlia assigns a b-bit hash value to every peer and every shared data item in the system. Given the bit-by-bit representation of identifiers, the identifier space in Kademlia can be depicted as binary tree. Figure 2.8 shows an exemplary Kademlia tree. We set $b = 3$ and put three peers in the system: peer $\mathcal{P}_1$ with ID 001, peer $\mathcal{P}_3$ with ID 011, and peer $\mathcal{P}_6$ with ID 110.

Kademlia defines the distance $s(x, y)$ between two identifiers x and y as their bitwise exclusive or (XOR) operation:

$$0 \leq s(x, y) = x \oplus y \tag{2.6}$$

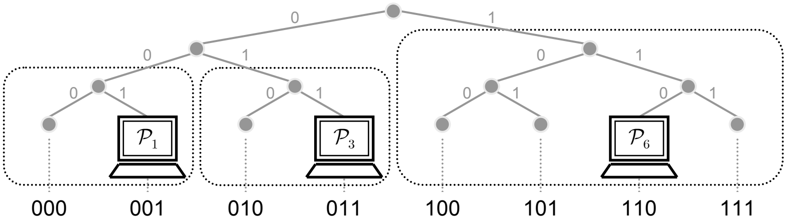


Figure 2.8: Exemplary Kademlia tree with 3 peers.

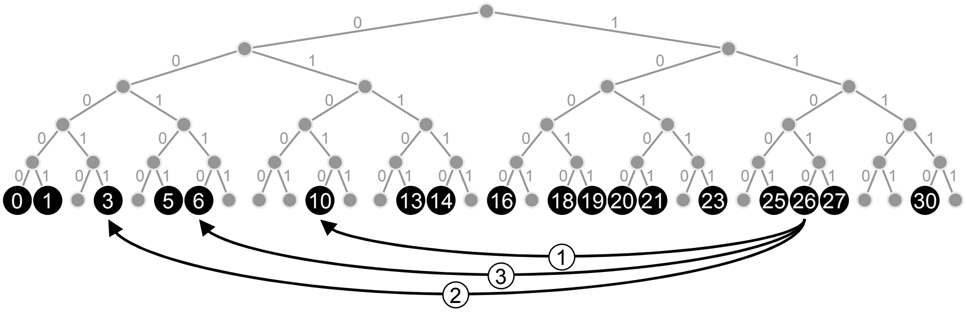
To associate data items with peers, Kademlia follows Chord's approach by mapping a data item with key k onto the peer whose ID has the smallest distance to k. As an example, we assume that data item d_4 with key 100 is inserted into the Kademlia system from Figure 2.8. According to the XOR metric, the responsible peer that stores the corresponding reference is $\mathcal{P}_6$, because

$$\begin{aligned} id(\mathcal{P}_1) \oplus key(\mathrm{d}_4) &= 001 \oplus 100 = 101 = 5_{10} \\ id(\mathcal{P}_3) \oplus key(\mathrm{d}_4) &= 011 \oplus 100 = 111 = 7_{10} \\ id(\mathcal{P}_6) \oplus key(\mathrm{d}_4) &= 110 \oplus 100 = 010 = 2_{10} \end{aligned}$$

Kademlia's mapping rule becomes more intuitive if we take a look on the partitioning of the identifier space that results from the XOR distance function. To determine the section of the identifier space that each peer is responsible for, the binary tree is divided into lower subtrees so that each subtree contains exactly one peer. This peer clearly is the responsible peer for that subtree, and thus for the corresponding section of the identifier space. In our exemplary Kademlia tree, peer $\mathcal{P}_1$ is responsible for the subtree $00*$ and thus for the interval $[000; 001]$, peer $\mathcal{P}_3$ is responsible for the subtree $01*$ and thus for the interval $[010; 011]$, and peer $\mathcal{P}_6$ is responsible for the subtree $1{*}{*}$ and thus for the interval $[100; 111]$. Now we immediately see that d_4 falls into the responsibility of $\mathcal{P}_6$.

Routing. To route a lookup request for key k to the responsible peer, i.e. the peer whose ID is closest to k according to the XOR metric, each peer maintains a routing table that is constructed as follows. For every section of the identifier space with distance $s_\mathrm{i} \in \left[2^\mathrm{i}; 2^{\mathrm{i}+1}\right[\ \forall\, i \in [0; b-1]$ from peer $\mathcal{P}$, i.e. for all subtrees that do not contain $\mathcal{P}$, peer $\mathcal{P}$ maintains a list with up to k peers in this subtree (typically $k = 20$). Each such list is called a "k-bucket". Figure 2.9 illustrates the k-buckets of the peers $\mathcal{P}_3$, $\mathcal{P}_6$, $\mathcal{P}_{10}$ and $\mathcal{P}_{26}$ in a Kademlia system with 18 peers where $b = 5$ and $k = 3$.

Like in Chord, the size of routing tables is $O(\log N)$, and the exponentially increasing distance s_i to other peers in the routing table results in a number of $O(\log N)$ routing hops to find the responsible peer in the DHT. A lookup request in Kademlia is routed as follows. We assume that peer $\mathcal{P}_{26}$ in Figure 2.9 initiates a lookup to find the responsible peer for key 00111. From its routing table, it selects the closest peer it knows in subtree $0{*}{*}{*}{*}$, which is $\mathcal{P}_{10}$. $\mathcal{P}_{10}$ in turn returns $\mathcal{P}_3$ as the closest peer it knows in subtree $00{*}{*}{*}$. In the next routing step, $\mathcal{P}_{26}$ contacts $\mathcal{P}_3$, and $\mathcal{P}_3$ returns $\mathcal{P}_6$ as the closest peer it knows in subtree $001{*}{*}$. Finally, when the lookup request reaches $\mathcal{P}_6$, $\mathcal{P}_6$ finds that it is the



Routing tables:

		0	1	2	3	4
	i	0	1	2	3	4
	***s*ᵢ**	1	2…3	4…7	8…15	16…31
$\mathcal{P}_3$:	**subtree**	00010	0000∗	001∗∗	01∗∗∗	1∗∗∗∗
	***k*-bucket**	–	$\mathcal{P}_0\,\mathcal{P}_1$	$\mathcal{P}_5\,\mathcal{P}_6$	$\mathcal{P}_{10}\,\mathcal{P}_{13}\,\mathcal{P}_{14}$	$\mathcal{P}_{16}\,\mathcal{P}_{19}\,\mathcal{P}_{27}$
$\mathcal{P}_6$:	**subtree**	00111	0010∗	000∗∗	01∗∗∗	1∗∗∗∗
	***k*-bucket**	–	$\mathcal{P}_5$	$\mathcal{P}_0\,\mathcal{P}_1\,\mathcal{P}_3$	$\mathcal{P}_{10}\,\mathcal{P}_{13}\,\mathcal{P}_{14}$	$\mathcal{P}_{20}\,\mathcal{P}_{21}\,\mathcal{P}_{30}$
$\mathcal{P}_{10}$:	**subtree**	01011	0100∗	011∗∗	00∗∗∗	1∗∗∗∗
	***k*-bucket**	–	–	$\mathcal{P}_{13}\,\mathcal{P}_{14}$	$\mathcal{P}_0\,\mathcal{P}_1\,\mathcal{P}_3$	$\mathcal{P}_{16}\,\mathcal{P}_{20}\,\mathcal{P}_{26}$
$\mathcal{P}_{26}$:	**subtree**	11011	1100∗	111∗∗	10∗∗∗	0∗∗∗∗
	***k*-bucket**	$\mathcal{P}_{27}$	$\mathcal{P}_{25}$	$\mathcal{P}_{30}$	$\mathcal{P}_{16}\,\mathcal{P}_{19}\,\mathcal{P}_{23}$	$\mathcal{P}_{10}\,\mathcal{P}_{13}\,\mathcal{P}_{14}$

Figure 2.9: Routing in a Kademlia DHT.

only peer located in subtree 0011∗, so it is the responsible peer for key 00111. Thus, the lookup request can be answered by $\mathcal{P}_6$.

In their paper, Maymounkov and Mazières introduce a parallel routing of lookup requests. They suggest that the initiating peer contacts x peers in parallel in each routing step, and that each contacted peer returns its k closest peers to the destination. For purpose of clarity, we set $x = 1$ in our example and let each contacted peer return only its closest peer to the destination. We refer to [MM02] for more details on parallel lookup routing.

Joining the Kademlia DHT. When connecting to the system, a joining peer $\mathcal{P}$ has to know another peer that is already participating in the Kademlia DHT and inserts this peer into its appropriate k-bucket. Then, $\mathcal{P}$ initiates a lookup of the responsible peer for its own ID, in order to find its closest neighbor in the identifier space. Finally, $\mathcal{P}$ initiates a lookup for a randomly chosen key from every k-bucket that is farther away than its closest neighbor. During these lookups, $\mathcal{P}$ populates its own k-buckets and notifies other peers about its presence (see below for details). After setting up its routing table, $\mathcal{P}$ takes over all references to shared data items that fall into its responsibility.

Maintaining correct routing information. In contrast to Chord, Kademlia does not implement any explicit maintenance algorithm. Instead, each peer updates its routing table on every incoming message. Therefore, the sending peer $\mathcal{S}$ includes its ID in every message, thus allowing the receiving peer $\mathcal{R}$ to update its appropriate k-bucket according to the following algorithm. If $\mathcal{S}$ is already present in $\mathcal{R}$'s k-bucket, $\mathcal{R}$ marks $\mathcal{S}$ as the most-recently seen peer in the k-bucket. Else, if the k-bucket contains less than k entries, $\mathcal{R}$ inserts $\mathcal{S}$ into the k-bucket and marks it as the most-recently seen peer. Else, if $\mathcal{S}$ is not present in $\mathcal{R}$'s k-bucket and the k-bucket is full, then $\mathcal{R}$ sends a Ping message to the least-recently seen peer $\mathcal{P}$ in the k-bucket. If $\mathcal{P}$ responds, it is marked as the most-recently seen peer, and the information about $\mathcal{S}$ is discarded. Only if $\mathcal{P}$ sends no response, $\mathcal{R}$ replaces $\mathcal{P}$ with $\mathcal{S}$ in the k-bucket and marks $\mathcal{S}$ as the most-recently seen peer.

The algorithm to update k-buckets offers two advantages. First, peers that are known to be online for a long time are not removed from the routing table as long as they respond to Ping messages. This is beneficial because the longer a peer is connected to the system, the higher is the probability that it remains connected in the future [SGG02, MM02]. Second, it is impossible to perform a DoS attack by flooding the Kademlia system with new peers and thus capturing peers' routing tables, because new peers will be inserted into k-buckets only if other peers leave.

In the rare case that a peer has not initiated a lookup for a particular k-bucket since a predefined time (typically one hour) and thus k-bucket entries may be outdated, the peer initiates a lookup for a randomly chosen key from the respective k-bucket.

Leaving the Kademlia DHT. The maintenance of routing tables as described above reliably detects peers that have left the system, so Kademlia does not provide an explicit leave procedure. Nevertheless, to maintain a correct mapping of data items onto peers, a leaving peer should transfer its stored references to the next responsible peer.

Summary. Kademlia is a DHT that interprets identifiers as binary numbers and thus defines a binary tree of height b as identifier space. To map data items onto peers, it uses a distance function that is defined by the bitwise exclusive or (XOR) metric. Routing tables have size $O(\log N)$ and contain so-called k-buckets that represent sections of the identifier space in exponentially increasing distance, so that lookups can be answered after $O(\log N)$ routing hops.

2.4.4 Further DHT-based P2P Systems

Besides Chord and Kademlia, many further DHTs have been proposed in recent years. In the following, we briefly explain the primary characteristics and basic algorithms of three other popular DHTs: CAN, Pastry and Viceroy. Note that this list is by far not exhaustive. For information on DHTs that are not discussed in this work, e.g. Koorde [KK03], P-Grid [ACMD$^+$03], Symphony [MBR03] or Tapestry [ZHS$^+$04], we refer to the literature.

CAN. The work on CAN ("Content-Addressable Network") was presented by Ratnasamy *et al.* in 2001 [RFH$^+$01]. CAN sets up a multi-dimensional torus as identifier space. Figure 2.10 shows a two-dimensional example where the identifier space is partitioned among 6 peers. (Note that, for purposes of clarity, we show the identifier space as a plane instead of a torus. Keep in mind that the coordinate space wraps at its borders.)

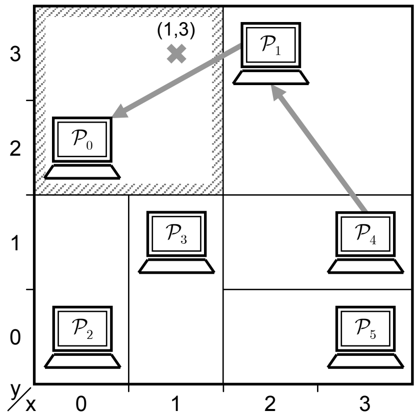


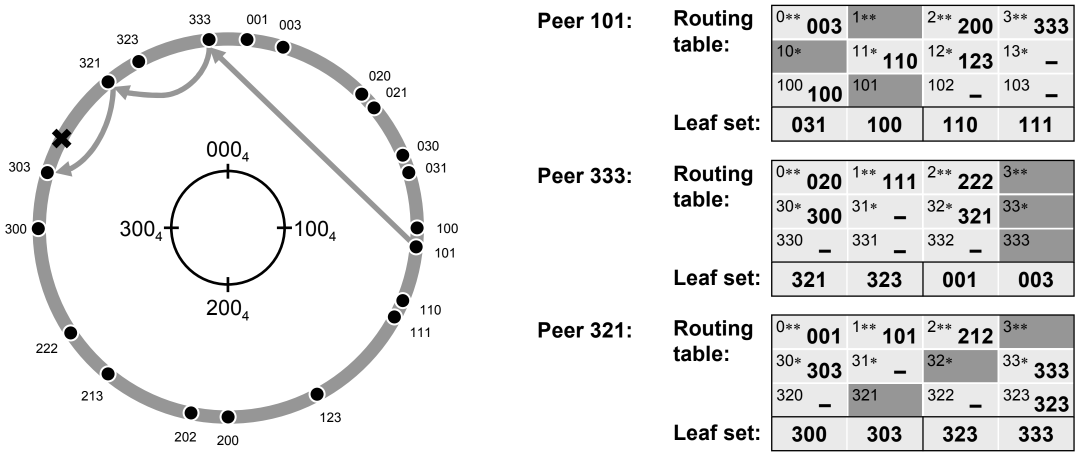
Figure 2.10: Exemplary Content Addressable Network with 6 peers.

Basically, the partition that each peer is responsible for is the zone in which the peer is located. For example, peer $\mathcal{P}_0$ is responsible for the zone $x \in [0; 1],\ y \in [2; 3]$.

Routing tables in CAN are of constant size (2 entries per dimension if all zones are of equal size) and contain the immediate neighbors of the peer in the identifier space. Thus, the routing table of $\mathcal{P}_4$ in Figure 2.10 consists of $\mathcal{P}_3$ (left), $\mathcal{P}_2$ (right), $\mathcal{P}_1$ (up) and $\mathcal{P}_5$ (down). If $\mathcal{P}_4$ initiates a lookup for key $k = (1; 3)$, it will include the destination coordinates in the message and forward the lookup request to its neighbor that is closest to the destination, i.e. $\mathcal{P}_1$. As k is not located in the zone of $\mathcal{P}_1$, $\mathcal{P}_1$ again forwards the message to its neighbor $\mathcal{P}_0$, which in turn is the responsible peer and can answer the lookup request. With this greedy forwarding of lookup requests, CAN achieves an average lookup path length of $1/4 D \cdot N^{1/\mathrm{D}}$ routing hops, where D is the dimension of the identifier space and N the number of peers in the system.

Pastry. Rowstron and Druschel propose a DHT called Pastry [RD01]. Pastry constitutes a ring-shaped identifier space like Chord, and uses b-bit identifiers that are represented by a sequence of digits with base 2^{B}. Typical values are $b = 128$ and $B = 4$. For purposes of clarity, we set $b = 6$ and $B = 2$ in the example from Figure 2.11. Thus, the identifier space ranges from 000_4 to 333_4. The responsible peer for a given key k is the peer that is numerically closest to k.

The routing information at each peer consists of a routing table and a so-called leaf set. The routing table of peer $\mathcal{P}$ contains b/B rows (one row per digit) and 2^{B} columns. The entry in row $x \in [0; b/B - 1]$ and column $y \in [0; 2^{\mathrm{B}} - 1]$ points to a peer whose first x digits are the same as $\mathcal{P}$'s, and whose $(x + 1)^{\mathrm{th}}$ digit is y. The right side of Figure 2.11 shows the routing tables of the peers 101_4, 333_4 and 321_4, respectively. For example, in the routing table of peer 101_4, the entry in row 1 and column 2 points to peer 123_4, as this peer shares the first digit with peer 101_4, and its second digit is 2. Note that, to simplify matters, we characterize peers only by their IDs here. In addition to the routing table, peer $\mathcal{P}$ maintains a leaf set that (typically) contains the $2^{\mathrm{B}-1}$ closest peers with a smaller ID and the $2^{\mathrm{B}-1}$ closest peers with a larger ID. For example, the leaf set of peer 101_4 includes the peers 031_4, 100_4, 110_4 and 111_4.



Peer 101: Routing table:

0** 003	1**	2** 200	3** 333
10*	11* 110	12* 123	13* –
100 100	101	102 –	103 –

Leaf set:

031	100	110	111

Peer 333: Routing table:

0** 020	1** 111	2** 222	3**
30* 300	31* –	32* 321	33*
330 –	331 –	332 –	333

Leaf set:

321	323	001	003

Peer 321: Routing table:

0** 001	1** 101	2** 212	3**
30* 303	31* –	32*	33* 333
320 –	321	322 –	323 323

Leaf set:

300	303	323	333

Figure 2.11: Exemplary Pastry ring with 20 peers.

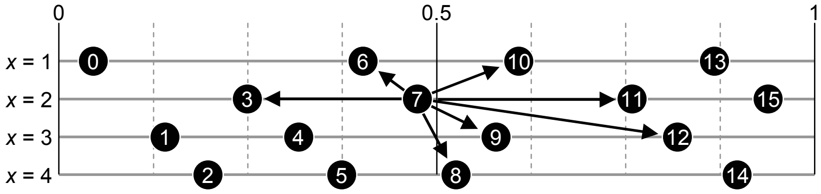
When routing a lookup request for key k to the responsible peer, each peer along the lookup path first checks if a peer in its leaf set is the responsible peer for k. If so, the lookup request will be directly forwarded to this peer. If not, the routing table will be used in the following way. We assume that the first x digits of the peer's ID and the key k match. Then, the peer will choose that peer from its routing table whose ID has a prefix that matches k in at least $x + 1$ digits. In case that the corresponding entry in the routing table is empty, the lookup request is forwarded to a peer whose ID also has the first x digits in common with k but is closer to k. Such a peer is always available (at least in the leaf set) until the lookup request has reached the responsible peer.

An exemplary routing procedure is shown in Figure 2.11. We assume that peer 101_4 initiates a lookup for key $k = 311_4$. In a first step, peer 101_4 selects peer 333_4 from the first row in its routing table as the next hop. Then, peer 333_4 checks the second row in its routing table and finds no corresponding entry for $31*$. Thus, it forwards the lookup request to peer 321_4 which is the closest peer to k it knows. Finally, peer 321_4 checks its leaf set and sends the lookup request to the responsible peer 303_4.

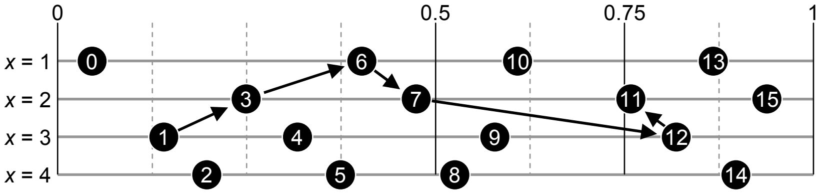
Pastry achieves a lookup path length of $O(\log_{2^B} N)$ routing hops with routing tables that are also of size $O(\log_{2^B} N)$. Additionally, Pastry improves the routing performance by taking network locality into account. This is achieved by preferably including peers in the routing table that are nearby according to an appropriate proximity metric, e.g. the number of IP routing hops in the Internet.

Viceroy. Viceroy, proposed by Malkhi *et al.* in [MNR02], assigns identifiers in the range $[0; 1[$ to peers and data items. The identifier space is represented by multiple rings on different levels, where the number of levels is (ideally) $\log N$. In Figure 2.12(a) we show an example Viceroy system with 16 peers (and thus 4 levels). Note that the identifier rings are depicted as straight lines that wrap at position 1, and that the peers are distributed uniformly across the 4 levels.

The routing table at each peer in a Viceroy system contains 7 pointers to other peers: two "ring pointers" to its predecessor and successor on the global identifier space, two "level ring pointers" to its predecessor and successor on the level ring, and three "butterfly



(a) Viceroy requires only 7 pointers to other peers.



(b) Routing procedure for key 0.75, initiated by peer $\mathcal{P}_1$.

Figure 2.12: Exemplary Viceroy system with 16 peers.

pointers" that are established as follows. We assume that peer $\mathcal{P}$ is located at the x^{th} level and that there are $\log N$ levels in total. Given that $x > 1$, $\mathcal{P}$ maintains an "up pointer" to the clockwise-closest peer of level $x-1$. Given that $x < \log N$, $\mathcal{P}$ maintains a "left down pointer" to the clockwise-closest peer of level $x+1$, and a "right down pointer" to the first peer of level $x+1$ that follows the identifier $id(\mathcal{P}) + 2^{-\mathrm{x}}$. Figure 2.12(a) illustrates these pointers for peer $\mathcal{P}_7$ at level $x = 2$. Its ring pointers point to $\mathcal{P}_6$ and $\mathcal{P}_8$, and its level ring pointers to $\mathcal{P}_3$ and $\mathcal{P}_{11}$. Its up pointer points to $\mathcal{P}_{10}$, its left down pointer to $\mathcal{P}_9$, and its right down pointer to the first peer of level 3 that follows $id(\mathcal{P}_7) + 2^{-2} = 0.47 + 0.25 = 0.72$, i.e. $\mathcal{P}_{12}$.

Although the routing tables are of constant size, Viceroy achieves a lookup path length of $O(\log N)$ routing hops. Viceroy defines the globally closest peer in clockwise direction (= the global successor) of a key k as the responsible peer. To efficiently route lookup requests, the routing procedure is separated into three distinct phases: the "proceed to root" phase, the "traverse tree" phase and the "traverse ring" phase. Figure 2.12(b) shows an example where peer $\mathcal{P}_1$ initiates a lookup for key $k = 0.75$. In the "proceed to root" phase, the lookup request is following the up pointers of the involved peers ($\mathcal{P}_1$ and $\mathcal{P}_3$) until it reaches a level-1 peer ($\mathcal{P}_6$). Then, in the "traverse tree" phase, it follows down pointers until it reaches the lowest level or it overshoots the target. During this phase, each peer that forwards the lookup request will choose its right down pointer if and only if its distance to k is at least $2^{-\mathrm{x}}$, where x is the level at which the peer is located. Otherwise, the left down pointer is chosen. Thus, $\mathcal{P}_6$ forwards the lookup request along its left down pointer to $\mathcal{P}_7$, because its distance to k is smaller than $2^{-\mathrm{x}}$: $s\,(id(\mathcal{P}_6), k) = s\,(0.40, 0.75) = 0.35 < 2^{-1}$. In contrast, $\mathcal{P}_7$ chooses its right down pointer and forwards the lookup request to $\mathcal{P}_{12}$. The reason is that the distance from $\mathcal{P}_7$ to k is greater than 2^{-x}: $s\,(id(\mathcal{P}_7), k) = s\,(0.47, 0.75) = 0.28 \geq 2^{-2}$. At $\mathcal{P}_{12}$, which has ID 0.82, the "traverse tree" phase terminates as the lookup request overshoots the target. In the final “traverse ring” phase, the lookup request is following ring pointers and level ring pointers until it reaches the responsible peer ($\mathcal{P}_{11}$).

2.4.5 Summary

Having explained the basic concepts of structured P2P systems on the example of five popular DHTs (Chord, Kademlia, CAN, Pastry and Viceroy), we conclude this section with a brief comparison of the presented DHTs. Table 2.1 summarizes their main characteristics with regard to the identifier space, the routing table size and the lookup path length in terms of routing hops.

Table 2.1: Comparison of Chord, Kademlia, CAN, Pastry and Viceroy.

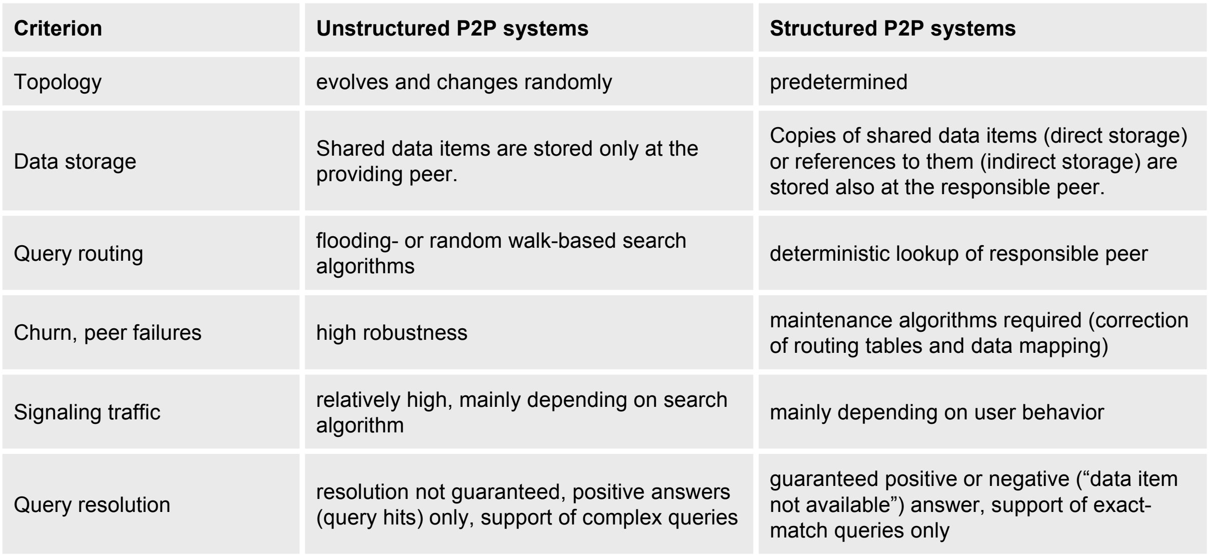
	Identifier space	Routing table size	Lookup path length
Chord	Ring	$O(\log N)$	$O(\log N)$
Kademlia	Binary Tree	$O(\log N)$	$O(\log N)$
CAN	D-dimensional torus	$O(D)$	$O(D \cdot N^{1/\mathrm{D}})$
Pastry	Ring	$O(\log_{2^\mathrm{B}} N)$	$O(\log_{2^\mathrm{B}} N)$
Viceroy	Multiple rings on different levels	$O(1)$	$O(\log N)$

2.5 Comparison of Unstructured and Structured P2P Systems

In the previous sections, we have discussed the basic principles of centralized, unstructured and structured P2P systems, and we have given some popular examples for each of these systems. We close this chapter with a comparison of unstructured and structured P2P systems. Note that we do not include the special case of a centralized P2P system in the comparison, as we do not consider such a system as a true P2P system. Figure 2.13 summarizes the main differences of unstructured and structured P2P systems. The criteria that we use for the comparison are the topology of the P2P system, the storage of provided data items, the routing of queries, the robustness to churn and to failures of participating peers, the generated signaling traffic, and the resolution of queries.

The topology of unstructured P2P systems evolves randomly. Each peer usually chooses its neighboring peers randomly. During system operation, the topology changes continuously with every arriving and departing peer. In contrast, the topology of structured P2P systems is predetermined by the organization of the identifier space and the functions *id* and *neighbors*. During system operation, special algorithms ensure the continuous maintenance of the predefined topology in the presence of churn.

A fundamental property of P2P systems is that peers can easily share their data items with other peers in the system. In unstructured P2P systems, the data items that are provided by a peer are stored only at this peer and nowhere else in the system. Structured P2P systems make use of their predefined topology and store a copy of every shared data



Criterion	Unstructured P2P systems	Structured P2P systems
Topology	evolves and changes randomly	predetermined
Data storage	Shared data items are stored only at the providing peer.	Copies of shared data items (direct storage) or references to them (indirect storage) are stored also at the responsible peer.
Query routing	flooding- or random walk-based search algorithms	deterministic lookup of responsible peer
Churn, peer failures	high robustness	maintenance algorithms required (correction of routing tables and data mapping)
Signaling traffic	relatively high, mainly depending on search algorithm	mainly depending on user behavior
Query resolution	resolution not guaranteed, positive answers (query hits) only, support of complex queries	guaranteed positive or negative ("data item not available") answer, support of exact-match queries only

Figure 2.13: Comparison of unstructured and structured P2P systems.

item (in case of a direct storage strategy) or a reference to every shared data item (in case of an indirect storage strategy) also on the responsible peer in the system, determined by the functions *key* and *partition*. Directly related with the storage of data is the way how queries are routed through the P2P system, in order to discover ideally all peers that share relevant data items. The random nature of unstructured P2P systems requires search algorithms that are based on flooding or the random walk method. Structured P2P systems, in contrast, allow for a deterministic lookup of the responsible peer along a predefined lookup path. Usually, such a DHT lookup requires a number of hops that is logarithmic in the number of participating peers N, by using routing tables that are also of size $O(\log N)$.

To maintain connections to neighboring peers in unstructured P2P systems, a simple Ping/Pong algorithm is sufficient. In case that a neighboring peers does not respond to a Ping message, it is simply replaced by another, currently active peer. As a result, unstructured P2P systems are highly robust to churn and peer failures. Structured P2P systems, however, require sophisticated maintenance algorithms in the presence of churn and peer failures. Their tasks are to correct the entries in the peers' routing tables, and to ensure a correct mapping of shared data items onto peers. This is also reflected in the generated signaling traffic. While a flooding-based search algorithm in unstructured P2P systems generates a relatively high signaling traffic, the signaling traffic in structured P2P systems mainly depends on the user behavior, particularly on the frequency of arriving and departing participants. In case of very high churn rates, structured P2P systems require a significant amount of signaling traffic in order to maintain the system structure.

Last but not least, we compare both approaches with respect to the resolution of queries. In unstructured P2P systems, neither a flooding-based search algorithm nor a random walk can guarantee the resolution of a query, i.e., they cannot guarantee that the query reaches all peers that are currently present in the system. Consequently, only positive answers in form of Query Hit messages are possible. In contrast, structured P2P systems benefit from their predefined mapping of shared data items onto peers. This mapping allows for using the responsible peer as a contact point of providing and requesting peers. Therefore, the resolution of a query is guaranteed, and also negative answers (the information that no matching data item is currently available) are possible. However, there is also an important disadvantage of using hash values to map data items onto peers. While unstructured P2P systems natively support complex queries, i.e. queries with multiple keywords, range queries or queries containing wildcards and/or regular expressions, structured P2P systems basically support only "exact match" queries. Given a key, i.e. the hash value of the specified keyword, they return a value, i.e. the contact information of the providing peer(s). As a result, multiple sophisticated solutions have been developed recently that support complex queries also in DHT-based P2P systems [RV03, BAS04, JFY05, SOTZ05].

Summing up, we state that both unstructured and structured P2P systems have their advantages and disadvantages. The question which lookup concept should be used in a real-world system is depending on the P2P application itself and the communication environment in which it is supposed to work. In this thesis, we put a special focus on heterogeneous environments, characterized by a large diversity in the resources of the participating devices, in the network access technology that connects participants to the P2P system, and in the user behavior. In the following chapter, we describe the main characteristics of such communication scenarios and the resulting challenges for the applied P2P system. Based on the above comparison, we then explain why we have selected DHTs as the basic lookup concept for the solutions proposed later in this work.

3 Heterogeneous Communication Environments

3.1 Introduction

The current mainstream research on P2P systems concentrates on fixed IP networks and usually assumes functionally equal peers. As such, it neglects potential heterogeneity among the connected communication devices. In contrast, the focus of this thesis lies on studying P2P systems that are appropriate to heterogeneous communication environments. In this chapter, we first define what we understand by the term "heterogeneous environments" and describe the main characteristics of such communication scenarios and the resulting challenges for the applied P2P system (Section 3.2). Then, in Section 3.3, we present a survey of an existing P2P file sharing system that helps us to concretize some of the main challenges, by evaluating the impact of content distribution on the applicability of DHTs in a heterogeneous environment.

3.2 Definitions, Characteristics and Challenges

With heterogeneous communication environments, we refer to environments where the communicating entities show a large diversity

- in the hardware resources they provide, such as CPU power, RAM size or storage capacity,
- in the network access technology they use, e.g. hard-wired high speed connections or low speed connections via analog modem or GPRS, and
- in the user behavior, i.e. in all characteristics that do not emerge from the technical infrastructure, but from people using it to communicate.

A typical example of a highly heterogeneous environment is the extension of P2P systems to the world of mobile communication networks, i.e. the integration of mobile devices such as cell phones or handheld computers into a P2P system. Such systems form a capable platform for mobile ubiquitous environments. For example, they leverage the composition of user-provided services or the deployment of context-aware applications. Moreover, from an operator's point of view, P2P systems save infrastructure costs because they are heavily based on user equipment and usually operate without central entities.

In recent years, the area of mobile P2P systems has gained an increasing popularity in the research community. Typically, the work on mobile P2P systems is divided into two main directions: first, the integration of mobile devices into P2P systems as mentioned above, and second, the use of P2P solutions for ad-hoc communication of mobile devices, i.e. the setup and use of a communication environment without any existing infrastructure or central coordination entity. As already mentioned, this thesis studies suitable P2P systems for heterogeneous communication scenarios and thus focuses on the former type of mobile P2P systems.

The heterogeneity of participating peers leads to special characteristics of the considered communication environments:

1. There is a large diversity in the provided resources of peers, especially with regard to the constrained resources of battery-operated and handheld devices. For example, the computational power of workstations or personal computers is far beyond those of cell phones.

2. The numerous technologies for network access lead to a big variation in the transmission data rates of peers. At the moment, they range from multiple Mbit/s in broadband access networks like FTTH or DSL down to some kbit/s when using an analog modem, an ISDN adapter or a wireless GPRS connection.

3. The aforementioned heterogeneity in the user terminals' resources and access technology, together with varying payment models (flat rate, time-based, volume-based), results in a highly varying user behavior. The most important criterion here is the session duration of participants, i.e. how long a peer remains connected to the P2P system. While peers with a flat rate-based broadband connection often stay online for several days, resource-constrained peers like cell phones may be charged comparatively high fees for mobile data transfer, and hence they usually leave the P2P system immediately after retrieving the desired information.

4. Particularly the participation of mobile, wirelessly connected devices in the P2P system is reflected in a significantly increased failure probability of these peers, caused e.g. by wireless link breaks or discharged batteries.

Note that mobility, especially of wirelessly connected peers, is not mentioned in the above enumeration, although one may think of it as another characteristic of the considered scenarios. The reason is that we assume that mobility of peers is covered by the lower layers of the ISO/OSI reference model. When a peer moves, it may change its access technology (e.g. from WLAN to UMTS) and thus its IP address. However, from the P2P system's point of view, this simply corresponds to one leave and one join event of the peer. As such, mobility has no direct impact on the P2P system, but is implicitly included in the above characteristics.

Naturally, these characteristics pose several challenges for P2P systems that are applied in heterogeneous environments:

1. Peers cannot be modeled as one class of nodes anymore, but their varying capacities require different consideration of different types of nodes.

2. The different data rates have to be considered by the applied P2P system so that peers with a low data rate face a lower signaling traffic than peers with a high speed network connection.

3. The short session durations of some peers lead to a high rate of joining and leaving peers, i.e. a high churn rate, resulting in a highly dynamic environment. The P2P system has to be able to manage these dynamics.

4. An appropriate P2P system must be resilient to frequent peer failures, i.e. peers that leave the system without notifying their neighbors.

An elementary question when designing a P2P system is which type of lookup concept should be used – an unstructured, flooding-based one or a structured, DHT-based one? In Section 2.5 we have seen that both approaches have their advantages and disadvantages, and that it usually depends on the particular application scenario which lookup concept is preferable. However, especially in the considered heterogeneous environments, we believe that DHTs outperform flooding-based search algorithms due to the following reasons:

- DHTs offer a deterministic routing path of lookup requests and avoid the high signaling traffic that is generated by flooding-based search algorithms. This is especially important when resource-constrained peers with a low data rate participate in the system.

- DHTs can guarantee the resolution of a query and always provide a response, whether the requested content is available in the system or not.

- Due to their predefined topology, DHTs allow for a good estimation of the generated signaling traffic in the system. This enables the system designer to pre-estimate the resulting traffic load of the participating peers.

As a result, we have decided to focus on DHT-based P2P systems in this thesis. In the following chapters, we propose multiple solutions for their efficient operation in heterogeneous communication environments. Beforehand, we concretize some of the aforementioned challenges by means of a real-world survey. More precisely, we evaluate the impact of content distribution on the applicability of DHT-based P2P systems in heterogeneous environments.

3.3 The Impact of Content Distribution

3.3.1 Motivation

Any P2P system that is applied in a heterogeneous communication environment will operate efficiently only if it considers the special characteristics of such environments. In this context, we assume that DHT-based P2P systems outperform flooding-based systems due to the aforementioned reasons. A major argument is that DHTs establish well-defined key-value-pairs that associate the provided data items with the responsible peers, and thus completely avoid flooding. Nevertheless, especially when resource-constrained devices such as cell phones participate in the system, it is essential to have a well-balanced load among the peers.

In principle, the use of an appropriate hash function allows for subdividing the identifier space equally among all peers in the DHT, i.e. to map a similar amount of keys onto each participating peer. However, the number of keys a peer is responsible for is not a reliable parameter to judge on the load of the peer. The reason is that the number of references to a shared data item with key k that is mapped onto the peer responsible for k is depending on the popularity of the data item. For example, the peer that is responsible for the key of “madonna” will store significantly more references than the responsible peer for the key of “der funkmast”, a relatively unknown German rock band. Therefore, also the distribution of the provided content, i.e. the varying number of copies of all shared data items, has to be considered.

In this section, we evaluate the impact of content distribution on the applicability of DHT-based P2P systems in heterogeneous environments. To be able to do so, we first make a survey in an existing P2P system and try to find as many shared data items as possible, including their absolute frequency (Section 3.3.2). Then, in Section 3.3.3, we analyze the results of the survey and determine the distribution of the discovered content in a DHT with a changing number of peers. Based on this analysis, we concretize some of the main challenges for DHT-based P2P systems and judge on their applicability in heterogeneous environments in Section 3.3.4.

3.3.2 Content Discovery

Our goal is to evaluate the impact of content distribution on the applicability of DHT-based P2P systems in heterogeneous environments. However, there currently exists no large-scale DHT-based P2P system that allows for a detailed survey of the data items that are shared among the users. Therefore, we investigate the content distribution in one of the largest unstructured P2P systems, namely Gnutella [GNU], adopt this distribution for the considered scenarios and base our conclusions on this distribution.

Creation of the keyword list. In a first step to discover as many shared data items as possible in the Gnutella system, we create a list with 578627 different keywords in total. We obtain this list by combining the contents of the following free accessible online dictionaries:

- the “BEOLINGUS” dictionary of TU Chemnitz [BEO] with more than 170000 entries at the time of the survey,
- the “National Puzzlers’ League 2nd Unabridged Dictionary” [DIC] (235544 entries at the time of the survey), and
- the “Official Scrabble™ Player’s Dictionary” [SCR] (172823 entries at the time of the survey)

Additionally, we take into account that the most popular content in Gnutella are compressed audio files. As the names of most song writers and artists cannot be found in a normal dictionary, we also add the content of FreeDB [FDB] (1486728 entries at the time of the survey) to our list. FreeDB is a free music data base that provides information about audio CDs. Combining the data of all these databases and removing duplicated entries finally yields a list with 578627 different keywords.

Modification of the "Mutella" client. As Gnutella is one of the most popular P2P systems with an open protocol specification and open source clients, we use the Gnutella system for our survey. An appropriate open source client for Gnutella is "Mutella", a command line-based Linux client implemented in C++ [MUT]. For our measurements, we have modified the "Mutella" client so that it is able to automatically execute the following tasks:

1. Read a predefined number of keywords from the keyword list.
2. Initiate a query for every keyword.
3. Collect and count the received Query Hit messages for every keyword. Every received Query Hit message represents one peer in the P2P system that shares a data item that matches the keyword.
4. After a given measurement period T_q, write every keyword together with the number of received Query Hit messages into a log file.

Determination of the measurement period T_q. When searching an unstructured P2P system like Gnutella, every Query message is flooded through the system. All peers that receive a Query message and share a data item that matches the included keyword respond with a Query Hit message. Due to this search algorithm, the time between initiating a query and receiving the according Query Hit messages can vary significantly, depending on the number of hops between the querying and the responding peer.

As mentioned above, we limit the time to collect all Query Hit messages for an initiated query to T_q. On the one hand, this measurement period T_q should be as small as possible in order to keep the overall duration of the survey short. On the other hand, T_q must be long enough to receive a Query Hit message from ideally all peers in the system sharing a data item that matches the keyword included in the query. To determine a suitable value for T_q, we use four common keywords ("easy", "madonna", "music" and "contact") and trigger the Gnutella system with a query for every keyword. Then we log the number of received Query Hit messages every five seconds.

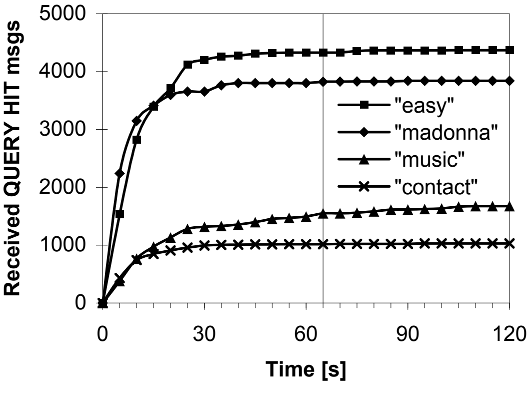


Figure 3.1: Cumulative number of received Query Hit messages over time.

In Figure 3.1, the cumulative number of received Query Hit messages for every keyword is plotted over time. We see that the number of Query Hit messages for every keyword increases significantly at the beginning of the measurement period, and finally reaches a level of saturation where the growth in the number of received Query Hit messages can be neglected. Based on these results, we specify a measurement period of $T_\mathrm{q} = 65$ s for our survey. Thus we can keep the overall time for the measurements short, and at the same time we can guarantee to find a majority of all peers sharing a data item that matches the current keyword.

3.3.3 Analysis

To eliminate statistical aberrations, we run the measurements five times within several weeks. Table 3.1 shows the number of Query Hit messages we received in total for the 578627 keywords. Note that the number of received Query Hit messages reflects the total number of shared data items in the system. It ranges from a minimum of 3991511 to a maximum of 4486803. The average number of received Query Hit messages is 4166156. Throughout the rest of this section we use the averaged values of all five measurements.

Table 3.1: Number of received Query Hit messages.

Number of keywords	578627
Total number of received Query Hit messages – 1st run	4183027
2nd run	4058265
3rd run	4486803
4th run	3991511
5th run	4111175
Average number of received Query Hit messages	4166156

An interesting result from our survey is that 495648 or 85.7% of all keywords result in zero Query Hit messages, i.e. no respective data items can be found in the Gnutella system. For the remaining 82979 keywords, we receive between 1 and 4595 Query Hit messages per keyword.

In Figure 3.2, the cumulative frequency distribution (CFD) of the received Query Hit messages is plotted. Note that we consider only successful queries in this diagram, i.e. queries that lead to at least one Query Hit message. For example, 29.2% of all successful queries result in one Query Hit message, while 40.5% result in one or two Query Hit messages. Altogether, we see that a lot of keywords result only in a few Query Hit messages, while only a minority of the used keywords leads to a high number of Query Hit messages. In other words, many data items in the system are shared by only a few peers, while only a few data items are shared by many peers.

Based on our measurement results in the Gnutella system, we now calculate the resulting load of peers in a DHT, assuming that the same data items that we have discovered in the Gnutella system are now provided in the DHT. We therefore map the data items that match a particular keyword onto the responsible peer in the DHT. For example, the 160-bit SHA-1 [Pub02] key of "madonna" is k_madonna = "64e424263f75a6813399e794d801

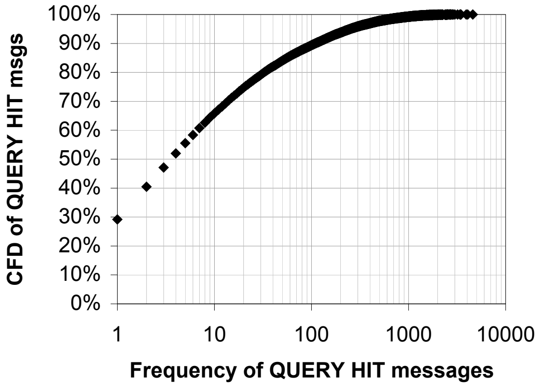


Figure 3.2: Cumulative frequency distribution of received Query Hit messages.

b574fcc1bd99". Thus, we store references to all shared data items that match the keyword "madonna" on the peer in the DHT that is responsible for k_{madonna}.

We assume that the size of the identifier space of the DHT is $[0; 2^{160} - 1]$ and that the identifier space is subdivided equally among the N peers in the DHT, so that every peer is responsible for an equal amount of keys. As a result, the section of the identifier space that the i^{th} peer is responsible for is given by

$$I_{\text{i}} = \left[(i-1) \cdot \frac{2^{160}}{N};\ i \cdot \frac{2^{160}}{N}\right[\quad \forall\, i \in [1; N] \tag{3.1}$$

All Query Hit messages received for a particular keyword are mapped onto the responsible peer for this keyword, so the i^{th} peer stores references to all shared data items with key k that satisfy

$$k \in I_{\text{i}} \tag{3.2}$$

Figure 3.3 shows the load balance that results for our fictive DHT. The number of references per peer is plotted against the total number of peers N that participate in the system. If we assume that every peer stores the same proportion of all references to shared data items, each peer $\mathcal{P}$ will store

$$z_{\mathcal{P},\text{avg}} = \frac{F}{N} \tag{3.3}$$

references, where F is the total number of shared data items in the system. In other words, $z_{\mathcal{P},\text{avg}}$ refers to the average number of references per peer in our experiment. This average value is depicted as a straight line in Figure 3.3 because we use a double-logarithmic scale. Obviously, it decreases with an increasing number of peers because the number of shared data items (and thus the number of references) stays constant. Now we plot the minimum and maximum deviation from this average value for system sizes between 1 and 2^{18} peers, i.e. the number of references stored on the peer with the lowest load and the peer with the highest load, respectively. We see that, although the number of keys per peer is equally distributed, the number of references per peer varies

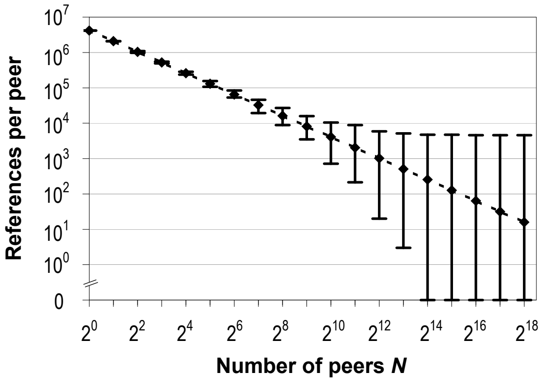


Figure 3.3: Number of references per peer for different system sizes.

significantly. For example, when the DHT consists of $N = 2^{12}$ peers, the average load per peer is

$$z_{\mathcal{P},\mathrm{avg}} = \frac{4166156}{2^{12}} = 1017 \tag{3.4}$$

However, in this case the peer with the highest load stores 5870 references, while the peer with the lowest load stores only 20 references. Further, we see that for $N \geq 2^{14}$ the number of references stored on the highest loaded peer does not decrease anymore with an increasing number of peers in the system. This follows directly from Figure 3.2, as the keyword with the highest number of Query Hit messages (4595 in our survey) is mapped onto one peer and therefore this peer stores all references to the 4595 shared data items that match this keyword. Under the assumption that a reference to a shared data item contains the keyword with a size of 10 Bytes (on average), the hash value of the keyword (160 bit = 20 Bytes) and the IPv6 address and port number of the providing peer (128 bit + 16 bit = 18 Bytes), the data volume that this peer stores is

$$v_{\mathcal{P}} = 4595 \cdot (10 + 20 + 18) \text{ Bytes} = 221 \text{ kB} \tag{3.5}$$

3.3.4 Applicability of DHTs in Heterogeneous Environments

In Section 3.2 we have described the main challenges for P2P systems that are applied in heterogeneous communication environments. These challenges result from the large diversity of peers in terms of provided resources, transmission data rate, session duration and reliability. Now we concretize these challenges with regard to the results from the previous section. There we have found that the minimal data volume that the highest loaded peer stores is 221 kB. In the worst case, this peer is a resource-constrained peer with a low data rate and a high failure probability.

With regard to RAM size and storage capacity, we see that this load is fully acceptable for any kind of contemporary communication device including cell phones and handheld computers. Even if the number of references that are stored on a peer exceeds the value of 4595 (e.g. due to a very low number of peers in the DHT) we can assume enough RAM and storage capacity on the communication device to handle the load.

The situation is different when we take a look at the available transmission data rate. Every time a peer joins the DHT, all references that the peer is responsible for are transmitted from the previously responsible peer to the new peer, in order to maintain the mapping rule of the DHT. In the above example, 221 kB have to be transferred. For devices that use a low speed network connection like an analog modem or GPRS this is a significant amount of data. For example, a cell phone connected via GPRS with a data throughput of 2500 Bytes per second (cf. real-world measurements in [CCP02]) takes more than 88 s (!) to transfer 221 kB of data. Additionally, users that are charged a time-based or volume-based fee for network access usually do not participate in the P2P system for a long time. As a result, the references that are transferred to the peer when joining the system are retransferred back a short period of time later when the peer leaves the P2P system. For this reason, the low data rate and the short session duration of some peers may become a significant challenge for the DHT.

Another critical aspect when using DHT-based P2P systems in heterogeneous environments is the comparatively high failure probability particularly of mobile devices. As such devices commonly have a wireless network connection and only a limited power supply, the probability that a mobile peer fails is notably higher than in a fixed environment. When a peer in a DHT fails, all the references to the shared data items it stores are lost. A failing peer has no possibility to transfer its references to the next responsible peer. This leads to the unavailability of all shared data items that are mapped onto the failed peer, until they are republished by the sharing peers and therefore available again. Possible solutions to overcome this problem are the replication of references on multiple neighboring peers in the identifier space, and the use of short time intervals for republishing shared data items. However, both approaches result in an increased signaling traffic, which in turn is critical for resource-constrained devices such as cell phones or handheld computers.

Concluding, we state that DHT-based P2P systems are, generally speaking, a suitable solution for heterogeneous environments, because they avoid the use of flooding and thus the corresponding high signaling traffic, they can guarantee the resolution of queries, and they allow for an estimation of the generated signaling traffic. However, there remain three main challenges for their efficient operation in heterogeneous environments:

- Peers with a low data rate must be prevented from a high signaling traffic, particularly from a frequent transfer of references.
- The short session duration of some peers must be considered as they result in a continuous shifting of references, which is problematic especially for resource-constrained devices.
- Peers with a high failure probability lead to a high number of lost references and thus to the (at least temporary) unavailability of shared data items.

In the following chapters, we address these challenges and present appropriate solutions.

4 A Hybrid DHT Design Scheme for Heterogeneous Environments

4.1 Introduction

As pointed out in Chapter 3, the special characteristics of heterogeneous communication environments, i.e. the large diversity of the participants with regard to each peer's provided resources, transmission data rate, session duration and reliability, result in three main challenges for the applied DHT-based P2P system. First, the signaling traffic for peers with a low data rate should be minimal. Second, the short session durations of some peers should not lead to a continuous shifting of references. Third, the high failure probability of some peers should not result in a permanent loss of references to shared data items.

In this chapter, we present a novel hybrid design scheme for DHTs that addresses these challenges. In Section 4.2 we point out how to set up such a hybrid DHT, and we explain its differences and advantages in comparison to a conventional DHT. Then, we show by analytical and simulative evaluation in Section 4.3 that the proposed design scheme is a good solution to the challenges named above. In Section 4.4 we compare our solution to related work, while Section 4.5 summarizes the work and the results presented in this chapter.

4.2 Hybrid DHT Design

The key idea of our approach is to set up a hybrid system structure that divides the participating peers into two groups:

- So-called "static peers" are powerful peers in the system that are characterized by long session durations, low failure probabilities and good hardware resources. A typical example of a static peer is an office computer with a hard-wired network connection.

- So-called "temporary peers" are all other peers in the system that have a lower performance than static peers. A typical example of a temporary peer is a mobile device that joins the P2P system only for a short period of time to find some pieces of information, and leaves the system immediately after retrieving the desired information.

Certainly, this differentiation violates a basic principle of P2P systems that considers all participating peers to be equal in functionality and to contribute equally to the system's maintenance and operation. However, especially in heterogeneous environments where

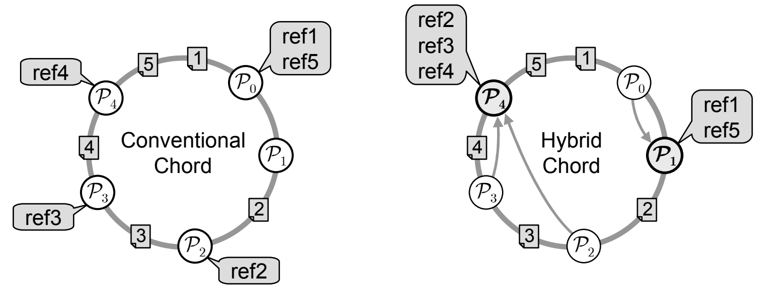


Figure 4.1: Hybrid design scheme for a Chord DHT ($\mathcal{P}_1$ and $\mathcal{P}_4$ are static peers).

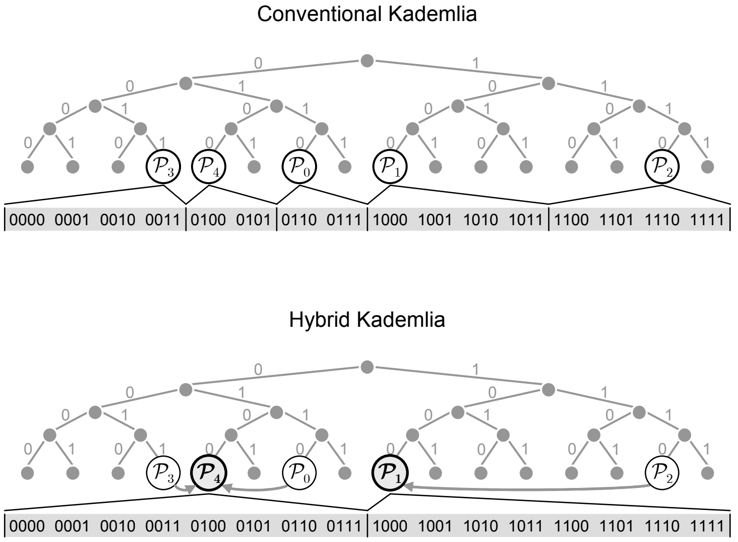


Figure 4.2: Hybrid design scheme for a Kademlia DHT ($\mathcal{P}_1$ and $\mathcal{P}_4$ are static peers).

the capacities of peers diverge heavily, we will see that this differentiation leads to a significantly increased system performance in comparison to a "flat" system design where all peers have the same functionality.

To benefit from the above differentiation of peers, we require a minor modification of the mapping rule of the DHT. In contrast to the conventional DHT protocol, a reference to a shared data item is stored on the closest static peer of the data item's key. (The term "closest" here refers to the distance metric of the DHT. For example in Chord, the distance s from an identifier x to an identifier y is the absolute distance from x to y in clockwise direction, while in Kademlia it is the XOR function of x and y.) On the other hand, temporary peers maintain only a pointer to their closest static peer, instead of storing references they actually were responsible for. Figures 4.1 and 4.2 illustrate this hybrid DHT design scheme for the Chord protocol and the Kademlia protocol, respectively. Whenever a temporary peer in the system receives a lookup request due to its responsibility for the key included in the request, it forwards the lookup request to its closest static peer. The static peer in turn stores the according reference.

```
01 𝓟.joinStatic()
02   setupRoutingTable();
03   𝒮 = find_closest_static(𝓟.id);
04   transfer_matching_references(𝒮);
05   start_timers();

06 𝓟.joinTemporary()
07   setupRoutingTable();
08   𝒮 = find_closest_static(𝓟.id);
09   set_closest_static(𝒮);
10   start_timers();
11   start_closest_static_timer();

12 𝓟.leaveStatic()
13   𝒮 = find_closest_static(𝓟.id);
14   transfer_references(𝒮);
15   inform_neighboring_peers();
16   stop_timers();

17 𝓟.leaveTemporary()
18   inform_neighboring_peers();
19   stop_timers();
```

Figure 4.3: Pseudocode for setting up a hybrid DHT structure.

The hybrid system design can be realized by calling different ‘join’ and ‘leave’ procedures when peers enter or leave the system, depending on the category that the joining/leaving peer belongs to. The respective pseudocode of these procedures is depicted in Figure 4.3. When peer $\mathcal{P}$ joins the system *(line 01, 06)*, it sets up its routing table *(02, 07)* and initiates a lookup of the closest static peer $\mathcal{S}$ *(03, 08)*. If $\mathcal{P}$ joins as a static peer, it will store references to shared data items and thus, all references that lie in the responsibility of $\mathcal{P}$ are transferred from $\mathcal{S}$ to $\mathcal{P}$ *(04)*. In contrast, if $\mathcal{P}$ joins as a temporary peer, it will store only a pointer to $\mathcal{S}$ *(09)*. Finally, the joining peer starts all timers that are required for participation *(05, 10)*. Note that each temporary peer starts one additional timer that periodically verifies the correctness of the pointer to the closest static peer, and updates it when necessary *(11)*. When peer $\mathcal{P}$ leaves the system *(12, 17)*, it informs its neighboring peers *(15, 18)* and stops all running timers *(16, 19)*. Additionally, if $\mathcal{P}$ is a static peer, it will transfer all stored references to its closest static peer in the DHT, which is now the responsible peer for these references *(13, 14)*.

The differentiation between static and temporary peers has three major advantages:

- The varying characteristics and capacities of the participating peers in heterogeneous environments are addressed.

- The maintenance traffic in the system decreases as references to shared data items have to be shifted only when static peers (which usually have long session durations) join or leave the system. Moreover, resource-constrained temporary peers are prevented from storing references.

- Only reliable static peers with a low failure probability store references to shared data items. As a result, the probability that a data item is available in the system but cannot be found because the peer that is responsible for storing a reference to it has failed is reduced. Consequently, the availability of provided content increases.

Because only the static peers in the system store references to shared data items, a high data volume may be transferred when a static peer joins or leaves the system. However, we believe that this signaling traffic does not cause severe problems, as it usually is transmitted over a fixed broadband connection between the static peers. Moreover, due to the long session durations of static peers, join and leave events of static peers occur much less frequently than those of temporary peers.

In the following section, we evaluate the above advantages of the proposed hybrid DHT design scheme, i.e. the reduced maintenance traffic and the increased content availability, and compare it to a conventional DHT.

4.3 Evaluation

For the evaluation of our hybrid DHT design scheme we have implemented the Chord protocol and its hybrid version to which we refer as “Hybrid Chord Protocol”, or “HCP” in short. However, we emphasize that the presented results are almost completely transferable to any other DHT protocol. In a first step we evaluate by which factor the maintenance traffic in the system is reduced when we apply our hybrid design scheme. In a second step we see whether and how much the availability of provided content in the system can be increased.

4.3.1 Reduction of Maintenance Traffic

In general, maintenance traffic in a DHT is necessary because of the continuous changes in the system caused by joining and leaving peers. In this context, two types of maintenance traffic are generated:

1. Maintenance traffic to ensure the correctness of routing table entries: Every peer must periodically check whether the peers it stores in its routing table are still alive and whether they are still the responsible peers for the respective portion of the identifier space. For example in Chord, a peer periodically updates its successor list and its finger table.

2. Maintenance traffic to ensure the correctness of the DHT’s mapping rule: Whenever a peer joins the system, it becomes responsible for a portion of the identifier space. Thus, all references to shared data items that are mapped onto this portion of the identifier space have to be transferred from the previously responsible peer (e.g. the successor of the new peer in Chord) to the new peer. The same applies if a peer leaves the system.

Our hybrid DHT design scheme aims at this second type of maintenance traffic. As we modify only the mapping rule of the DHT protocol and not the DHT protocol itself, we cannot reduce the first type of maintenance traffic. This traffic is the same in both cases, the conventional DHT and its hybrid version. As a result, in the following we evaluate the reduction of the second type of maintenance traffic, i.e. the number of references that are transferred during system operation.

Theoretical analysis. Before we present our simulation results we analytically determine by which factor the number of transferred references can be reduced in a DHT that applies our hybrid design scheme. Under the assumption that F is the total number of shared data items in a system with N peers, each peer $\mathcal{P}$ is responsible for storing

$$z_{\mathcal{P},\text{avg}} = \frac{F}{N} \tag{4.1}$$

references on average. Thus, a mean amount of $z_{\mathcal{P},\text{avg}}$ references has to be shifted when a peer joins or leaves the system. The total number of join and leave events ε in the system per time unit is determined by the number of peers N and their mean session duration T_{session} (one join and one leave event per session):

$$\varepsilon = N \cdot \frac{2}{T_{\text{session}}} \tag{4.2}$$

As a result, the system's rate τ of the number of references that are transferred to joining or from leaving peers is given by

$$\tau = z_{\mathcal{P},\text{avg}} \cdot \varepsilon = \frac{F \cdot 2}{T_{\text{session}}} \tag{4.3}$$

In a conventional DHT all peers store references. Therefore, the relevant session duration in equation 4.3 is the mean session duration $T_{\text{session,all}}$ of all peers in the system. In contrast, our hybrid DHT protocol stores references only on the static peers. Therefore, their mean session duration $T_{\text{session,static}}$ is the relevant session duration in equation 4.3. The total number of references F is the same in both cases.

Now we see the advantage of our solution. By definition, the mean session duration of static peers is significantly higher than the mean session duration of all peers in the system:

$$T_{\text{session,static}} = x \cdot T_{\text{session,all}} \quad \text{with } x >> 1 \tag{4.4}$$

As a result, our hybrid DHT design scheme reduces the number of references that are transferred during system operation by a factor of

$$\frac{\tau_{\text{hybrid}}}{\tau_{\text{conventional}}} = \frac{T_{\text{session,all}}}{T_{\text{session,static}}} = \frac{1}{x} \quad \text{with } x >> 1 \tag{4.5}$$

in comparison to the conventional DHT protocol.

Simulation setup. To verify the above analysis and to compare the performance of the conventional Chord and the Hybrid Chord Protocol (HCP) we have implemented both protocols in the Network Simulator 2 (ns-2) [NS2]. Since ns-2 simulates every layer of the ISO/OSI reference model it requires a high amount of CPU power and RAM. Thus, the size of the simulated system is very limited. To improve simulation performance, we simplify the model of the physical network and add a randomly chosen latency between 10 ms and 200 ms to every message exchanged between two peers. Further we assume that no computing time is needed to process incoming messages. For the creation of simulation scenarios we have implemented a traffic generator that performs the following tasks:

- Definition of different classes of peers: For the simulated systems, we define different classes of participating peers. Appropriate parameters of a peer class are the mean session duration, the failure probability (i.e. the probability that the peer leaves the system without notifying neighboring peers), the average lookup rate, and the number of shared data items. (Note that, for the evaluation of maintenance traffic, only the mean session duration and the number of shared data items are relevant parameters. For the evaluation of content availability in Section 4.3.2, also the failure probability and the average lookup rate have to be specified.)
- Creation of an initial P2P system: The traffic generator creates an initial Chord or HCP system with a given number of peers, including the setup of predecessor pointers, successor lists, finger tables, 'closest static' pointers (for temporary peers in HCP) and shared data items.
- Generation of an event file: The event file is created according to the specified parameters of all peer classes and is used as an input for the simulator.

Figure 4.4 illustrates an exemplary scenario file specifying a simulation scenario with three different classes of peers. In detail, the whole simulation process runs as follows. First, the traffic generator parses the scenario file and generates – according to the parameters given in the scenario file – an event file containing the initial P2P system as well as simulation events. Simulation events include join, leave, fail and lookup events. Second, the generated event file is taken as an input for the simulator, which in turn produces a tracing file that can finally be analyzed with appropriate evaluation tools.

```
PeerClass WLAN_NOTEBOOK
     MeanSessionDuration 3600s
     FailureProbability 10%
     MeanTimeBetweenLookups 300s
     SharedDataItems 50

PeerClass UMTS_PDA
     MeanSessionDuration 900s
     FailureProbability 25%
     MeanTimeBetweenLookups 120s
     SharedDataItems 10

PeerClass GPRS_PHONE
     MeanSessionDuration 300s
     FailureProbability 50%
     MeanTimeBetweenLookups 60s
     SharedDataItems 5

Quantity
     100 WLAN_NOTEBOOK
     100 UMTS_PDA
     100 GPRS_PHONE

SimulationDuration 3600s
```

Figure 4.4: Scenario file for setting up a simulation scenario.

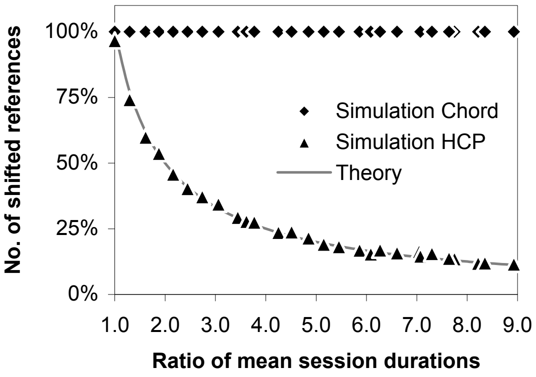


Figure 4.5: Reduction of maintenance traffic in scenario 1.

Two peer classes. In a first simulation, we set up a P2P system with 1000 peers, divided into two classes. The main goal of this simulation is to verify our theoretical analysis in a P2P system with a high percentage of resource-constrained peers such as wirelessly connected handheld devices. We simulate multiple scenarios that we generate according to the parameters given in Table 4.1. Note that we increase the mean session duration of static peers from 5 to 50 minutes in subsequent simulation runs, and that the individual session durations of peers are negative exponentially distributed around the specified mean value.

Table 4.1: Simulation parameters for scenario 1.

Number of peer classes:	2
Peer class:	STATIC
Mean session duration:	300 s ... 3000 s (neg. exp. distr.)
Number of shared data items:	1 per peer
Peer class:	TEMPORARY
Mean session duration:	300 s (neg. exp. distr.)
Number of shared data items:	1 per peer
Mean number of online peers:	1000
Partitioning:	100 STATIC, 900 TEMPORARY
Simulation duration:	2 hours

In total, we generate 56 different event files where the ratio of the mean session duration of static peers to the mean session duration of all peers in the system ranges from 1.0 to 8.9. The individual values of this ratio result directly from varying mean session durations of the participating peers in every particular simulation scenario. We simulate every scenario with the conventional Chord and the Hybrid Chord Protocol. In Figure 4.5, the resulting maintenance traffic in terms of shifted references is depicted. The 100% line represents the number of transferred references that we measure in a conventional Chord DHT. As expected, the number of transferred references in a hybrid Chord DHT decreases with an increasing ratio of mean session durations, which is in this case resulting from an increasing mean session duration of the static peers. In conformity with our theoretical analysis, the number of transferred references is reduced inversely proportional to the ratio of the mean session durations of static peers and all peers in the system.

Real-world scenario. Our first simulation has considered a theoretical scenario with only two different classes of peers. To compare Chord and HCP in a more realistic scenario, we set up a heterogeneous P2P system with 2000 peers, partitioned into five different classes: 100 office computers, 700 DSL subscribers, 400 ISDN users, 400 UMTS participants and 400 GPRS-connected cell phones. Table 4.2 illustrates the modeling of these peer classes.

Table 4.2: Simulation parameters for scenario 2.

Number of peer classes:	5
Peer class:	Office
Mean session duration:	24 hours (neg. exp. distr.)
Number of shared data items:	0 ... 30
Peer class:	DSL
Mean session duration:	2 hours (neg. exp. distr.)
Number of shared data items:	0 ... 30
Peer class:	ISDN
Mean session duration:	30 minutes (neg. exp. distr.)
Number of shared data items:	0 ... 15
Peer class:	UMTS
Mean session duration:	10 minutes (neg. exp. distr.)
Number of shared data items:	0 ... 8
Peer class:	GPRS
Mean session duration:	2 minutes (neg. exp. distr.)
Number of shared data items:	0 ... 5
Mean number of online peers:	2000
Partitioning:	100 Office, 700 DSL, 400 ISDN, 400 UMTS, 400 GPRS
Simulation duration:	1 hour

As in the previous simulation, the session durations of peers belonging to a particular class are distributed negative exponentially around the mean value of this peer class. The number of shared data items of each peer is selected randomly from the specified range. The simulated time is one hour. When simulating HCP, we allow only peers that belong to the classes Office and DSL to become static peers, and thus to store references to shared data items. Peers from classes ISDN, UMTS and GRPS act as temporary peers in this case. To have a statistical validation of our simulation results, we perform 10 independent simulation runs and show the average values of all runs. In addition, we measure the 90% confidence intervals every 5 minutes.

Figure 4.6 shows the maintenance traffic (in terms of transferred references per minute) of both protocols over time. Since a large portion of the system consists of peers with a

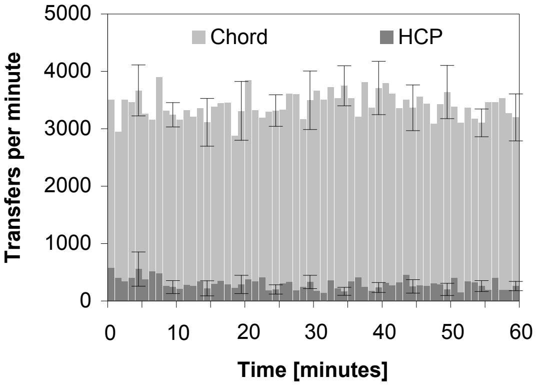


Figure 4.6: Reduction of maintenance traffic in scenario 2.

comparatively short session duration, we notice a continuously high amount of transferred references in Chord. In contrast, HCP offers a significantly decreased maintenance traffic, as references to shared data items are stored only on the static peers (Office and DSL peers) which are characterized by longer session durations.

4.3.2 Improvement of Content Availability

In the previous section, we have found that the proposed hybrid DHT design scheme can significantly reduce the maintenance traffic in a DHT by decreasing the amount of references transferred from one peer to another during system operation. This section evaluates a second advantage of our concept: the improved availability of provided content.

A DHT maps the data items that are provided by the peers onto the responsible peers in the DHT, according to the DHT's mapping rule. The main drawback of this mapping is that peers store references to content that is provided by other peers. In case of a peer failure, all references stored on the failed peer are lost. Consequently, the related content is unavailable, although the providing peers are still online.[1] This is especially true for heterogeneous environments, where we expect high failure probabilities of some (usually mobile) peers, e.g. due to wireless link breaks or discharged batteries. To keep references within the DHT up-to-date and to ensure content availability even in the presence of peer failures, all participating peers have to republish their shared data items periodically. Huebsch *et al.* [HHB$^+$03] evaluate this scheme by determining content availability as a function of failure probability and republish period length. However, they do not consider the signaling traffic that is generated by republishing data items, as described in the following.

[1] One approach to handle this problem is the replication of references on multiple peers in the system. We abandon from this method because of two reasons. First, it generates additional signaling traffic and may not be reliable enough if the failure probability of peers is high (as expected in the considered heterogeneous environments). Second, replication linearly increases the number of references stored in the system, and thus the maintenance traffic generated when peers join or leave the system (cf. Section 4.3.1).

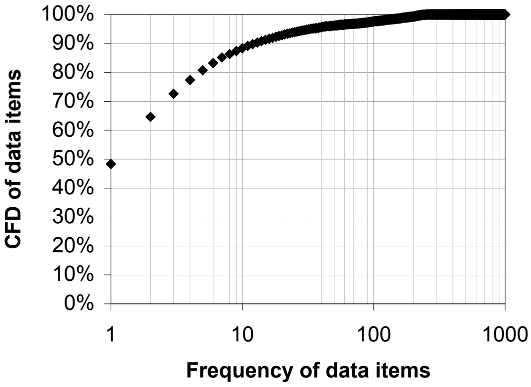


Figure 4.7: Cumulative frequency distribution of shared data items.

Republishing a shared data item requires the lookup of the currently responsible peer in the DHT, and naturally this generates signaling traffic in the P2P system. As a result, we face a trade-off between content availability and signaling traffic. On the one hand, short intervals for republishing shared data items ensure a high content availability and therefore a high user acceptance, but on the other hand also generate a high amount of signaling traffic, which may be harmful to resource-constrained peers such as mobile devices.

To estimate the impact of republish intervals on the availability of provided content in a heterogeneous P2P system, we perform multiple simulations in a Chord system with 1000 peers that we partition equally into five classes: Office, DSL, ISDN, UMTS and GPRS. In particular, we assign a varying mean session duration, failure probability, lookup rate and number of shared data items to each class. The detailed values are given in Table 4.3.

As in Section 4.3.1, the individual session durations are distributed negative exponentially around the mean value of the respective peer class. The same applies for the lookup rates of peers. We model the shared data items in the system as follows. When created, a peer selects the number of data items that it shares randomly from the range that is specified for the class that the peer belongs to. Then, for each particular data item that is generated, the peer has two options. With 10% probability it generates a new data item that is not available in the system so far. With 90% probability it selects a data item that is already available on one or more other peers. With this implicit distribution function, we obtain a cumulative frequency distribution (CFD) of shared data items as depicted in Figure 4.7. This frequency distribution follows the frequency distribution that we have measured in reality (cf. Section 3.3.3). There are many data items that are available on only a few peers, while only a minority of data items is shared by a high number of peers.

To evaluate the content availability we refer to the search recall. The search recall is defined as the ratio of the number of peers that are returned in the result set of a lookup and the number of all peers currently sharing the requested data item. For example, if the result set of a lookup for key k contains the peers $\mathcal{P}_0$ and $\mathcal{P}_1$ as hosts of the data

Table 4.3: Simulation parameters for scenario 3.

Number of peer classes:	5
Peer class:	OFFICE
Mean session duration:	10 hours (neg. exp. distr.)
Failure probability:	5%
Lookup rate:	1 every 10 minutes (neg. exp. distr.)
Number of shared data items:	100 ... 200
Peer class:	DSL
Mean session duration:	5 hours (neg. exp. distr.)
Failure probability:	5%
Lookup rate:	1 every 10 minutes (neg. exp. distr.)
Number of shared data items:	50 ... 100
Peer class:	ISDN
Mean session duration:	30 minutes (neg. exp. distr.)
Failure probability:	15%
Lookup rate:	1 every 5 minutes (neg. exp. distr.)
Number of shared data items:	25 ... 50
Peer class:	UMTS
Mean session duration:	10 minutes (neg. exp. distr.)
Failure probability:	30%
Lookup rate:	1 every 60 seconds (neg. exp. distr.)
Number of shared data items:	10 ... 20
Peer class:	GPRS
Mean session duration:	5 minutes (neg. exp. distr.)
Failure probability:	50%
Lookup rate:	1 every 30 seconds (neg. exp. distr.)
Number of shared data items:	5 ... 10
Mean number of online peers:	1000
Partitioning:	200 OFFICE, 200 DSL, 200 ISDN, 200 UMTS, 200 GPRS
Simulation duration:	3 hours

item with key k, and the peers $\mathcal{P}_0$, $\mathcal{P}_1$ and $\mathcal{P}_2$ are currently providing this data item, then the search recall will be $^2/_3$ or 67%.

The simulation of a conventional Chord DHT that is depicted in Figure 4.8 and Figure 4.9(a) verifies our above assumptions. Figure 4.8 shows an increasing content availability for shorter republish intervals, i.e. the more often shared data items are republished, the lower is the proportion of lookups with a low search recall. For example, if the republish interval is set to 1800 s, 28% of all lookups will have a search recall of at most 75%. An equal search recall will be observable only in 2% of all lookups if the republish interval is set to 60 s. However, we also encounter an increasing signaling traffic with shorter republish intervals, as shown in Figure 4.9(a). While a republish interval

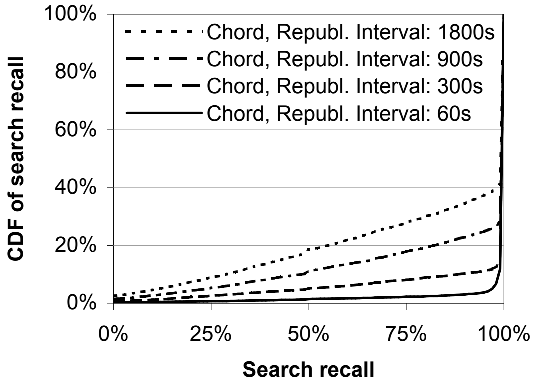
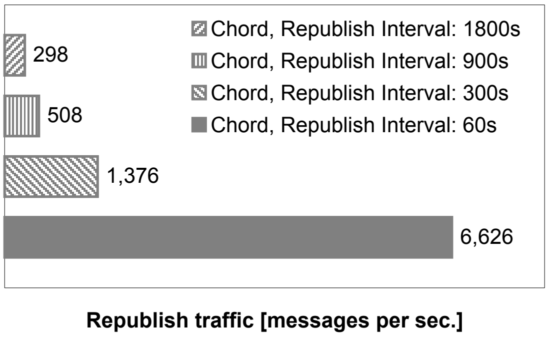
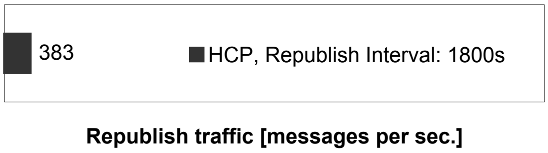


Figure 4.8: Content availability in Chord.



(a) Republish traffic in Chord.



(b) Republish traffic in HCP.

Figure 4.9: Signaling traffic caused by a periodic republishing of shared data items.

of 1800 s generates only 298 republish messages per second in the system, a republish interval of 60 s results in a high number of 6,626 republish messages per second.

Now we simulate the same scenario also with a hybrid Chord DHT. In particular, we compare storing references only on reliable peers (Office and DSL peers in this case) with a republish interval of 1800 s, to storing references on all peers (= conventional DHT) with republish intervals of 60 s and 1800 s, respectively. The simulation results are shown in Figure 4.9(b) and Figure 4.10. With regard to the signaling traffic that is caused by a periodic republishing of shared data items (cf. Figure 4.9(b)), we see that the Hybrid Chord Protocol requires only a slightly increased signaling traffic (383 republish messages per second) in comparison to the conventional Chord protocol applying the same

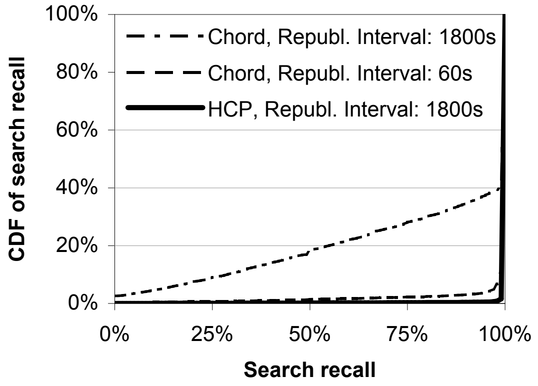


Figure 4.10: Comparison of content availability.

republish interval of 1800 s. This results from forwarding incoming lookup requests from temporary to static peers. On the other hand, a hybrid DHT offers an excellent content availability. As shown in Figure 4.10, the search recall of nearly all lookups reaches 100% when storing references only on reliable Office and DSL peers and republishing shared data items every 1800 s. The content availability in this case is even higher than for a conventional Chord DHT with a republish interval of 60 s.

The above simulations prove the increased content availability with a hybrid DHT design that results from storing references only on reliable static peers. From our point of view, content availability is an important aspect for the provider of DHT-based services, as the successful resolution of queries is an important criterion for user acceptance and hence the number of customers.

4.4 Related Work

One of the first works that consider the heterogeneity of peers is [CRB+03]. The authors propose a dynamic topology adaptation algorithm that selects peers with a high capacity to have a high degree, i.e. a high number of connections to other peers, while peers with a low capacity are within a short reach of these high capacity peers. In combination with an active flow control scheme that avoids overloaded peers, an efficient replication strategy, and a search algorithm that is based on random walks, the proposed P2P system shows significant advantages (three to five orders of magnitude) in comparison to the conventional Gnutella [GNU] protocol. However, in contrast to our hybrid DHT design scheme, the approach in [CRB+03] is based on an unstructured P2P system.

[HK03] focuses particularly on the problem of peer mobility in DHTs where some peers move to other network attachment points and are thus assigned a new IP address. This usually corresponds to one leave and one join event in the DHT, and may lead to inconsistent routing table entries, signaling traffic due to shifted references, or unavailability of shared data items. The authors propose a system design that, similar to our approach, classifies the participating peers as "stationary" and "mobile" peers. The key idea is that stationary peers as well as mobile peers are organized in two separate DHTs, and that stationary peers are responsible for storing the current IP address of mobile peers. When a mobile peer with ID x moves and is assigned a new IP address y, it updates the key-value-pair $\langle x; y \rangle$ on the stationary peer that is responsible for ID x. Thus, the DHT that is built by stationary peers forms a location information repository that can be used to find the current IP address of any mobile peer, provided that its ID is known and does not change with a changing IP address. As a result, peer movements are transparent to the P2P system. In addition, the authors present an optimized routing scheme that requires $O(\log N)$ hops to resolve the responsible peer in the system. Although the proposed system design can reduce the maintenance traffic and increase the availability of shared content in the presence of moving peers (with changing IP addresses), it mostly ignores the main challenges of resource-constrained peers in a DHT that we address with our hybrid design scheme, i.e. low data rates, short session durations and high failure probabilities.

A DHT design scheme that is closely related and has been published shortly after our work is the idea of a "stealth" DHT, presented in [BMR$^+$06]. A stealth DHT also splits the participating peers into two different types called "service nodes" and "stealth nodes". While the service nodes (which are expected to be highly capable, reliable machines) are normal peers in the DHT, the stealth nodes only set up their own routing table when joining the system, but do not announce their presence to other peers, so they never appear in the routing table of other peers. Yet they are able to use their own routing table to send lookup messages into the DHT. As a result, stealth nodes are "invisible" to the DHT routing procedure and only service nodes forward lookup messages. Clearly, this includes that references to shared data items are stored only on service nodes, and the resulting advantages (unloading of resource-constrained peers, reduction of maintenance traffic, increased content availability) are the same that our hybrid DHT design scheme offers. However, the isolation of stealth nodes leads to a problem that we avoid with our concept. As stealth nodes are not addressable by service nodes, they never receive any update messages when service nodes enter or leave the system. Consequently, it is difficult to keep the routing table of stealth nodes up to date, and in contrast to our design scheme, it requires additional algorithms (called "rejoining", "periodic polling" and "piggybacking") to circumvent this problem.

In general, all aforementioned approaches that try to enable an efficient use of DHTs in heterogeneous communication environments benefit from the fact that they consider the varying resources and capacities of peers, and that they assign different tasks to different types of peers. In doing so they can be seen as a first step towards hierarchical DHTs. As we will see in the following chapters, hierarchical DHTs are an extension to the ideas we have presented so far, and thus form appropriate architectures for P2P systems in heterogeneous environments.

4.5 Summary

In this chapter we have presented a hybrid DHT design scheme that aims at the extension of DHT-based P2P systems to heterogeneous communication environments. By defining two different types of participating peers, static and temporary peers, and storing references to shared data items only on the static peers, our concept allows for unloading resource-constrained peers such as cell phones or handheld computers, for a significant reduction of maintenance traffic, and for an increased availability of provided content.

We have performed multiple simulations of a conventional Chord DHT and its hybrid version in different scenarios. The simulations prove our theoretical analysis that the hybrid DHT reduces the maintenance traffic (in terms of shifted references) by a factor of $1/x$ in comparison to the conventional DHT, where x is the ratio of the mean session duration of static peers in the hybrid DHT and the mean session duration of all peers in the conventional DHT. Moreover, the simulations verify the increased availability of provided content that can be achieved with our hybrid design scheme.

5 An Analytical Approach for Optimal Hierarchical DHT Design

5.1 From ‘Hybrid’ to ‘Hierarchical’ P2P Systems

In the previous chapter we have proposed a design scheme for DHTs where the peers are divided into static peers and temporary peers. This differentiation offers significant performance gains, especially when the P2P system is applied in a heterogeneous environment. We named our design scheme “hybrid”, pointing out that the system is composed of two different types of peers that build a single DHT, without setting up an explicit hierarchy among the peers.

In general, there are numerous important questions for the designer of such a system, e.g. ”which peers should become static peers, which peers should become temporary peers?” or ”what is the best number of static peers in the system?”. Before answering these questions, we extend the ideas in the previous chapter by introducing hierarchical DHTs as a more general approach of designing efficient DHT-based P2P systems for heterogeneous environments.

Formally, we define a hierarchical DHT as a P2P system that

- organizes all participating peers into h hierarchical layers, with $h > 1$,
- defines different tasks to the peers in different layers,
- allows transition between the layers, and
- uses one or multiple DHTs as the basic lookup concept.

While the hybrid DHT design scheme from Chapter 4 provides already a good solution to the challenges of DHT-based P2P systems in heterogeneous environments, hierarchical DHTs offer additional benefits. For example, resource-constrained peers can be put onto the lowest layer in the hierarchy and therefore be kept completely outside the basic DHT. In the following chapters, we exploit the benefits of hierarchical DHTs and develop a self-organizing hierarchical DHT-based P2P system that is especially suited for heterogeneous communication environments. Note that we choose Chord here as the applied DHT. However, the presented methods, analyses and results are smoothly transferable to other DHT protocols.

The remainder of this chapter is structured as follows. Section 5.2 discusses related work. In Section 5.3 we introduce three types of hierarchical DHTs that we analyze and compare in the following. Therefore, we first provide a formal analytical cost model that enables us to calculate the total operation costs of each considered system (Section 5.4). Then, in Section 5.5 we explain how to derive the optimal operating point from total operation costs and the individual costs of superpeers, in order to judge on the system optimality. Based on this cost model we compare the three considered systems in Section 5.6. The influence of potential changes in the assumptions we make is discussed in Section 5.7. Section 5.8 concludes this chapter.

5.2 Related Work

There are a number of reasons why hierarchical P2P systems came in the focus of P2P research. The most important ones include the recognition that they reduce the total traffic generated in the underlying physical network, they offer a better fault isolation and they fit better to the inherent heterogeneity of P2P systems. All these reasons are relevant for unstructured and structured P2P systems. Chronologically, the success of the early version of Gnutella was followed by the introduction of hierarchical unstructured P2P systems such as Gnutella version 0.6 [G06] or FastTrack [FTR]. The main reason for this development was the recognition that the signaling traffic becomes too high for large systems and that the search recall (the ratio of the number of received Query Hit messages and the total number of available copies of the data item in the system) can be increased if the system is kept smaller. We can observe a similar trend in the DHT research field, although it is motivated by a different reason. Churn has been found to be the most serious challenge to a wide adoption of DHTs. If the churn rate (the rate of peers joining or leaving the system) becomes too high, the maintenance algorithm of the DHT is not able to fix incorrect routing table entries fast enough, and thus the system performance decreases seriously. Therefore, it makes sense to organize the system in two or more hierarchical layers and enable distinction between peers with different session durations and capacities. Powerful peers, which have higher sessions durations, are put in the highest hierarchical layer, while low performance peers participate in the lower layers so that the impact of their short session durations on the system operation is reduced as much as possible.

The research on hierarchical DHT-based system architectures usually makes a distinction between vertical and horizontal hierarchical DHTs [SAGPS05]. A vertical system architecture follows a tree-like hierarchy in which every node is a self-contained DHT. These DHTs are often referred to as clusters. Transition from a lower layer cluster to an upper layer cluster is realized through one or more dedicated gateway nodes (often referred to as “superpeers”) which are members of both clusters at the same time. [GEBF$^+$03] presents a system with one such a superpeer per cluster, [XMH03] presents the other extreme in which all peers participate in all hierarchical clusters, while [MD04] does not put any constraints on the peers with respect to this. The key design point of vertical hierarchical system architectures is that only the superpeers have multiple routing tables, one for each hierarchical cluster they participate in. The other peers (often referred to as “normal peers” or “leafnodes”) are aware only of their primary cluster so that their routing tables refer only to peers from their own cluster. We also mention [MCKS03], which bears some similarity to the SiCo architecture that we introduce and analyze in the following sections. The main difference is that the top-layer DHT in [MCKS03] is essentially a full graph and that any superpeer and its leafnodes constitute a set of consecutive IDs viewed from the perspective of the entire system. In our SiCo architecture there are no constraints on the peer IDs. [LV05] proposes another vertical approach, similar to [GEBF$^+$03], adding just the possibility to group semantically close peers (e.g. those interested in similar content) in the lower layer.

In a horizontal hierarchical DHT, the peers in the lowest layer clusters form DHTs as usual. When forming a higher layer cluster, each peer adds links toward a number of peers from sibling DHTs. It is important to emphasize that these links are carefully chosen so that the total number of links per peer remains logarithmic in the total number of peers. Thus, there are no gateway peers (superpeers) in the sense mentioned above. This is in our opinion the key distinguishing point between vertical and horizontal hierarchical DHTs. [SAGPS05] and [GGGM04] are two well-known examples of horizontal hierarchical DHTs.

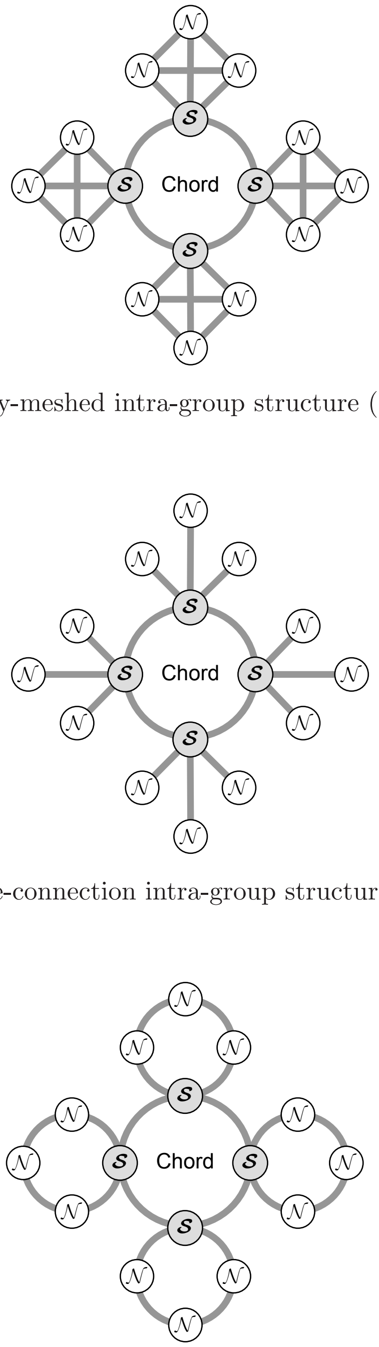
Reduction of the total number of maintained connections is typically mentioned as the main advantage of horizontal hierarchical DHTs. On the other hand, the vertical hierarchy is more appropriate for heterogeneous environments, because it makes a clear distinction among the load that different peers are supposed to take in the system. Because such environments are the main target of this thesis, we focus on vertical hierarchical DHTs in the following.

As far as the cost-based modeling aspect of our analysis is concerned, we are aware only of attempts to apply such techniques to flat DHTs ([CC05] presents an example). To the best of our knowledge, no modeling of hierarchical DHTs in a cost-based framework can be found in the literature.

5.3 Hierarchical DHT-based System Architectures

In the following sections, we study hierarchical DHT-based P2P systems. More precisely, we consider hierarchical systems that establish two layers of hierarchy: a higher layer built by so-called “superpeers”, and a lower hierarchy layer (so-called “groups”) in which all normal peers participate. A typical hierarchical P2P system is set up e.g. by the Gnutella 0.6 file sharing application [G06]. Gnutella 0.6 uses an unstructured P2P system to route messages between superpeers, and manages groups simply by connecting each normal peer to a randomly selected superpeer.

This work analyzes different types of hierarchical systems. In the higher layer, we apply a DHT (Chord in our case) between the superpeers in the system. This allows us to benefit from the advantages of DHTs in comparison to unstructured P2P systems, e.g. the avoidance of flooding and the deterministic resolution of queries. Every superpeer manages a group of normal peers at the lower layer. To connect peers within a group, we study three alternatives of intra-group structures: the fully-meshed intra-group structure (we abbreviate it “FuMe”), where every group peer is connected to all other group peers including the superpeer, the single-connection intra-group structure (“SiCo”), where group peers are connected only to the group’s superpeer, and the DHT intra-group structure (“2DHT”), where a second DHT (Chord in our case) is applied to connect the peers within each group. Figures 5.1(a), 5.1(b) and 5.1(c) depict the respective system architectures with superpeers (“$\mathcal{S}$”) and normal peers (“$\mathcal{N}$”).



(a) Fully-meshed intra-group structure (FuMe).

(b) Single-connection intra-group structure (SiCo).

(c) DHT intra-group structure (2DHT).

Figure 5.1: Analyzed hierarchical DHT-based system architectures.

In the analyzed systems, a unique b-bit ID is associated with all participating peers, with superpeers as well as with normal peers. Also every shared data item is labeled with a key k. For our analysis, we assume references to shared data items to be stored on both types of peers, on superpeers and on normal peers. Therefore, we require a key length of $2b$ bits in our hierarchical systems, with the first b bits being independent from the second b bits. The first b bits of k are used for determining the responsible group, the second b bits for determining the responsible peer within that group. This is necessary to achieve a balanced storage load among the peers of a group. Otherwise, if a key length of only b bits was used (for determining first the responsible group and then again for determining the responsible group peer), then in a group that is responsible for all keys $k \in [a; b]$ only those group peers would store references whose IDs also lie in [a; b].

As aforementioned, we assume references to shared data items to be stored on both types of peers. For our analysis, we make this assumption as a basis for comparing the three hierarchical system architectures. There is also the possibility of storing references only on superpeers. However, this does not allow to compare the different system structures as it requires the single-connection intra-group structure and makes the fully-meshed intra-group structure and the DHT intra-group structure superfluous. In Section 5.7.3 we discuss the alternative of storing references only on superpeers.

When references are stored on superpeers and normal peers, the responsible peer for a given key is determined by first resolving the responsible group (i.e. the responsible superpeer) in the top-layer DHT through a DHT lookup, and then by determining the responsible peer within that group. In the FuMe architecture and the SiCo architecture, a group's superpeer knows the IDs of each of its group peers as it directly exchanges messages with them. In the 2DHT architecture, a DHT lookup has to be performed to find the responsible group peer.

Note that even in case that the peer $\mathcal{I}$ that initiates the lookup and the peer $\mathcal{R}$ that is responsible for storing the corresponding key-value-pair are within the same group, a lookup has to be routed via the superpeer of $\mathcal{I}$, as $\mathcal{I}$ does not have a priori knowledge about the responsible group (= the responsible superpeer) in the system.

5.4 Traffic Analysis

In the following we develop a formal analytical cost model of the considered hierarchical system architectures. In particular, we calculate the total operation costs in the analyzed system architectures and express these costs as a function of the fraction of so-called superpeers in the system, i.e. peers operating in the top-layer DHT. Additionally, we introduce a quantity called "load level" that allows us to consider each peer's capacity and therefore to avoid overload, i.e., not to assign a load that exceeds the peer's load limit. This leads us to the definition of the optimal operating point of each considered system architecture, i.e. the optimal ratio between peers in the top-layer DHT and peers at the lower layer. Finally, the comparison of total operation costs in this optimal operating point allows us to compare the analyzed system architectures and to decide which one is optimal.

Throughout the following analysis we evaluate costs in terms of the number of transmitted signaling messages. The intuition behind is to assume equal costs per message at each peer, and to model the heterogeneity of peers by assigning a varying load limit, i.e. a varying maximum number of messages that peers can process per time unit. To evaluate all messages that are generated in the analyzed system architectures we make the following definitions and assumptions:

- Total number of peers in the system: N
- Number of superpeers = Number of groups: N_{SP}
- Superpeer ratio: $\alpha = \dfrac{N_{\mathrm{SP}}}{N}$
- Number of normal peers: $N_{\mathrm{NP}} = (1 - \alpha) \cdot N$
- Group size = Number of peers per group: $G = \dfrac{N}{N_{\mathrm{SP}}} = \dfrac{1}{\alpha}$
  (Chapter 6 presents a distributed algorithm to achieve equal group sizes across superpeers.)
- Lookup rate of peer $\mathcal{P}$: $q_{\mathcal{P}}$
- Total number of lookups per time unit: $Q = \sum\limits_{\mathcal{P}} q_{\mathcal{P}}$
- Number of data items shared by peer $\mathcal{P}$: $f_{\mathcal{P}}$
- Total number of shared data items: $F = \sum\limits_{\mathcal{P}} f_{\mathcal{P}}$
- The system is in a steady state, i.e. no churn occurs. (Section 5.7.1 discusses the impact of churn.)
- All DHTs implement an iterative routing scheme. (Section 5.7.2 addresses recursive routing.)
- Each group is managed by one superpeer. (We do not address redundancy aspects here when multiple superpeers are responsible for a group.)
- A group peer always knows its group's superpeer.
- All groups in the system apply the same intra-group structure.
- Extensions like caching or replication are not included in the model. Section 5.7.3 addresses the option of storing references only on superpeers.

We begin our analysis with the signaling traffic that is generated by lookups. In a second step, we evaluate the necessary signaling traffic for maintaining the particular system architecture. Finally, we analyze the signaling traffic caused by republishing all shared data items periodically. We therefore define $l_{\mathcal{P}}^{s/r}$ as the number of lookup messages sent/received by peer $\mathcal{P}$ (per time unit), $m_{\mathcal{P}}^{s/r}$ as the number of maintenance messages sent/received by peer $\mathcal{P}$ (per time unit), and $r_{\mathcal{P}}^{s/r}$ as the number of republish messages sent/received by peer $\mathcal{P}$ (per time unit). The sum of lookup messages, maintenance messages and republish messages then yields the total number of messages $c_{\mathcal{P}}^{s/r}$ sent/received by peer $\mathcal{P}$ (per time unit), i.e. the traffic load (= the individual costs) of peer $\mathcal{P}$. In case that the number of sent messages $c_{\mathcal{P}}^{s}$ is equal to the number of received messages $c_{\mathcal{P}}^{r}$, we also use the shortened notation $c_{\mathcal{P}}$.

5.4.1 Lookup Traffic

A lookup in the considered hierarchical DHT-based P2P systems generally involves four peers: the peer $\mathcal{I}$ that initiates the lookup, the superpeer of $\mathcal{I}$'s group $\mathcal{S}_1$, the superpeer of the responsible group $\mathcal{S}_2$, and the responsible peer for the lookup request $\mathcal{R}$. The lookup procedure, illustrated in Figure 5.2, can be divided into six steps:

1. $\mathcal{I}$ forwards the lookup request to $\mathcal{S}_1$.
2. $\mathcal{S}_1$ resolves the responsible group for the lookup request in the top-layer DHT.
3. $\mathcal{S}_1$ forwards the lookup request to the responsible group's superpeer $\mathcal{S}_2$.
4. $\mathcal{S}_2$ resolves the responsible group peer $\mathcal{R}$ by using the intra-group structure.
5. $\mathcal{S}_2$ forwards the lookup request to $\mathcal{R}$.
6. $\mathcal{R}$ sends the corresponding key-value-pair directly back to $\mathcal{I}$.

In the analyzed system architectures, three special cases of a lookup can occur. First of all, the peer that initiates the lookup is already a superpeer, i.e. $\mathcal{I} = \mathcal{S}_1$. In that case, step 1 is unnecessary. Second, the responsible peer $\mathcal{R}$ is superpeer $\mathcal{S}_1$, then steps 2 to 5 are unnecessary. Third, the responsible peer $\mathcal{R}$ is superpeer $\mathcal{S}_2$, which makes steps 4 and 5 unnecessary. In a hierarchical P2P system we usually have much more normal peers than superpeers, so the special cases above happen rarely. Further, these special cases result only in minor changes to the general lookup procedure. Therefore, we do not consider them in our analysis. In other words, we consider a superpeer to be represented by two

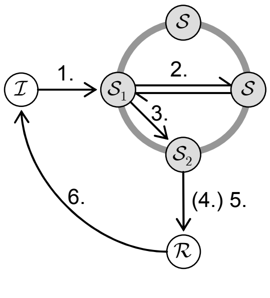


Figure 5.2: Lookup procedure in a hierarchical DHT-based P2P system.

"virtual" nodes, one participating as a normal peer at the lower layer and one as a superpeer in the top-layer DHT, and we count the messages between both virtual nodes as normal messages, though they are sent and received by the same physical node.

On every lookup, the peer $\mathcal{I}$ that initiates the lookup sends one message (step 1) and receives one message (step 6). Therefore, the total number of messages of any peer $\mathcal{I}$ is given by

$$l_{\mathcal{I}} = q_{\mathcal{I}} \tag{5.1}$$

On every lookup, $\mathcal{I}$'s superpeer $\mathcal{S}_1$ receives one message in step 1. Then, to resolve the responsible superpeer $\mathcal{S}_2$ in the top-layer DHT, it performs a DHT lookup. According to Section 2.4.2, this lookup requires $1/2 \log N_{\text{SP}}$ hops on average. As we assume iterative routing, every hop generates two messages, one Request Successor message from $\mathcal{S}_1$ to the current superpeer along the lookup path, and one Response Successor message back to $\mathcal{S}_1$, including the next hop. Finally, $\mathcal{S}_1$ sends one message in step 3. If we assume that every superpeer processes an equal proportion of all Q initiated lookups, the total number of messages of any superpeer acting as $\mathcal{S}_1$ is given by

$$l_{\mathcal{S}_1} = \frac{Q}{N_{\text{SP}}} \cdot (1/2 \log N_{\text{SP}} + 1) \tag{5.2}$$

The routing of a top-layer DHT lookup also involves other superpeers. On average, $1/2 \log N_{\text{SP}}$ superpeers are contacted during a top-layer DHT lookup. As seen above, every involved superpeers receives a Request Successor message and sends a Response Successor message. If we assume that the top-layer routing of all Q initiated lookups involves all superpeers equally, the total number of messages of any superpeer $\mathcal{S}$ involved in top-layer DHT lookups is given by

$$l_{\mathcal{S}} = \frac{Q}{N_{\text{SP}}} \cdot 1/2 \log N_{\text{SP}} \tag{5.3}$$

On every lookup, the responsible group's superpeer $\mathcal{S}_2$ receives one message (step 3) and sends one message (step 5). Depending on the applied intra-group structure, $\mathcal{S}_2$ may need additional $l_{\mathcal{S}_2,\text{add}}$ messages to resolve the responsible group peer $\mathcal{R}$ (step 4). If we assume that every group is responsible for an equal proportion of all Q initiated lookups, the total number of messages of any superpeer acting as $\mathcal{S}_2$ is given by

$$l_{\mathcal{S}_2} = \frac{Q}{N_{\text{SP}}} + l_{\mathcal{S}_2,\text{add}} \tag{5.4}$$

In the FuMe architecture and the SiCo architecture, superpeer $\mathcal{S}_2$ knows the responsible group peer $\mathcal{R}$ and can forward the lookup request without additional signaling traffic. In the 2DHT architecture, $\mathcal{S}_2$ first needs to resolve $\mathcal{R}$. On average, this generates $1/2 \log G$ messages for $\mathcal{S}_2$. Consequently, we state that

$$l_{\mathcal{S}_2,\text{add(FuMe)}} = l_{\mathcal{S}_2,\text{add(SiCo)}} = 0 \tag{5.5}$$

in the FuMe architecture and the SiCo architecture, and that

$$l_{\mathcal{S}_2,\text{add(2DHT)}} = \frac{Q}{N_{\text{SP}}} \cdot 1/2 \log G \tag{5.6}$$

in the 2DHT architecture. The resolution of $\mathcal{R}$ in an intra-group DHT also involves other group peers. If we assume that all G group peers are involved equally in the resolution of all Q/N_{SP} lookups of their group, the total number of messages of a normal group peer $\mathcal{N}$ to resolve $\mathcal{R}$ is given by

$$l_{\mathcal{N},\text{add(2DHT)}} = \frac{Q}{N_{\text{SP}} \cdot G} \cdot {}^{1}\!/_{2} \log G \tag{5.7}$$

Finally, the responsible peer $\mathcal{R}$ receives one message (step 5) and sends one message (step 6) on every lookup. If we assume that every peer is responsible for an equal proportion of all Q initiated lookups, the total number of messages of any responsible peer $\mathcal{R}$ is given by

$$l_{\mathcal{R}} = \frac{Q}{N} = \frac{Q}{N_{\text{SP}} \cdot G} \tag{5.8}$$

With the above equations we can calculate the signaling traffic that is generated by lookups, for normal peers and for superpeers. Note that a normal peer only acts as initiating peer $\mathcal{I}$ and as responsible peer $\mathcal{R}$ (and is involved in intra-group lookups when a DHT is used as intra-group structure), while a superpeer is also involved in top-layer DHT lookups.

In the FuMe architecture and the SiCo architecture, any normal peer $\mathcal{N}$ sends and receives

$$l^{s}_{\mathcal{N}(\text{FuMe})} = l^{r}_{\mathcal{N}(\text{FuMe})} = l^{s}_{\mathcal{N}(\text{SiCo})} = l^{r}_{\mathcal{N}(\text{SiCo})} = l_{\mathcal{I}} + l_{\mathcal{R}} = q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G} \tag{5.9}$$

messages, while the number of sent and received messages of any superpeer $\mathcal{S}$ is given by

$$\begin{aligned} l^{s}_{\mathcal{S}(\text{FuMe})} &= l^{r}_{\mathcal{S}(\text{FuMe})} = l^{s}_{\mathcal{S}(\text{SiCo})} = l^{r}_{\mathcal{S}(\text{SiCo})} = l_{\mathcal{I}} + l_{\mathcal{S}_1} + l_{\mathcal{S}} + l_{\mathcal{S}_2} + l_{\mathcal{R}} = \\ &= q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot ({}^{1}\!/_{2} \log N_{\text{SP}} + 1) + \frac{Q}{N_{\text{SP}}} \cdot {}^{1}\!/_{2} \log N_{\text{SP}} + \frac{Q}{N_{\text{SP}}} + \frac{Q}{N_{\text{SP}} \cdot G} = \\ &= q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2 \right) \end{aligned} \tag{5.10}$$

In the 2DHT architecture, any normal peer $\mathcal{N}$ sends and receives

$$\begin{aligned} l^{s}_{\mathcal{N}(\text{2DHT})} &= l^{r}_{\mathcal{N}(\text{2DHT})} = l_{\mathcal{I}} + l_{\mathcal{N},\text{add(2DHT)}} + l_{\mathcal{R}} = \\ &= q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G} \cdot {}^{1}\!/_{2} \log G + \frac{Q}{N_{\text{SP}} \cdot G} = \\ &= q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G} \cdot ({}^{1}\!/_{2} \log G + 1) \end{aligned} \tag{5.11}$$

messages, while the number of sent and received messages of any superpeer $\mathcal{S}$ is given by

$$
\begin{aligned}
& l^s_{\mathcal{S}(\text{2DHT})} = l^r_{\mathcal{S}(\text{2DHT})} = l_{\mathcal{I}} + l_{\mathcal{S}_1} + l_{\mathcal{S}} + l_{\mathcal{S}_2} + l_{\mathcal{R}} = \\
& = q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot (1/2 \log N_{\text{SP}} + 1) + \frac{Q}{N_{\text{SP}}} \cdot 1/2 \log N_{\text{SP}} + \frac{Q}{N_{\text{SP}}} + \frac{Q}{N_{\text{SP}}} \cdot 1/2 \log G + \\
& \quad + \frac{Q}{N_{\text{SP}} \cdot G} = \\
& = q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + 1/2 \log G + \frac{1}{G} + 2 \right)
\end{aligned}
\tag{5.12}
$$

5.4.2 Maintenance Traffic

Any DHT-based P2P system requires periodical maintenance algorithms. Given the natural churn that occurs in P2P systems, they are necessary to maintain the specified structure of the DHT in case of joining and leaving peers, and to detect failed peers that left the system without notifying their neighbors. In the considered hierarchical system architectures both the top-layer DHT and the lower layer groups implement such maintenance algorithms.

Top-layer DHT maintenance. As stated in Section 5.3 we study a Chord system as top-layer DHT between superpeers. To maintain the Chord DHT, every superpeer periodically runs the Stabilize and the FixFingers algorithm. We analyze the original Stabilize algorithm that is proposed in [SMK+01] and do not consider extensions from [RGRK04]. We further use a slightly improved FixFingers algorithm that periodically sends a Finger Ping message to every peer in the finger table, and initiates a finger lookup only when no Finger Pong message is received or when the finger peer indicates that a new peer is responsible for this finger ID. This modification avoids unnecessary finger lookups when the system is in a steady state.

Every superpeer runs the Stabilize algorithm periodically every T_{STAB} seconds. Stabilize requires three messages. The initiating superpeer sends a Request Predecessor message to its successor, the successor responds with a Response Predecessor message and finally the initiating peer sends a Notify message. Therefore, the number of sent and received Stabilize messages $m^s_{\mathcal{S},\text{STAB}}$ and $m^r_{\mathcal{S},\text{STAB}}$ of any superpeer $\mathcal{S}$ is given by

$$
m^s_{\mathcal{S},\text{STAB}} = m^r_{\mathcal{S},\text{STAB}} = \frac{3}{T_{\text{STAB}}}
\tag{5.13}
$$

Every superpeer runs the FixFingers algorithm periodically every T_{FIX} seconds for each of its $\log N_{\text{SP}}$ fingers. As stated above, two messages are required per finger peer. Therefore, the number of sent and received FixFingers messages $m^s_{\mathcal{S},\text{FIX}}$ and $m^r_{\mathcal{S},\text{FIX}}$ of any superpeer $\mathcal{S}$ is given by

$$
m^s_{\mathcal{S},\text{FIX}} = m^r_{\mathcal{S},\text{FIX}} = \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}}
\tag{5.14}
$$

Here we assume that every superpeer receives the same number of Finger Ping messages from other superpeers as it sends to them.

In total, every superpeer $\mathcal{S}$ sends and receives

$$m^s_{\mathcal{S}(\text{TL})} = m^r_{\mathcal{S}(\text{TL})} = \frac{3}{T_{\text{STAB}}} + \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}} \tag{5.15}$$

messages for maintaining a top-layer (TL) DHT with N_{SP} superpeers.

Maintenance of intra-group structure. Depending on the applied intra-group structure, different algorithms are required for group maintenance. In the fully-meshed intra-group structure and the single-connection intra-group structure, a Ping/Pong algorithm ensures the detection of failed peers. When a Chord DHT is used as intra-group structure, the maintenance algorithms explained above, Stabilize and FixFingers, are applied.

In the fully-meshed intra-group structure, Ping and Pong messages are exchanged between all group peers. To avoid unnecessary double pinging, we assume that a peer with ID x sends Ping messages only to peers in the group with an ID $y > x$. Peers that receive a Ping message respond with a Pong message, thus ensuring that both peers receive a "keep alive" message from each other. This Ping/Pong algorithm is executed every T_{PING} seconds. Therefore, the number of sent and received Ping/Pong messages $m^s_{\mathcal{P}(\text{FuMe})}$ and $m^r_{\mathcal{P}(\text{FuMe})}$ of any group peer $\mathcal{P}$, i.e. of the superpeer and of each normal peer, is given by

$$m^s_{\mathcal{P}(\text{FuMe})} = m^r_{\mathcal{P}(\text{FuMe})} = \frac{G-1}{T_{\text{PING}}} \tag{5.16}$$

The single-connection intra-group structure also uses a Ping/Pong algorithm to detect failed peers. The group peers periodically (every T_{PING} seconds) send a Ping message to their superpeer, and the superpeer responds with a Pong message. Therefore, the number of sent and received Ping/Pong messages $m^s_{\mathcal{N}(\text{SiCo})}$ and $m^r_{\mathcal{N}(\text{SiCo})}$ of any normal group peer $\mathcal{N}$ is given by

$$m^s_{\mathcal{N}(\text{SiCo})} = m^r_{\mathcal{N}(\text{SiCo})} = \frac{1}{T_{\text{PING}}} \tag{5.17}$$

while the number of sent and received Ping/Pong messages $m^s_{\mathcal{S}(\text{SiCo})}$ and $m^r_{\mathcal{S}(\text{SiCo})}$ of a group's superpeer $\mathcal{S}$ is given by

$$m^s_{\mathcal{S}(\text{SiCo})} = m^r_{\mathcal{S}(\text{SiCo})} = \frac{G-1}{T_{\text{PING}}} \tag{5.18}$$

When a Chord DHT is applied as intra-group structure, the Stabilize and FixFingers algorithms ensure the consistency of routing tables and the detection of failed group peers. The analysis of this signaling traffic is similar to the above analysis of the top-layer DHT. Thus, we state that every group peer $\mathcal{P}$, i.e. the superpeer and each normal peer, sends and receives

$$m^s_{\mathcal{P}(\text{2DHT})} = m^r_{\mathcal{P}(\text{2DHT})} = \frac{3}{T_{\text{STAB}}} + \frac{\log G \cdot 2}{T_{\text{FIX}}} \tag{5.19}$$

messages for maintaining an intra-group Chord DHT with G peers.

5.4.3 Republish Traffic

In DHT-based P2P systems every peer periodically republishes its shared data items. This is necessary to keep the references on the responsible peer up-to-date, as peers may fail and thus references may be lost. In the following we analyze how much signaling traffic is generated by republishing shared data items. Note that this analysis does not consider replication of references. Replication, i.e. storing references not only on the responsible peer but also on multiple subsequent peers, can be used in order to reduce the necessity of republishing shared data items. However, as we put a main focus on heterogeneous environments where some peers have a limited capacity and show a high failure probability, we abandon from this method because of two reasons. First, it requires additional signaling traffic. Second, replication linearly increases the number of references stored in the system. As a result, also the maintenance traffic generated by shifting references when peers join or leave the system increases linearly with the number of replicas.

Every peer runs the republish algorithm periodically every T_{REP} seconds for each of its shared data items. The total number of republish events per time unit $e_{\mathcal{P}}$ of any peer $\mathcal{P}$ is therefore given by

$$e_{\mathcal{P}} = \frac{f_{\mathcal{P}}}{T_{\text{REP}}} \tag{5.20}$$

The total number of republish events in the system per time unit E can be expressed by

$$E = \sum_{\mathcal{P}} e_{\mathcal{P}} = \frac{F}{T_{\text{REP}}} \tag{5.21}$$

Republishing a shared data item corresponds to a lookup of the responsible peer, according to the data item's key. The only difference is that the republishing peer sends the key-value-pair, and the responsible peer acknowledges the reception. Consequently, we compute the republish traffic by using the analysis of lookup traffic in Section 5.4.1, and by replacing a peer's lookup rate $q_{\mathcal{P}}$ by its number of republish events per time unit $e_{\mathcal{P}}$ and the total number of lookups Q by the total number of republish events E.

Hence, in the FuMe architecture and the SiCo architecture, any normal peer $\mathcal{N}$ sends and receives

$$r^s_{\mathcal{N}(\text{FuMe})} = r^r_{\mathcal{N}(\text{FuMe})} = r^s_{\mathcal{N}(\text{SiCo})} = r^r_{\mathcal{N}(\text{SiCo})} = \frac{f_{\mathcal{N}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}} \cdot G} \tag{5.22}$$

messages, while the number of sent and received messages of any superpeer $\mathcal{S}$ is given by

$$r^s_{\mathcal{S}(\text{FuMe})} = r^r_{\mathcal{S}(\text{FuMe})} = r^s_{\mathcal{S}(\text{SiCo})} = r^r_{\mathcal{S}(\text{SiCo})} = \frac{f_{\mathcal{S}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2\right) \tag{5.23}$$

In the 2DHT architecture, any normal peer $\mathcal{N}$ sends and receives

$$r^s_{\mathcal{N}(\text{2DHT})} = r^r_{\mathcal{N}(\text{2DHT})} = \frac{f_{\mathcal{N}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}} \cdot G} \cdot (1/2 \log G + 1) \tag{5.24}$$

messages, while the number of sent and received messages of any superpeer $\mathcal{S}$ is given by

$$r^s_{\mathcal{S}(\text{2DHT})} = r^r_{\mathcal{S}(\text{2DHT})} = \frac{f_{\mathcal{S}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + 1/2 \log G + \frac{1}{G} + 2\right) \tag{5.25}$$

5.4.4 Summary

In the previous analysis we have derived expressions that specify the number of messages that normal peers and superpeers send and receive on average, for resolving lookups, maintaining the hierarchical system architecture and republishing shared data items. By combining these expressions we can calculate the total number of messages of normal peers and of superpeers in the analyzed hierarchical system architectures.
In the FuMe architecture, any normal peer $\mathcal{N}$ sends/receives

$$c^{s/r}_{\mathcal{N}(\text{FuMe})} = l^{s/r}_{\mathcal{N}(\text{FuMe})} + m^{s/r}_{\mathcal{P}(\text{FuMe})} + r^{s/r}_{\mathcal{N}(\text{FuMe})} \tag{5.26}$$

messages. Any superpeer $\mathcal{S}$ sends/receives

$$c^{s/r}_{\mathcal{S}(\text{FuMe})} = l^{s/r}_{\mathcal{S}(\text{FuMe})} + m^{s/r}_{\mathcal{S}(\text{TL})} + m^{s/r}_{\mathcal{P}(\text{FuMe})} + r^{s/r}_{\mathcal{S}(\text{FuMe})} \tag{5.27}$$

messages per time unit.
In the SiCo architecture, any normal peer $\mathcal{N}$ sends/receives

$$c^{s/r}_{\mathcal{N}(\text{SiCo})} = l^{s/r}_{\mathcal{N}(\text{SiCo})} + m^{s/r}_{\mathcal{N}(\text{SiCo})} + r^{s/r}_{\mathcal{N}(\text{SiCo})} \tag{5.28}$$

messages per time unit. Any superpeer $\mathcal{S}$ sends/receives

$$c^{s/r}_{\mathcal{S}(\text{SiCo})} = l^{s/r}_{\mathcal{S}(\text{SiCo})} + m^{s/r}_{\mathcal{S}(\text{TL})} + m^{s/r}_{\mathcal{S}(\text{SiCo})} + r^{s/r}_{\mathcal{S}(\text{SiCo})} \tag{5.29}$$

messages per time unit.
In the 2DHT architecture, any normal peer $\mathcal{N}$ sends/receives

$$c^{s/r}_{\mathcal{N}(\text{2DHT})} = l^{s/r}_{\mathcal{N}(\text{2DHT})} + m^{s/r}_{\mathcal{P}(\text{2DHT})} + r^{s/r}_{\mathcal{N}(\text{2DHT})} \tag{5.30}$$

messages per time unit. Any superpeer $\mathcal{S}$ sends/receives

$$c^{s/r}_{\mathcal{S}(\text{2DHT})} = l^{s/r}_{\mathcal{S}(\text{2DHT})} + m^{s/r}_{\mathcal{S}(\text{TL})} + m^{s/r}_{\mathcal{P}(\text{2DHT})} + r^{s/r}_{\mathcal{S}(\text{2DHT})} \tag{5.31}$$

messages per time unit.

Table 5.1 summarizes the results of the above traffic analysis. It shows the average number of sent and received messages, respectively, of normal peers and of superpeers, divided into lookup, maintenance and republish messages. It can be used as a mathematical base for any analysis in vertical hierarchical Chord-based P2P systems.

Table 5.1: Expressions for lookup, maintenance and republish traffic for normal peers and superpeers.

	Fully-meshed intra-group structure	Single-connection intra-group structure	DHT intra-group structure
$l_{\mathcal{N}}$	$q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G}$	$q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G}$	$q_{\mathcal{N}} + \frac{Q}{N_{\text{SP}} \cdot G} \cdot ({}^{1}\!/_{2} \log G + 1)$
$m_{\mathcal{N}}$	$\frac{G-1}{T_{\text{PING}}}$	$\frac{1}{T_{\text{PING}}}$	$\frac{3}{T_{\text{STAB}}} + \frac{\log G \cdot 2}{T_{\text{FIX}}}$
$r_{\mathcal{N}}$	$\frac{f_{\mathcal{N}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}} \cdot G}$	$\frac{f_{\mathcal{N}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}} \cdot G}$	$\frac{f_{\mathcal{N}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}} \cdot G} \cdot ({}^{1}\!/_{2} \log G + 1)$
$l_{\mathcal{S}}$	$q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2\right)$	$q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2\right)$	$q_{\mathcal{S}} + \frac{Q}{N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + {}^{1}\!/_{2} \log G + \frac{1}{G} + 2\right)$
$m_{\mathcal{S}(\text{TL})}$	$\frac{3}{T_{\text{STAB}}} + \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}}$	$\frac{3}{T_{\text{STAB}}} + \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}}$	$\frac{3}{T_{\text{STAB}}} + \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}}$
$m_{\mathcal{S}}$	$\frac{G-1}{T_{\text{PING}}}$	$\frac{G-1}{T_{\text{PING}}}$	$\frac{3}{T_{\text{STAB}}} + \frac{\log G \cdot 2}{T_{\text{FIX}}}$
$r_{\mathcal{S}}$	$\frac{f_{\mathcal{S}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2\right)$	$\frac{f_{\mathcal{S}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + \frac{1}{G} + 2\right)$	$\frac{f_{\mathcal{S}}}{T_{\text{REP}}} + \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot \left(\log N_{\text{SP}} + {}^{1}\!/_{2} \log G + \frac{1}{G} + 2\right)$

5.5 Optimal Operating Point

In this section we apply the above traffic analysis to determine the optimal operating point of each considered system architecture. With "optimal operating point" we refer to the optimal ratio of superpeers to normal peers in the system at which the costs in terms of the total signaling traffic are minimized while not overloading any peer in the system.

5.5.1 Superpeer ratio

Our goal is to determine optimal configurations of the three considered hierarchical structures, to compare them, and to decide which one is best, at least in settings of practical interest. It is easy to see that the configurations are fully determined by specifying the fraction of peers we place in the top layer of the hierarchy. We call this quantity the superpeer ratio α and define it as $\alpha = N_{\mathrm{SP}}/N = 1/G$. Table 5.2 shows the expressions from Table 5.1 as functions of α. They are obtained by replacing N_{SP} with $\alpha \cdot N$ and G with $1/\alpha$.

In the following we concentrate on superpeer ratios $\alpha \leq 25\%$, which corresponds to group sizes of $G \geq 4$. The reason is that for group sizes of $G = 3$ the DHT intra-group structure equals the fully-meshed intra-group structure. Further, if the group size is $G = 2$, we have only one normal peer attached to every superpeer, hence the intra-group structure is always a single-connection intra-group structure. Finally, for $\alpha = 100\%$ or $G = 1$, only superpeers participate in the system, and the hierarchical system is reduced to a flat DHT.

5.5.2 Total Signaling Traffic

Our definition of the optimal operating point of a given system implies minimizing the total costs C, i.e. the total signaling traffic that is generated during the operation of the P2P system, while not overloading any peer in the system. C can easily be calculated and expressed in terms of the superpeer ratio α. According to Section 5.4.4, any normal peer $\mathcal{N}$ in the system sends/receives

$$c_{\mathcal{N}} = l_{\mathcal{N}} + m_{\mathcal{N}} + r_{\mathcal{N}} \tag{5.32}$$

messages, while any superpeer $\mathcal{S}$ sends/receives

$$c_{\mathcal{S}} = l_{\mathcal{S}} + m_{\mathcal{S}(\mathrm{TL})} + m_{\mathcal{S}} + r_{\mathcal{S}} \tag{5.33}$$

messages. The total signaling traffic C is the accumulated number of messages of all normal peers and all superpeers:

$$C = \sum_{\mathcal{N}} c_{\mathcal{N}} + \sum_{\mathcal{S}} c_{\mathcal{S}} \tag{5.34}$$

Table 5.2: Expressions for lookup, maintenance and republish traffic of normal peers and superpeers as functions of α.

	Fully-meshed intra-group structure	Single-connection intra-group structure	DHT intra-group structure
$l_{\mathcal{N}}$	$q_{\mathcal{N}} + \dfrac{Q}{N}$	$q_{\mathcal{N}} + \dfrac{Q}{N}$	$q_{\mathcal{N}} + \dfrac{Q}{N} \cdot (1 - {}^{1}\!/_{2} \log \alpha)$
$m_{\mathcal{N}}$	$\dfrac{1-\alpha}{\alpha \cdot T_{\text{PING}}}$	$\dfrac{1}{T_{\text{PING}}}$	$\dfrac{3}{T_{\text{STAB}}} - \dfrac{\log \alpha \cdot 2}{T_{\text{FIX}}}$
$r_{\mathcal{N}}$	$\dfrac{f_{\mathcal{N}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot N}$	$\dfrac{f_{\mathcal{N}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot N}$	$\dfrac{f_{\mathcal{N}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot N} \cdot (1 - {}^{1}\!/_{2} \log \alpha)$
$l_{\mathcal{S}}$	$q_{\mathcal{S}} + \dfrac{Q}{\alpha \cdot N} \cdot (\log \alpha N + \alpha + 2)$	$q_{\mathcal{S}} + \dfrac{Q}{\alpha \cdot N} \cdot (\log \alpha N + \alpha + 2)$	$q_{\mathcal{S}} + \dfrac{Q}{\alpha \cdot N} \cdot (\log \alpha N - {}^{1}\!/_{2} \log \alpha + \alpha + 2)$
$m_{\mathcal{S}(\text{TL})}$	$\dfrac{3}{T_{\text{STAB}}} + \dfrac{\log \alpha N \cdot 2}{T_{\text{FIX}}}$	$\dfrac{3}{T_{\text{STAB}}} + \dfrac{\log \alpha N \cdot 2}{T_{\text{FIX}}}$	$\dfrac{3}{T_{\text{STAB}}} + \dfrac{\log \alpha N \cdot 2}{T_{\text{FIX}}}$
$m_{\mathcal{S}}$	$\dfrac{1-\alpha}{\alpha \cdot T_{\text{PING}}}$	$\dfrac{1-\alpha}{\alpha \cdot T_{\text{PING}}}$	$\dfrac{3}{T_{\text{STAB}}} - \dfrac{\log \alpha \cdot 2}{T_{\text{FIX}}}$
$r_{\mathcal{S}}$	$\dfrac{f_{\mathcal{S}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot \alpha \cdot N} \cdot (\log \alpha N + \alpha + 2)$	$\dfrac{f_{\mathcal{S}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot \alpha \cdot N} \cdot (\log \alpha N + \alpha + 2)$	$\dfrac{f_{\mathcal{S}}}{T_{\text{REP}}} + \dfrac{F}{T_{\text{REP}} \cdot \alpha \cdot N} \cdot (\log \alpha N - {}^{1}\!/_{2} \log \alpha + \alpha + 2)$

Equation 5.34, together with the expressions from Table 5.2, allows for calculating C as a function of α for any system containing N peers, given their individual lookup rate $q_{\mathcal{P}}$ and their individual number of shared data items $f_{\mathcal{P}}$. Note that the periodic timers T_{PING}, T_{STAB}, T_{FIX} and T_{REP} have globally predefined values. In the following we introduce multiple experimental parameter settings with realistic values for N, $q_{\mathcal{P}}$ and $f_{\mathcal{P}}$, on which we base our conclusions in this and particularly later sections. The parameter settings are listed in Table 5.3. For each parameter, we specify three different values (low – medium – high). We consider small P2P systems with 100 peers, medium sized systems with 10000 peers and large systems with 1000000 peers. Further, we vary the lookup rate between one lookup every 5 minutes, one lookup every minute and one lookup every 10 seconds. Finally, the number of shared data items per peer in our experiments is 5, 50 and 500, respectively. In all experiments, we use the following realistic timer values: $T_{\text{PING}} = T_{\text{STAB}} = 5$ s, $T_{\text{FIX}} = 30$ s and $T_{\text{REP}} = 300$ s.

Table 5.3: Experimental parameter values.

Parameter	Assigned values
Total number of peers	100; 10000; 1000000
Lookup rate	1/300 s^{-1}; 1/60 s^{-1}; 1/10 s^{-1}
Number of shared data items	5; 50; 500

We determine the total signaling traffic C, depending on the superpeer ratio α, in every experimental setting that can be obtained from combining the above parameter values (27 in total). Figure 5.3 shows the result of an example scenario with $N = 10000$ peers, where we assign a lookup rate $q = 1/60\ \text{s}^{-1}$ and a number of shared data items $f = 50$ to every peer. Although we show the results of a specific example scenario here, we emphasize that the results that are obtained from all other parameter combinations show qualitatively the same behavior.

Our experiments show that for each particular superpeer ratio α_0 the SiCo architecture generates the lowest total signaling traffic, i.e.

$$C_{\text{SiCo}}(\alpha_0) \leq C_{\text{FuMe}}(\alpha_0) \quad \forall\ \alpha_0\ \in\ ]0; 0.25] \tag{5.35}$$

$$C_{\text{SiCo}}(\alpha_0) \leq C_{\text{2DHT}}(\alpha_0) \quad \forall\ \alpha_0\ \in\ ]0; 0.25] \tag{5.36}$$

The reason is that the single-connection intra-group structure implements an economic group maintenance algorithm, and does not generate additional signaling traffic to resolve the responsible peer within a group (cf. step 4 in the lookup procedure). In the special case $\alpha \rightarrow 0$ where only one peer acts as a superpeer, the system equals a Napster-like centralized P2P system. Clearly, such a system architecture generates the lowest possible total signaling traffic, because the query for a shared data item in the system is reduced to a lookup in a central data base. For increasing values of α, the top-layer DHT contains more and more superpeers. This results in an increasing signaling traffic in the top layer, and therefore in an increasing total signaling traffic.

The situation is different in the 2DHT architecture. Both special cases "all peers are superpeers" ($\alpha = 100\%$) and "only one superpeer" ($\alpha \rightarrow 0$) correspond to a flat

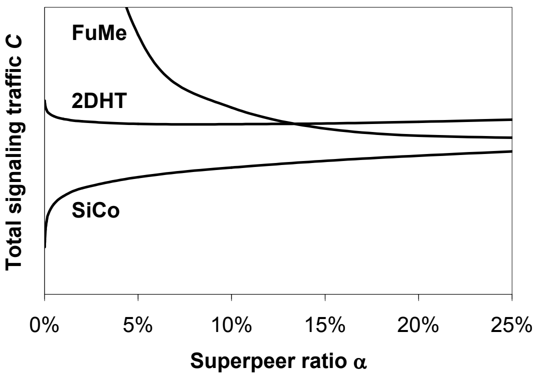


Figure 5.3: Total signaling traffic.

DHT where all peers participate in. Consequently, the total signaling traffic for $\alpha \to 0$ approaches the same total signaling traffic as for $\alpha = 100\%$. Between these two extremes there exists a value of α where the system generates the minimal total signaling traffic that is possible with this system architecture.

In the FuMe architecture, we notice a very high total signaling traffic for low superpeer ratios. Low values for α correspond to large group sizes, which generate an extreme amount of maintenance traffic due to the Ping/Pong traffic among group peers. We again notice a value of α between the two extremes $\alpha \to 0$ and $\alpha = 100\%$ giving rise to the lowest possible total signaling traffic in this system architecture. Note that this value of α is not visible in Figure 5.3 as its value is higher than 25% in the given scenario.

It is important to emphasize that we cannot yet conclude from the above results that the SiCo architecture is best. The reason is that we have not considered the individual load that the superpeers take, i.e. the load distribution in the system as a whole. The next section introduces load levels as an additional class of parameters on which we base our optimality considerations.

5.5.3 Load Level of Superpeers

With regard to the total signaling traffic that is generated during the operation of the considered systems we observe that the SiCo architecture, in combination with a low number of superpeers (ideally one), is optimal. However, most of the arising signaling traffic in our analyzed systems is handled by superpeers, as they participate in the top-layer DHT and at the same time manage their group of normal peers. For low values of α (corresponding to a large group size managed by every single superpeer), the traffic load of superpeers may exceed their capacity. As a result, superpeers are overloaded and we face the risk of breaking system stability. On the other hand, a sufficiently large value of α ensures that the traffic load of superpeers is shared among a sufficient number of superpeers.

To determine the minimal necessary number of superpeers to avoid overload, taking into account their individual capacities, we define a load level λ for every peer. The load

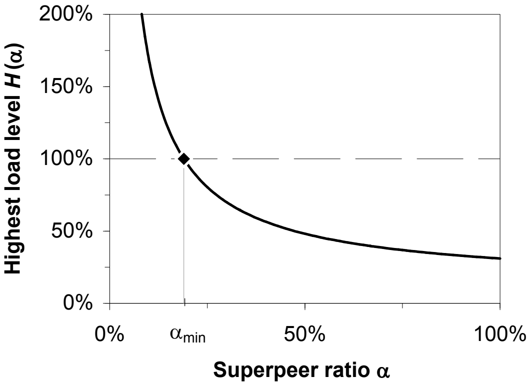


Figure 5.4: Highest load level against α in a homogeneous scenario.

level of peer $\mathcal{P}$ specifies the ratio between the traffic load $c_{\mathcal{P}}$ of $\mathcal{P}$ at a specific time, and the maximum traffic load that $\mathcal{P}$ is able to process, i.e. the load limit $c_{\mathcal{P}}^{\max}$ of $\mathcal{P}$:

$$\lambda_{\mathcal{P}}(\alpha) = \frac{c_{\mathcal{P}}(\alpha)}{c_{\mathcal{P}}^{\max}} \cdot 100\% \tag{5.37}$$

As superpeers process most of the generated signaling traffic, we focus on the load levels of superpeers in the following. We say a superpeer is overloaded if its load level exceeds 100%. A state of the system is acceptable only if no superpeer is overloaded, i.e., if all load levels are below 100%. Therefore, we focus on the highest load level

$$H(\alpha) = \max_{\text{superpeers}} (\lambda_{\mathcal{P}}(\alpha)) \tag{5.38}$$

that can be observed across all superpeers in a given system, and we rely on $H(\alpha)$ to determine acceptable superpeer ratios where the system is operating reliably, i.e. without overloading any superpeer. The following examples illustrate this. They are based on the same example scenario as in the previous section, with $N = 10000$, $q = 1/60\ \text{s}^{-1}$ and $f = 50$. As intra-group structure we apply the single-connection structure. However, the results for the other two intra-group structures are qualitatively the same. First, we consider a homogeneous scenario where all peers have the same capacity. In a second step, we evaluate the differences in a heterogeneous scenario with varying capacities of peers.

Figure 5.4 gives an example of a homogeneous environment. All peers in the system are able to process the same maximum traffic load $c_{\mathcal{P}}^{\max}$. In this particular example we determine $c_{\mathcal{P}}^{\max}$ as follows. First, we assume that all peers have an upload bit rate of 600 kbit/s. We focus on the upload capacity here because this is normally a more limiting factor than download capacity or disk space. Second, we assume an average message size of 500 Bytes. Third, we allow 10% of a peer's upload bit rate to be used for signaling traffic, in order to have enough free upload capacity for user data traffic such as file transfers. As a result, every peer can process

$$c_{\mathcal{P}}^{\max} = \frac{600\ \text{kbit/s} \cdot 10\%}{500\ \text{Bytes}} \cdot \frac{125\ \text{Bytes}}{\text{kbit}} = 15\ \text{s}^{-1} \tag{5.39}$$

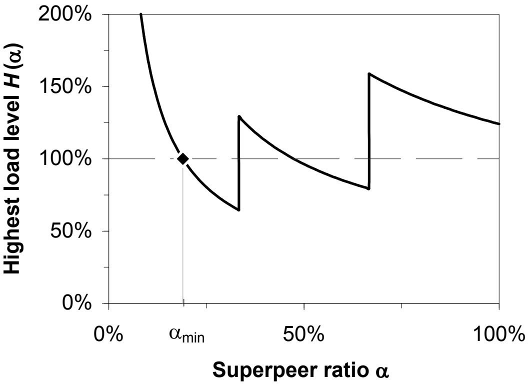


Figure 5.5: Highest load level against α in a heterogeneous scenario.

messages per second. As all peers in the system have this capacity, the highest observable load level $H(\alpha)$ shown in Figure 5.4 is the load level of any superpeer, provided that all superpeers process the same portion of the total signaling traffic. We see that the load level of superpeers drops as the fraction of superpeers grows. The intuition behind is clear: As more peers become superpeers, they share the load in the system, and thus their load level drops. Further, we see that in this example a minimal superpeer ratio of $\alpha_{\min} = 19\%$ is necessary to have enough superpeers in the system so that their load level is below 100%. Put another way, all superpeer ratios smaller than 19% are not acceptable, because they lead to overloaded superpeers.

The situation is slightly different in a heterogeneous environment. When the peers in the system have varying capacities, i.e. varying upload bit rates, we experience jumps in the $H(\alpha)$ curve. Figure 5.5 shows the same example as Figure 5.4, however, we now define three different classes of peers regarding their capacity. One third of all peers has an upload bit rate of 600 kbit/s (class1), one third has 300 kbit/s (class2), and one third has 150 kbit/s (class3). According to equation 5.39 this results in the following peer capacities:

$$c^{\max}_{\mathcal{P},\text{class1}} = 15.00 \text{ s}^{-1} \tag{5.40}$$

$$c^{\max}_{\mathcal{P},\text{class2}} = 7.50 \text{ s}^{-1} \tag{5.41}$$

$$c^{\max}_{\mathcal{P},\text{class3}} = 3.75 \text{ s}^{-1} \tag{5.42}$$

When increasing the superpeer ratio α, we assume that class1 peers are selected as superpeers first. For $1/3 < \alpha \leq 2/3$ we also take class2 peers as superpeers. Finally, for $\alpha > 2/3$ even class3 peers act as superpeers. Obviously, whenever the first peer from a class with a lower capacity acts as a superpeer, the highest observable load level in the system is the load level of this peer and thus $H(\alpha)$ suddenly increases. Between these jumps, $H(\alpha)$ shows the same course as in the homogeneous scenario. Figure 5.5 shows a further interesting result. $H(\alpha)$ exceeds 100% not only for $\alpha < \alpha_{\min} = 19\%$ but also for $1/3 < \alpha \leq 47\%$ and whenever class3 peers act as superpeers ($\alpha > 2/3$). These superpeer ratios also lead to overloaded superpeers and are thus not acceptable. This implies that in the given scenario, a flat Chord system ($\alpha = 100\%$) is not an applicable system architecture and only a hierarchical system architecture provides a stable system operation.

With regard to the above examples we come to the following general definitions. We define A as the set of all acceptable superpeer ratios that avoid overload:

$$A = \{\alpha \in]0;1] \mid H(\alpha) \leq 100\%\} \tag{5.43}$$

and α_{min} as the lowest superpeer ratio that avoids overload:

$$\alpha_{\text{min}} = \min_{\alpha} (A) \tag{5.44}$$

These definitions enable us to define the optimal operating point of any given system. We define it as the superpeer ratio α_{opt} where the total costs C are as small as possible, while at the same time no superpeer in the system is overloaded:

$$\alpha_{\text{opt}} = \operatorname*{argmin}_{\alpha \in A} C(\alpha) \tag{5.45}$$

Note that, in the SiCo architecture, the optimal superpeer ratio α_{opt} is always the minimal necessary ratio α_{min}, as the total signaling traffic is monotonically increasing with α here.

Summing up, the optimal operating point α_{opt} is determined as follows:

1. Plot the total signaling traffic C as a function of the superpeer ratio α.
2. Specify a capacity for every participating peer, and plot the highest observable load level in the system H, also depending on the superpeer ratio α.
3. Exclude all values of α that lead to overloaded superpeers, i.e. $H(\alpha) > 100\%$.
4. Select the superpeer ratio with the least possible total signaling traffic C without overloading any superpeer.

Throughout this work, we use the load level of superpeers to determine the optimal operating point of a system, defined as the superpeer ratio that generates the lowest total signaling traffic without producing load levels of more than 100%. However, the load level of peers can further be viewed as an indicator of the probability that the peer will fail. Consequently, distributions of load levels across peers may be indicative of what we could generally call the quality of the system. We do not attempt here to define the concept of P2P system quality, but we believe that this can be done by extending the analysis of Gummadi *et al.* [GGG$^+$03] by the above ideas.

5.6 Comparison of Analyzed System Architectures

The analytical model presented in the previous sections allows us to determine the optimal operating point of any given system, and to calculate the total signaling traffic that is generated at this specific point. In this section we use this framework to compare the three analyzed hierarchical system architectures. We first compare the FuMe architecture with the SiCo architecture. Then, as the latter turns out to be the preferable solution, we compare the SiCo architecture with the 2DHT architecture. In summary, we show that the SiCo architecture is the overall preferable system architecture in terms of cost efficiency.

5.6.1 FuMe and SiCo Architecture

From Tables 5.1 and 5.2 we see that a superpeer in the FuMe architecture bears the same traffic load $c_{\mathcal{S}}$ as a superpeer in the SiCo architecture. Consequently, the curve of the highest load level $H(\alpha)$ observable across all superpeers is the same in both system architectures, and therefore, also the set A of acceptable superpeer ratios is the same in both cases.

The optimal superpeer ratio of the FuMe architecture $\alpha_{\text{opt(FuMe)}}$ is basically located between $\alpha_{\min}$ and 1, while the optimal superpeer ratio of the SiCo architecture $\alpha_{\text{opt(SiCo)}}$ always equals $\alpha_{\min}$, because the total signaling traffic is monotonically increasing here. According to equation 5.35 in Section 5.5.2, the total costs for $\alpha = \alpha_{\text{opt(FuMe)}}$ are lower in the SiCo architecture:

$$C_{\text{SiCo}}(\alpha_{\text{opt(FuMe)}}) \leq C_{\text{FuMe}}(\alpha_{\text{opt(FuMe)}}) \tag{5.46}$$

With

$$\alpha_{\text{opt(SiCo)}} \leq \alpha_{\text{opt(FuMe)}} \tag{5.47}$$

and

$$C_{\text{SiCo}}(\alpha_1) \leq C_{\text{SiCo}}(\alpha_2) \quad \forall\ \alpha_1 \leq \alpha_2 \tag{5.48}$$

we see that

$$C_{\text{SiCo}}(\alpha_{\text{opt(SiCo)}}) \leq C_{\text{FuMe}}(\alpha_{\text{opt(FuMe)}}) \tag{5.49}$$

Hence, the SiCo architecture is in any case preferable to the FuMe architecture. Figure 5.6 illustrates our reasoning.

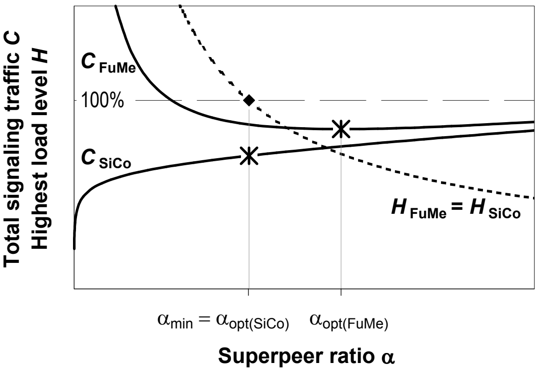


Figure 5.6: Comparison: FuMe vs. SiCo architecture.

5.6.2 SiCo and 2DHT Architecture

To investigate whether the SiCo architecture is also preferable to the 2DHT architecture, we again compare the total signaling traffic at optimal operating points. The key to understand that this is not a trivial problem is the fact that the $H(\alpha)$ curves are different so that we optimize on different ranges of α.

Formally, what we are doing here is constrained optimization. The objective function coincides with the total costs, while the constraint set is defined by the set A of acceptable superpeer ratios, i.e. the requirement that the highest observable load level in the system is below 100%. From this point of view, the previous case (comparison between the FuMe architecture and the SiCo architecture) was almost trivial. There are two key points we could use to derive the conclusion, which is also intuitively clear: The constraint sets coincide, and the curve of the total signaling traffic of the SiCo architecture is always below the curve of the FuMe architecture.

The case "SiCo vs. 2DHT" is different in the sense that the load levels of superpeers do not coincide, and thus the constraint sets of the two optimization problems are different. Therefore, even though the relative positions of the two curves of the total signaling traffic remain the same, we cannot derive the conclusion that the SiCo architecture is always better. This is illustrated in Figure 5.7. It shows a hypothetical scenario where the 2DHT architecture outperforms the SiCo architecture. The reason is that in this example, the set A of acceptable superpeer ratios is larger when a DHT is used to connect group peers ($\alpha_{\min(\text{2DHT})} < \alpha_{\min(\text{SiCo})}$). Therefore, the optimal superpeer ratio in this case is left of the optimal superpeer ratio of the SiCo architecture, which in this example results in a lower total signaling traffic in α_{opt}.

Theoretically, we could now proceed with determining the optimal operating point α_{opt} in both system architectures, for each of the parameter settings from Table 5.3. We could then compare the generated total signaling traffic in both system architectures and check whether one always outperforms the other, or whether there are some parameter settings in which the SiCo architecture is better and some in which the 2DHT architecture is better. However, we proceed in a slightly different manner. For each of the parameter

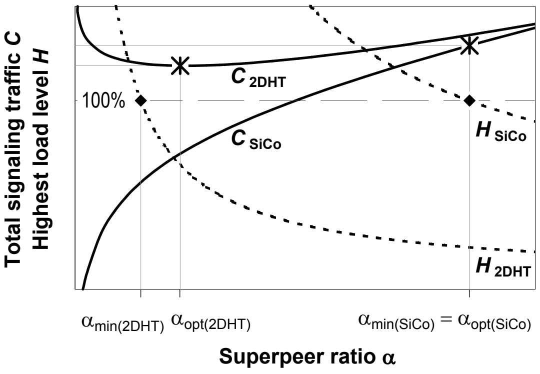


Figure 5.7: Hypothetical scenario where the 2DHT architecture generates a lower total signaling traffic in α_{opt} than the SiCo architecture.

settings we compute the minimal total signaling traffic in the 2DHT architecture as well as the maximal total signaling traffic in the SiCo architecture. This is done on the entire interval $\alpha \in\ ]0; 0.25]$, i.e. without taking load levels and constraint sets defined by them into account. Clearly, the total signaling traffic in the SiCo architecture always reaches its maximum for $\alpha = 0.25$.

It turns out that the maximal total signaling traffic in the SiCo architecture is always lower than the minimal total signaling traffic in the 2DHT architecture. This leads to the conclusion that the SiCo architecture is always better. Although this conclusion is based on empirical evaluations, we believe that there are no realistic parameter settings where a DHT between group peers is preferable according to our optimality criterion. Nevertheless, we have found some rare parameter settings where the constraint set (= the range of acceptable α) in the SiCo architecture is smaller, i.e., the traffic load per superpeer is lower in the 2DHT architecture. These settings are characterized by low lookup rates and a low number of shared data items. This indicates that the PING/PONG traffic of superpeers in the SiCo architecture outweighs the necessary traffic of a superpeer to resolve the responsible group peer when a DHT is applied as intra-group structure. However, also with these parameter settings the SiCo architecture generates the lower total signaling traffic.

5.6.3 Summary

The above comparison of the FuMe architecture, the SiCo architecture and the 2DHT architecture shows that, at least for realistic settings of the system parameters, a single connection between a normal peer and its superpeer is the preferable system architecture for a hierarchical DHT-based system design. The reason is that the SiCo architecture adopts an optimal operating point that generates the lowest total signaling traffic in the system.

There are further advantages of the SiCo architecture in addition to being optimal in terms of total costs. For example, the single-connection intra-group structure allows for an efficient handling of churn. Its star-like structure also supports a shift of the stored references from normal peers to superpeers to further offload traffic from resource-constrained (e.g. mobile) peers. We discuss the influence of both to our model in the following section.

5.7 Model Modifications

In the traffic analysis in Section 5.4 we made several assumptions to set up a formal model for our analysis. In the following section, we change and relax some of these assumptions and investigate the effects on our results.

5.7.1 Impact of Churn

The traffic analysis from Section 5.4 has modeled the usual dynamics of P2P systems only partially. As peers join and leave the system, maintenance algorithms need to be performed to keep neighbor lists consistent, especially with regard to failing peers. These maintenance routines were included in our analysis. However, peer arrivals and departures generate additional signaling traffic:

- Joining peers have to be integrated into the P2P system.
- Leaving peers (that do not fail) have to notify their neighbors to avoid inconsistent neighbor lists.
- References have to be shifted when peers join or leave, in order to maintain the mapping rule of the DHT.

This signaling traffic has to be taken into account in order to precisely compare the considered hierarchical system architectures. In the following we show its effect on our model and the achieved results.

Joining / leaving superpeer. When a superpeer joins or leaves the top-layer DHT, the same routines have to be performed in each system architecture. Thus, we have the same increments of the generated signaling traffic in each of the three system architectures.

Joining normal peer. When a normal peer joins a group at the lower layer, then in case of the single-connection intra-group structure the arriving peer exchanges two messages with the group's superpeer. The arriving peer sends a Connect Request message to notify the superpeer about its presence, and the superpeer acknowledges with a Connect Ack message. Obviously, the traffic increments in the fully-meshed intra-group structure and in the DHT intra-group structure are larger. In the fully-meshed intra-group structure, all other group peers have to be informed about the arriving peer. When a DHT is applied to connect group peers, the arriving peer has to be integrated into the lower layer DHT.

The additional signaling traffic inside the group that is caused by shifting references to the arriving peer is the same in each of the three system architectures.

Leaving normal peer. When a normal peer leaves its group, it notifies its neighbors. The corresponding signaling traffic is lowest in the single-connection intra-group structure, because only the group's superpeer has to be notified. In case of the DHT intra-group structure, the predecessor and the successor of the leaving peer are notified, while in the fully-meshed intra-group structure, all peers in the group have to be contacted.

Again, the additional signaling traffic by shifting references from the leaving peer to the peer that becomes responsible for them is the same in each of the three system architectures.

Summing up, we see that the signaling traffic that is generated by churn is either the same in each system architecture or lowest in the SiCo architecture. We conclude that the only way churn affects the curves of the total signaling traffic and the highest load level $H(\alpha)$ is that they move further apart than without churn – their relative positions remain the same. Recalling our conclusion from Section 5.6.2 that for all parameter settings the SiCo architecture generates the lowest total signaling traffic in its optimal operating point, and combining the reasoning we just presented, we conclude that churn does not have a qualitative impact on the results of our analysis and comparison presented earlier. Thus, the SiCo architecture remains most preferable.

5.7.2 Recursive Routing

There are two main reasons for considering an iterative routing scheme in Section 5.4. First, the focus of our work is on heterogeneous environments where some peers are connected through unreliable wireless links. In such scenarios, iterative routing benefits from the fact that the peer that initiates the lookup keeps track of the routing process and can immediately react to peer failures. In contrast, recursive routing may become unreliable in case of frequent peer failures during the DHT lookup, because of the unacknowledged forwarding of the lookup request. Second, recursive routing is considered to be more vulnerable to attacks, as a peer can generate multiple subsequent messages by initiating a single lookup (cf. Section 2.4.1). Nevertheless, in the following we investigate how changing the type of routing from iterative to recursive effects our model and the obtained results.

Recall the lookup procedure described in Section 5.4.1. When recursive routing is used to resolve the responsible superpeer $\mathcal{S}_2$ in the top-layer DHT (step 2), $^1\!/\!_2 \log N_{\text{SP}}$ messages are generated on average to forward the request to the preceding superpeer. This superpeer then sends a response containing its successor (= the responsible superpeer $\mathcal{S}_2$) back to the initiating superpeer $\mathcal{S}_1$. On average, $^1\!/\!_2 \log N_{\text{SP}} + 1$ messages are thus generated in the top-layer DHT on every lookup. If we assume that every superpeer processes an equal proportion of all Q initiated lookups and that the top-layer routing of all Q initiated lookups involves all superpeers equally, we obtain the following equations for $l_{\mathcal{S}_1}$, $l_{\mathcal{S}}$ and $l_{\mathcal{S}_2}$:

$$l_{\mathcal{S}_1} = \frac{Q}{N_{\text{SP}}} \tag{5.50}$$

$$l_{\mathcal{S}} = \frac{Q}{N_{\text{SP}}} \cdot (^1\!/\!_2 \log N_{\text{SP}} + 1) \tag{5.51}$$

$$l_{\mathcal{S}_2} = \frac{Q}{N_{\text{SP}}} + l_{\mathcal{S}_2,\text{add}} \tag{5.52}$$

When a DHT is applied as intra-group structure, also the resolution of the responsible group peer $\mathcal{R}$ (step 4) is different. Using recursive routing, it generates an average number of $^1\!/\!_2 \log G + 1$ messages in the lower layer DHT on every lookup. If we assume that all G group peers are involved equally in the resolution of all Q/N_{SP} lookups of their group, the total number of messages of any group peer, i.e. of the superpeer $\mathcal{S}_2$ and of each normal peer $\mathcal{N}$, to resolve $\mathcal{R}$ is given by

$$l_{\mathcal{S}_2,\text{add(2DHT)}} = l_{\mathcal{N},\text{add(2DHT)}} = \frac{Q}{N_{\text{SP}} \cdot G} \cdot (^1\!/\!_2 \log G + 1) \tag{5.53}$$

With the above changes we now calculate the modified equations for the lookup traffic, assuming a recursive routing scheme. We also modify the equations for the republish traffic in the same way. After integrating these changes into our model we recalculate all expressions for lookup, maintenance and republish traffic of normal peers and superpeers. This again allows us to determine the optimal operating point of any given system and to compare the different hierarchical system architectures. Note that the FuMe architecture and the SiCo architecture are influenced similarly by changing the top-layer DHT routing scheme. Thus, there is no qualitative change to the comparison in Section 5.6.1, so we focus on the comparison of the SiCo architecture and the 2DHT architecture in the following.

By performing the same experiments as presented in Section 5.5, we find that recursive routing reduces the traffic load of superpeers when a DHT is used to connect group peers. The reason is that intra-group lookups are now initiated by the group's superpeer, but afterwards resolved by the group peers without involving the group's superpeer. This traffic reduction leads to a decreased load level of superpeers, resulting in a larger set A of acceptable superpeer ratios. However, when we again compare the total signaling traffic in the optimal operating point in each system architecture, we still find no parameter setting where the 2DHT architecture is preferable to the SiCo architecture. Hence, we conclude that our results obtained in Section 5.6 are not influenced by the applied routing scheme.

5.7.3 Storage of Data References

For our comparison we assume that references to shared data items are stored on all peers in the system, on superpeers and on normal peers. The SiCo architecture, however, allows as an alternative for storing references only on superpeers. This leads to a decreased path length and a lower signaling traffic when performing lookups. Additionally, provided that powerful peers with a long session duration and low failure probability are selected as superpeers, the system benefits from the advantages that we have shown in Chapter 4. There we have found that the option of storing references only on reliable superpeers (we named them "static peers" in Chapter 4) unloads normal peers ("temporary peers") with constrained resources, decreases the maintenance traffic generated by shifting references in the presence of churn, and increases the availability of provided content.

Note that the above advantages can be achieved only with the SiCo architecture. The FuMe architecture and the 2DHT architecture both make sense only if references are stored also on normal peers. In this way, the above advantages lead to an even more superior performance of the SiCo architecture.

5.8 Conclusion

Hierarchical P2P systems provide many advantages in comparison to flat systems. In our view, they are the right choice for P2P systems that are applied in real-life heterogeneous environments, characterized by a varying reliability and capacity among the participating peers. In this chapter we have analyzed hierarchical DHT-based P2P systems with a top-layer DHT (Chord) between superpeers, and different intra-group structures to organize the normal peers at the lower hierarchy layer. In particular, we have studied a fully-meshed intra-group structure, a single-connection intra-group structure with connections only between the normal peers and their superpeer, and a DHT as a further possibility to connect group peers.

We have developed a formal analytical model, deriving the operation costs in terms of the total signaling traffic and the individual traffic load of superpeers in each system architecture. Both quantities have been expressed as functions of the fraction of superpeers in the system. This has enabled us to determine the optimal operating point of each system architecture, i.e. the optimal fraction of superpeers where the total operation costs are as low as possible while at the same time no superpeer is overloaded. Our analytical evaluation, based on our analytical model and backed up by numerical experiments, has shown that a star-like intra-group structure, where normal peers are connected through single connections to their superpeer, is superior to a fully-meshed intra-group structure or a DHT to organize the peers at the lower layer. As a result, we focus on this single-connection intra-group structure in the following chapters. Additionally, the single-connection intra-group structure offers the possibility to store references only on superpeers. To benefit from the corresponding advantages (see Section 5.7.3), we adopt this alternative for the rest of this thesis.

Our analytical model allows us to determine the optimal operating point of a given system. It is based on the global view on the relevant system parameters such as the number of peers, their capacity, lookup rate and number of shared data items. However, if we want to apply those theoretical results in a practical P2P system to automatically achieve and dynamically maintain such cost-optimal operation, we have no global knowledge about the system status. Instead, we have to apply algorithms that rely on the partial view of every single peer on the system. In the following, we present distributed algorithms that solve this problem. In Chapter 6 we develop a load balancing algorithm for the considered hierarchical system architecture to achieve a fair distribution of normal peers (we denote them “leafnodes” in the following as we focus on the single-connection intra-group structure) across superpeers. This load balancing is an essential prerequisite for the algorithm we present in Chapter 7 that allows for achieving and maintaining cost-optimal operation of our hierarchical system in a distributed manner.

6 Fair Leafnode Distribution

6.1 Motivation

In the previous chapter we have developed an analytical cost model to determine optimal configurations of hierarchical DHT-based P2P systems in terms of the fraction of superpeers in the system. Based on this model we have shown that the system design from Figure 6.1 is optimal in the sense that its optimal configuration generates the lowest total costs. The depicted system architecture divides the participating peers into two classes: superpeers ("$\mathcal{S}$") and leafnodes ("$\mathcal{L}$"). Superpeers are organized in a top-layer Chord ring and serve as proxies to their leafnodes. Whenever a leafnode looks up or inserts a shared data item, it calculates the data item's hash key k and sends a request to its superpeer. The superpeer resolves the responsible superpeer for the request in the top-layer DHT and forwards the request to this superpeer. The responsible superpeer finally sends the requested key-value-pair (in case of a lookup) or an acknowledgment (in case of an insert) back to the leafnode. Note that we assume here that references to shared data items are stored only on the superpeers. The reason is that our results from Chapter 4 show a superior system behavior when references are stored only on reliable peers, while the results from Chapter 5 are thereby not affected (cf. argumentation in Section 5.7.3).

Our analysis in Chapter 5 illustrates how the hierarchical system architecture from Figure 6.1 presents a trade-off between a centralized (only one superpeer) and a fully decentralized (all peers are superpeers) system architecture. From the point of view of the generated signaling traffic, the best system architecture is the centralized one. The problem is, clearly, that no peer in the system can act as a central index server and bear all the load. At the other extreme, the fully decentralized system architecture, a flat DHT among the participating peers normally generates unnecessary signaling traffic.

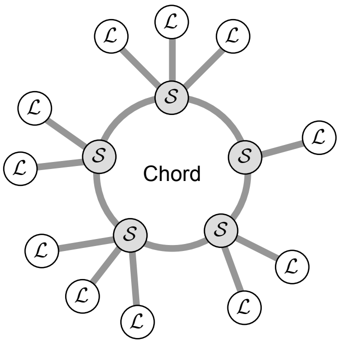


Figure 6.1: Hierarchical two-tier system architecture.

We therefore propose the system architecture from Figure 6.1 as an optimal trade-off between these two extreme cases.

A main prerequisite for the proper operation of the proposed system is a fair distribution of leafnodes across superpeers. Just as in a flat DHT, such load balancing is needed because unbalanced load can cause overloaded peers to fail. This holds especially true for the superpeers in our hierarchical system architecture. If a superpeer fails due to its high load, all its leafnodes rejoin the system and are assigned to other superpeers. In the worst case, this again results in overloaded (and thus failing) superpeers if no appropriate load balancing algorithm is used. Finally, the whole system may crash.

The hierarchical system architecture offers another dimension to balance load, i.e. the assignment of a joining leafnode to an appropriate superpeer. In fact, this is where our work differs substantially from state-of-the-art solutions to DHT load balancing. In this chapter, we exploit this possibility and present a novel load balancing algorithm for the considered hierarchical two-tier P2P system. However, we emphasize that the presented algorithm can easily be extended to systems with more than two layers, and be applied whenever peers from a lower layer must be balanced across peers in a higher layer. We perform simulations based on realistic load definitions that show that our algorithm provides a superior load balancing performance, while at the same time it generates less overhead than conventional algorithms.

The remainder of this chapter is structured as follows. Section 6.2 discusses related work. In Section 6.3 we introduce our load balancing algorithm. Sections 6.4 and 6.5 evaluate the load balancing performance, while Section 6.6 analyzes the generated overhead. In Section 6.7 we discuss possible changes in the assumptions we make. Section 6.8 concludes.

6.2 Related Work

Many algorithms have been proposed so far that deal with load balancing in DHTs. Normally they refer to the term "load" as the amount of keys or the amount of data items that a peer is responsible for. The goal of such load balancing algorithms is to assign to every participating peer an equally-sized partition of either the identifier space or of all data items available in the system.

The concept of virtual servers [RLS$^+$03] is based on managing multiple partitions of the identifier space at every participating peer. As a result, one peer represents multiple "virtual servers", where each of them acts as an independent node in the DHT. Those virtual servers can be shifted from heavy-loaded to light-loaded peers. This coincides with one leave and one join event in the DHT.

The algorithm introduced in [BCM03] is based on the "power of two choices" paradigm, proposed by Mitzenmacher *et al.* in [MRS00]. Whenever a data item is inserted into the DHT, n different hash functions are used to calculate n different keys $\{k_1 \dots k_\mathrm{n}\}$ of the data item, with $n \geq 2$. Then, the responsible peers for the n calculated keys are contacted in parallel, and a reference to the data item is stored on the peer with the currently lowest load. In Sections 6.4 and 6.5, we use this algorithm for comparison with our load balancing algorithm.

In [RPW04] Rieche *et al.* present an algorithm that balances load in a DHT similar to the process of heat dispersion. Every interval of the identifier space is managed by a minimum number of n and a maximum number of $2n$ peers. Each of these peers stores references to all shared data items that are mapped onto this interval. Load balancing is performed in three different ways. First, if an interval is managed by $2n$ peers, and the peers are heavy-loaded, the interval is halved. The halves are then assigned to the peers $1 \dots n$ and the peers $n+1 \dots 2n$, respectively. As a result, the involved peers loose half of their load. Second, if a light-loaded interval is managed by more than n but less than $2n$ peers, some peers of that interval can be moved to other, heavy-loaded intervals, which in turn can be split according to the above procedure. Third, if an interval is managed by no more than n peers, the borders of the interval may be shifted so that the load is balanced between the peers of the neighboring intervals.

Karger and Ruhl [KR04] propose algorithms for balancing the distribution of the identifier space and of shared data items, respectively. For identifier space balancing they assign multiple positions of the identifier space (so-called "virtual nodes") to every peer, but choose only one of those virtual nodes to become active at a time. From time to time, each peer determines its virtual node that manages the smallest partition of the identifier space, and activates that virtual node. For data item balancing they occasionally compare the load of peer $\mathcal{P}_{\mathrm{i}}$ with the load of another, randomly chosen peer $\mathcal{P}_{\mathrm{j}}$. If load balancing between $\mathcal{P}_{\mathrm{i}}$ and $\mathcal{P}_{\mathrm{j}}$ is necessary, $\mathcal{P}_{\mathrm{j}}$ changes its position in the identifier space so that it gets located between $\mathcal{P}_{\mathrm{i}}$ and its predecessor, and therefore takes half of the data items for which $\mathcal{P}_{\mathrm{i}}$ has been responsible so far.

Kenthapadi and Manku [KM05] try to balance the load in a DHT by dividing the identifier space into equally sized partitions. Their goal is to minimize the ratio σ between the largest and the smallest size of a partition. Therefore, they focus on joining peers and, on every join, select r random points in the DHT, for which they inspect the sizes of v partitions in proximity of each random point. Then the joining peer splits the largest partition encountered. The authors show that for any r and v satisfying $r \cdot v \geq c \cdot \log k$, where c is a small constant and $k-1$ is the number of peers in the DHT before the k^{th} peer joins, the ratio σ is at most 8, with high probability.

There are two important remarks regarding the above solutions that we make here to point out the main differences and advantages of our solution. First, the definition of load in the cited papers is – from our point of view – inappropriate to a certain extent. The amount of keys or the amount of data items that a peer is responsible for is used to define a peer's load. Although our load balancing algorithm could easily be adapted to this load definition, we rely on more realistic definitions. In a first step, where we consider homogeneous environments characterized by superpeers with similar capacities and leafnodes that generate a similar load, we define load as the number of leafnodes managed by one superpeer. Afterwards, we focus on the more general case of heterogeneous environments, and define load as the traffic load of peers, i.e. the number of messages peers forward during system operation.

Second, all of the above algorithms focus on a flat DHT design, i.e. a DHT design in which all peers are functionally the same. Consequently, they all opt to balance load by selecting how many data items are in any peer's responsibility. The decision on this is made either at the insertion phase or during the normal operation of the system. Our system architecture is, in contrast, not flat but hierarchical. Therefore, we have one more possibility to balance load. As leafnodes join the system, we can assign them to different superpeers according to the current load distribution among superpeers. This is the main difference between our solution and those presented above. Note that our algorithm is orthogonal to the existing ones. It can work together with any of them, rather than competing with them.

6.3 A Distributed Algorithm for Fair Leafnode Distribution

In this section we present our load balancing algorithm for the proposed hierarchical DHT-based P2P system. More precisely, we balance the load that is generated in the superpeers' top-layer DHT, thereby generating an acceptable overhead. To evaluate our algorithm and to compare it to state-of-the-art solutions, we consider the following scenario, which is expected to be the worst case scenario for load balancing. All peers that join the system first contact one predefined superpeer in the top-layer DHT. Note that this "bootstrap superpeer" is merely an entry point to the system, and has no global knowledge about the load distribution across all superpeers. Then, a distributed load balancing algorithm is used to determine to which superpeer the joining peer is connected. Our goal is to assign an equal load to every superpeer. In the following we describe our algorithm and present two other algorithms that we use for comparison in Section 6.4, Section 6.5 and Section 6.6.

6.3.1 The "Piggyback" Algorithm

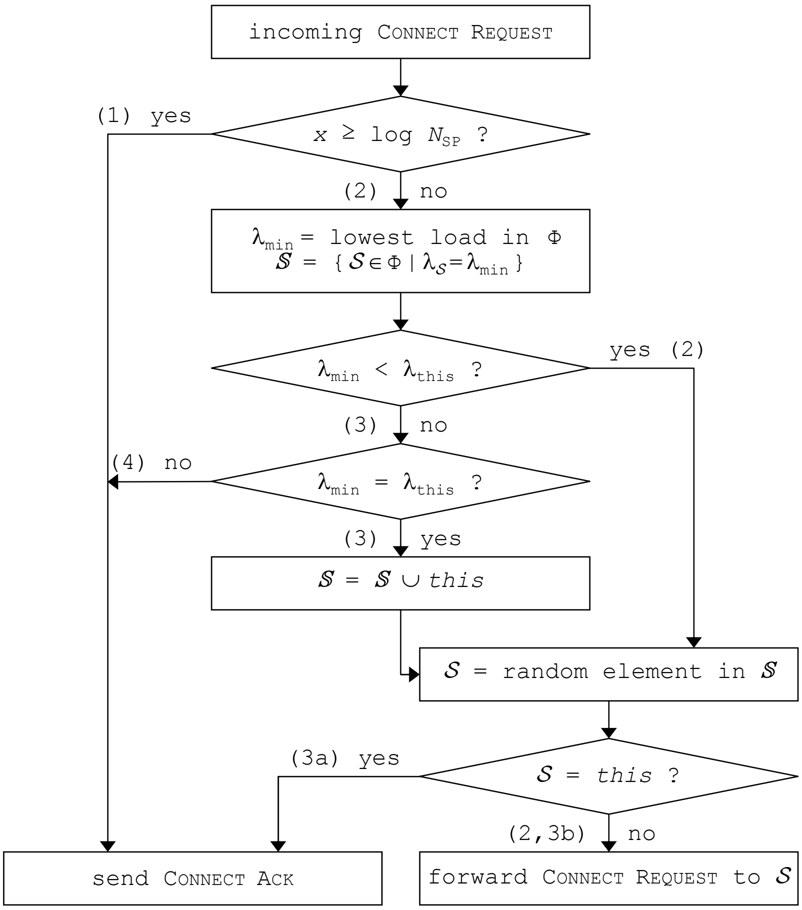
The "piggyback" algorithm we present below is named after the requirement that every message exchanged between two superpeers includes the current load of the sending superpeer as piggyback information. The receiving superpeer stores this information in a $\log N_{\mathrm{SP}}$ sized FIFO array, where N_{SP} is the number of superpeers in the top-layer DHT[1]. As a result, every superpeer knows the load of the $\log N_{\mathrm{SP}}$ superpeers that recently sent messages to it. We choose $\log N_{\mathrm{SP}}$ as the size of the FIFO array here so that it has the same dimension as the routing table. This ensures a logarithmic growth of the FIFO array size in terms of N_{SP}.

When the bootstrap superpeer or any subsequently contacted superpeer receives a Connect Request message, it runs the "piggyback" algorithm. The basic principle of this load balancing algorithm is depicted in Figure 6.2:

1. If the number of hops that the connection request has been forwarded is $\log N_{\mathrm{SP}}$, the superpeer accepts the connection by sending a Connect Ack message to the joining peer. We define $\log N_{\mathrm{SP}}$ as an upper bound for the number of forwardings because $\log n$ is the theoretical maximum number of hops in a DHT with n peers

[1] In Section 6.7 we discuss the problem of estimating N_{SP} in a real-world system.

this:	this superpeer
Φ:	FIFO array containing $\langle \mathcal{S}, \lambda_{\mathcal{S}} \rangle$ tuples
$\mathcal{S}$:	superpeer
$\lambda_{\mathcal{S}}$:	load of superpeer $\mathcal{S}$
x:	number of forwardings of CONNECT REQUEST message



incoming CONNECT REQUEST
$x \geq \log N_{SP}$?
(1) yes
(2) no
λ_{min} = lowest load in Φ
$\mathbb{S} = \{ \mathcal{S} \in \Phi \mid \lambda_{\mathcal{S}} = \lambda_{min} \}$
$\lambda_{min} < \lambda_{this}$?
yes (2)
(3) no
(4) no
$\lambda_{min} = \lambda_{this}$?
(3) yes
$\mathbb{S} = \mathbb{S} \cup this$
$\mathcal{S}$ = random element in $\mathbb{S}$
(3a) yes
$\mathcal{S} = this$?
(2,3b) no
send CONNECT ACK
forward CONNECT REQUEST to $\mathcal{S}$

Figure 6.2: "Piggyback" algorithm.

to reach any other peer in the system (provided that the identifier space is fully populated).

2. If the connection request has been forwarded less than $\log N_{\mathrm{SP}}$ times, the superpeer selects that superpeer from its FIFO array that has the lowest load. Occasionally, multiple superpeers in the FIFO array may have the same lowest load. If this load is lower than the load of the superpeer itself, it forwards the connection request to the superpeer with the lowest load. If there are more than one superpeer having the lowest load, it chooses randomly between one of those superpeers.

3. If the lowest load in the FIFO array is not lower but equal to the own load, the superpeer chooses randomly either (a) to accept the connection, or (b) to forward the connection request to one of those superpeers with equal load.

4. If there is no superpeer in the FIFO array that has a lower or equal load, the superpeer accepts the connection by sending a Connect Ack message to the joining peer.

In addition to this fundamental functionality of our “piggyback” algorithm we propose the following two modifications to further improve the load balancing performance. Note that for purposes of clarity, these modifications have not been included in Figure 6.2.

- To avoid loops in case of inconsistent FIFO arrays, a connection request is only forwarded to superpeers that have not already been contacted during the join procedure. We therefore include the already visited superpeers in every forwarded Connect Request message.
- When a connection request is forwarded for the last (= the $\log N_{\mathrm{SP}}$-th) time, the forwarding superpeer $\mathcal{F}$ deletes the destination address of the final hop $\mathcal{D}$ from its FIFO array. The reason is that $\mathcal{F}$ knows beforehand that $\mathcal{D}$ must accept the connection (cf. step 1), and thus the information about $\mathcal{D}$'s load that is stored on $\mathcal{F}$ is not valid any more.

6.3.2 Algorithms for Comparison

To be able to rank the performance of the algorithm presented above, we study two further load balancing algorithms. We refer to them as the “random assignment” algorithm and the “power of two choices” algorithm, respectively.

The “random assignment” algorithm can be seen as the simplest way of balancing load in our hierarchical system. When a peer joins the system, it sends a connection request to the bootstrap superpeer. The bootstrap superpeer then chooses a randomly selected ID and initiates a lookup of the responsible superpeer in the top-layer DHT. After determining the responsible superpeer, the bootstrap superpeer returns its address to the joining peer, which now connects to the chosen superpeer.

The “power of two choices” algorithm uses a similar approach to balance the load in our system. The only difference is that it chooses not one but two random IDs and determines the responsible superpeers. Then the superpeer with the currently lower load is selected. This “power of two choices” is often referred to in the literature as a good solution to the “balls into bins” problem. In our specific scenario, the joining peer sends a connection request to the bootstrap superpeer, which then looks up the responsible superpeers for two randomly selected IDs and checks their current load. Then the bootstrap superpeer returns the address of the superpeer with the lower load to the joining peer, and the connection is established.

6.4 Balancing Leafnode Connections

In this section, we define the term “load” of a superpeer as the number of leafnode connections that the superpeer maintains. Thus, the goal of our load balancing algorithm is to assign approximately the same number of leafnodes to every superpeer, i.e., to achieve equal group sizes at the lower layer of our hierarchical system architecture. This definition of load fits to scenarios where the superpeers show homogeneous capacities, and the load that is generated by every single leafnode is equal to some extent. In Section 6.5 we consider a more general definition of load that is particularly suited for highly heterogeneous scenarios.

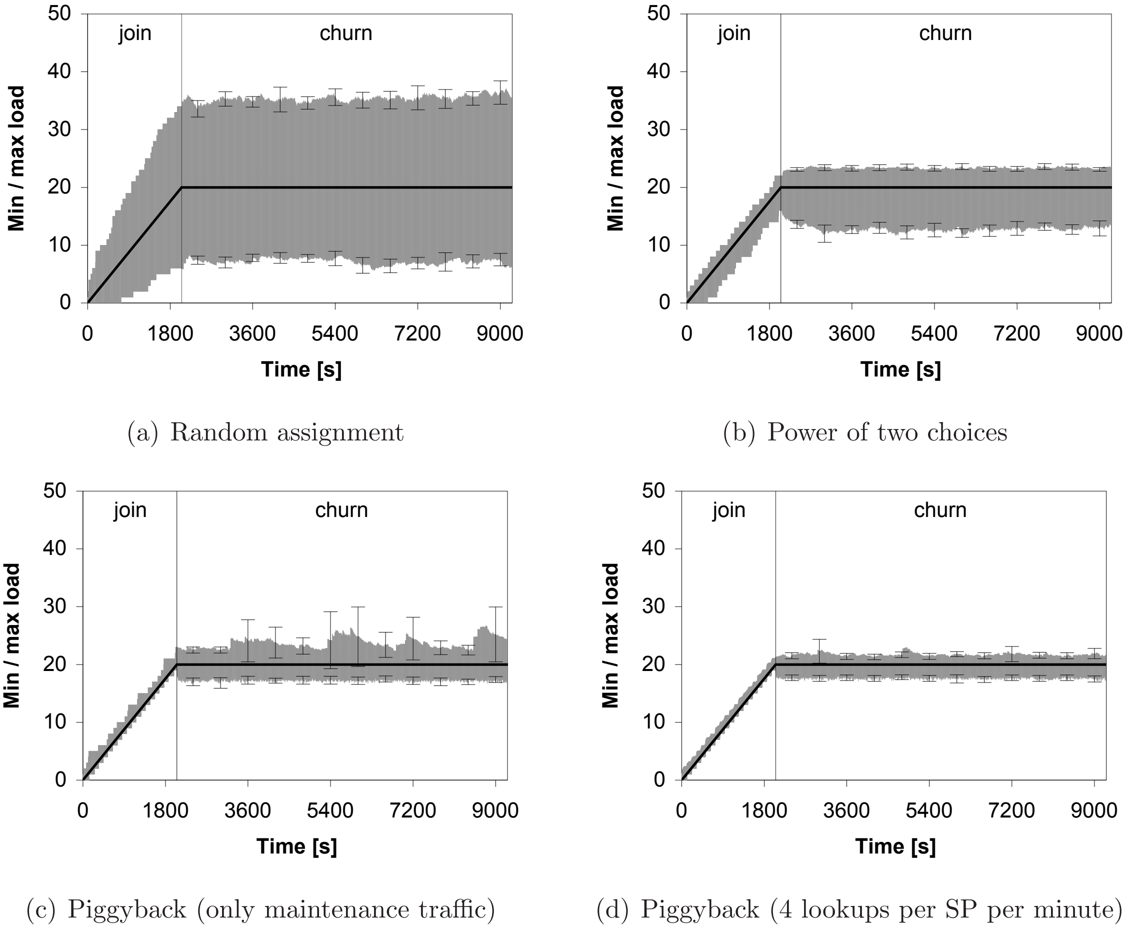
Based on the following experimental scenario, we evaluate the load balancing performance of our "piggyback" algorithm, and compare it to the "random assignment" and the "power of two choices" algorithm, respectively. We create a Chord ring with $N_{\text{SP}} = 2^{10} = 1024$ superpeers, and let $G = 20 \cdot 2^{10} = 20480$ leafnodes successively join the system. Every joining leafnode sends the initial Connect Request message to the superpeer with ID 0, i.e. the bootstrap superpeer. The arrival rate is set to 10 peers per second. As a result, after 2048 seconds all leafnodes have joined the system, and in case of perfect load balancing every superpeer should manage 20 leafnode connections. We refer to this first part of our experiment as the "join phase". In the second part, which we call the "churn phase", we randomly choose 10 leafnodes per second to leave the system and immediately rejoin via the bootstrap superpeer. We set the duration of the churn phase to 2 hours. After every finished join procedure, we measure the current minimum and maximum load that can be observed across all superpeers, as well as the standard deviation from the current mean value.

To have a statistical validation of the achieved results, we perform 10 independent simulation runs with different seeds for the random number generator. In the following figures, the average values of the 10 simulation runs are depicted. Additionally, during the churn phase, we measure the 99% confidence intervals of the minimum and maximum load and of the standard deviation every 600 seconds.

Figure 6.3(a) shows the minimum and maximum number of leafnode connections when we apply the "random assignment" algorithm. In the join phase, it takes 733 seconds before at least one leafnode is assigned to every superpeer. At the end of the join phase, the minimum and maximum loads are 6 and 34, respectively. During the churn phase, the situation is even worse. Here the number of leafnode connections varies between a minimum of 6 and a maximum of 37. Consequently, we state that this algorithm is not a good approach to balance load between superpeers.

As expected, the load balancing performance of the "power of two choices" algorithm is significantly better. Figure 6.3(b) shows a time span of 459 seconds to assign at least one leafnode to every superpeer. After completing the join phase, loads have a minimum value of 16 and a maximum value of 22. In the course of the churn phase, the algorithm shows a slightly worse load balancing performance than in the join phase. Here, the number of leafnode connections varies between 12 and 24.

When we apply the "piggyback" algorithm, the load balancing performance depends on the number of messages that are exchanged between superpeers. The more messages are sent, the more information about the current load of superpeers is distributed among them, and hence better load balancing is achieved. Figure 6.3(c) shows the results of a scenario where Chord's periodic maintenance messages, generated by the Stabilize and FixFingers algorithms (cf. Section 5.4.2), are used to distribute load information among superpeers. We see that piggybacking load information solely on these maintenance messages already leads to a good load balancing. In our specific case (we set $T_{\text{STAB}} = 5$ s



(a) Random assignment

(b) Power of two choices

(c) Piggyback (only maintenance traffic)

(d) Piggyback (4 lookups per SP per minute)

Figure 6.3: Minimum/maximum load (number of leafnode connections).

and $T_{\text{FIX}} = 30$ s), it takes only 138 seconds to assign at least one leafnode to every superpeer. Further, the minimum and maximum number of leafnode connections at the end of the join phase are 19 and 23, respectively. During the churn phase, we observe an overall minimum load of 17 and a maximum load of 27 leafnode connections. However, we sometimes also measure a relatively high confidence interval, indicating that – from time to time – the load information is outdated and a superpeer gets assigned more leafnodes than the mean value.

This temporary "unfairness" disappears if also lookup messages are exchanged between superpeers, which is the normal mode of operation of our hierarchical P2P system. For our experiment, we specify an average rate of 1 lookup every 5 minutes for each of the 20480 leafnodes in the system. In the hierarchical system, every lookup by a leafnode is resolved by its superpeer. Thus, every superpeer performs 4 DHT lookups per minute for its leafnodes on average. In Figure 6.3(d), we see the impact of this (comparatively low) lookup traffic on our experiment. The minimum and maximum number of leafnode connections in the system are even closer to the mean value of 20 than without lookup traffic, and small confidence intervals prove the excellent load balancing performance of our "piggyback" algorithm across all simulation runs.

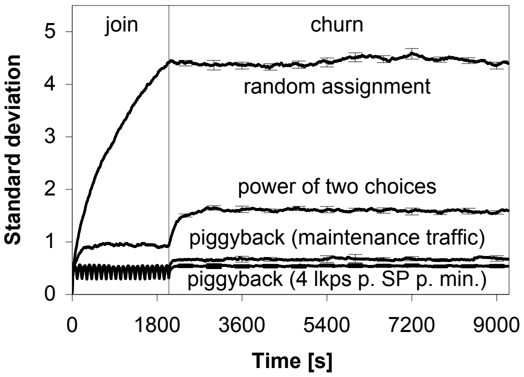


Figure 6.4: Standard deviation of load (number of leafnode connections).

As a second evaluation criterion, we compare the mean standard deviation of the number of leafnode connections in Figure 6.4. The standard deviation of the "random assignment" algorithm rises quickly during the join phase and stays at a continuously high value of approximately 4.5 during the churn phase. This backs our claim above that "random assignment" is not an applicable load balancing algorithm. The "power of two choices" algorithm shows a relatively low standard deviation of about 0.9 during the join phase. Interestingly, in the churn phase we notice an increased standard deviation of around 1.6. We see directly from Figure 6.4 that our "piggyback" algorithm offers the best load balancing performance, even when only maintenance traffic is present in the superpeers' top-layer DHT. In this case, the standard deviation fluctuates around an average value of 0.7, while it is even further decreased to a relatively constant value of 0.5 when there are 4 lookups from every superpeer per minute.

Another result from our experiments that is not shown in the figures is that the mean number of forwardings of a connection request (and therefore the generated overhead and the time until a joining peer is connected to a superpeer) decreases with an increasing signaling traffic in the system. Clearly, the "piggyback" algorithm benefits from a more up-to-date information about the current load of other superpeers in this case.

6.5 Balancing Traffic Load

In heterogeneous environments the capacities of superpeers as well as the load that is generated by leafnodes may vary significantly. As a result, the number of assigned leafnodes may become an inappropriate definition for the load of a superpeer. If a superpeer has a higher capacity, it should also manage more leafnodes. To reflect this heterogeneity of peers in our load balancing problem, we consider another definition of load in this section, and evaluate its effect on the performance of the analyzed load balancing algorithms.

In Section 5.5.3 we have introduced the load level of a superpeer as a quantity to measure its traffic load. The load level $\lambda_{\mathcal{S}}$ of superpeer $\mathcal{S}$ has been defined as the ratio between the traffic load $c_\mathcal{S}$ of $\mathcal{S}$ at a specific time, and the maximum traffic load $c_\mathcal{S}^{\max}$ that $\mathcal{S}$ is able to process:

$$\lambda_\mathcal{S} = \frac{c_\mathcal{S}}{c_\mathcal{S}^{\max}} \cdot 100\% \tag{6.1}$$

In the following we use this definition of load to evaluate our “piggyback” algorithm and to compare it to the presented alternatives. Clearly, the goal of each load balancing algorithm now is to ensure equal load levels across all superpeers.

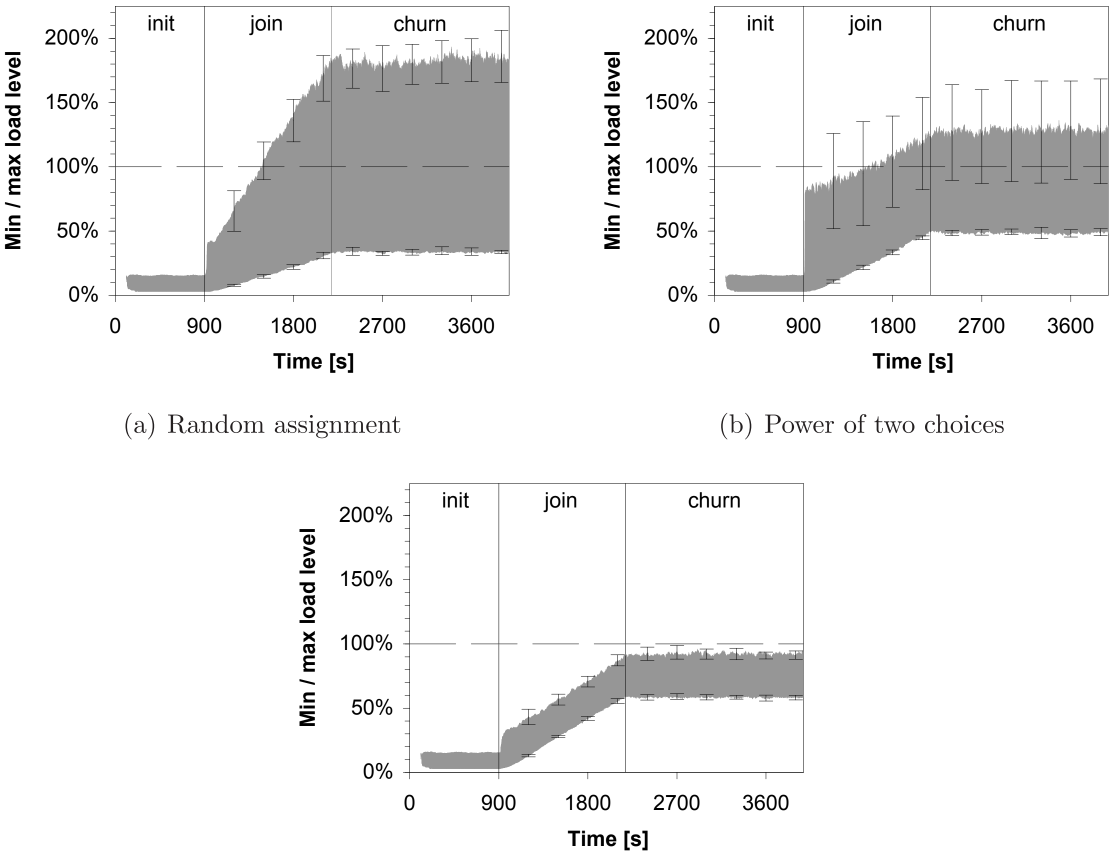
Below, we show by simulation that the “piggyback” algorithm is also the most preferable load balancing algorithm for the above definition of load. In our particular scenario, we again focus on the upload capacity of superpeers and specify the load limit $c_\mathcal{S}^{\max}$ as the maximum number of messages per second that a superpeer can upload. We vary the upload bit rate of superpeers between 1 Mbit/s and 3 Mbit/s, and allow 10% thereof for signaling traffic. Therefore, a superpeer provides an available upload bit rate between 100 kbit/s and 300 kbit/s. We assume an average message size of 500 Bytes, so that a superpeer can upload between 25 and 75 messages per second.

To determine a superpeer's load level $\lambda_\mathcal{S}$ we measure the superpeer's current number of uploaded messages per second and divide it by its assigned load limit. This, however, raises another problem. The number of uploaded messages can vary over time, although the system conditions stay constant. This may be caused e.g. by a bursty lookup traffic. To ensure a good load balancing performance, we require an algorithm that flattens the currently measured load of a superpeer, i.e. a low-pass filter, but at the same time reacts fast enough to changing system conditions and therefore to a changing load of the superpeer. By performing multiple experiments we have found that a simple sliding-window algorithm can meet both requirements. Upon every 10^{th} uploaded message, a superpeer determines the time interval T_{500} for sending the previous 500 messages. Then it calculates its current load level $\lambda_\mathcal{S}$ by

$$\lambda_\mathcal{S} = \frac{500}{T_{500} \cdot c_\mathcal{S}^{\max}} \cdot 100\% \tag{6.2}$$

We set up a Chord ring with $N_{\text{SP}} = 2^8 = 256$ superpeers and let $N_{\text{LN}} = 15 \cdot 2^8 = 3840$ leafnodes successively join the system. We specify an individual load limit $c_\mathcal{S}^{\max} \in [25; 75]$ for every superpeer $\mathcal{S}$ (cf. above). Further, we assign a randomly selected lookup rate $q_\mathcal{P} \in [1/900\ \text{s}^{-1}; 1/60\ \text{s}^{-1}]$ and a number of shared data items $f_\mathcal{P} \in [10; 100]$ to every peer $\mathcal{P}$, to superpeers as well as to leafnodes. After 900 seconds of simulation time (which allows the superpeers to measure their initial load level, without any load from a leafnode) we connect 3 peers per second to the system, all joining the system by initially contacting the bootstrap superpeer. As a result, the join phase takes 1280 seconds. Finally, during the churn phase (duration is 1800 seconds) we select randomly 3 leafnodes per second to leave and immediately rejoin the system. To evaluate the load balancing performance, we continuously measure the load levels of all superpeers and determine their standard deviation. Moreover, we record the minimum and maximum load level observable across all superpeers.

As in the previous section, we perform 10 independent simulation runs with each load balancing algorithm. The figures below show the average values of the 10 simulation runs, and additionally the 99% confidence intervals measured every 300 seconds.



(a) Random assignment

(b) Power of two choices

(c) Piggyback

Figure 6.5: Minimum and maximum load levels.

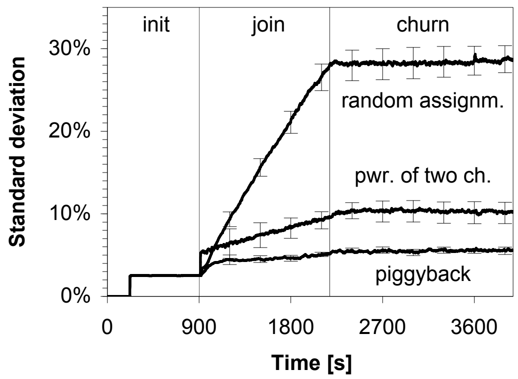


Figure 6.6: Standard deviation of load levels.

Figures 6.5(a), 6.5(b) and 6.5(c) show the minimum and maximum load levels that we measure across all superpeers over time. Further, in Figure 6.6 the mean standard deviation is depicted for each algorithm. Again we see that the "random assignment" algorithm is not a suitable solution to our load balancing problem. The highest observable load level reaches up to 188% and also the minimum deviates heavily from the mean value. Its standard deviation rises quickly during the join phase and then remains at a high value of about 28%. In contrast, the "power of two choices" algorithm offers a standard deviation that is around 10%. However, the maximum observable load level increases quickly during the join phase and – even worse – takes on values of more than 100% during the churn phase. The only algorithm that can avoid load levels above 100% in this particular simulation scenario is the "piggyback" algorithm. Due to the deterministic assignment of joining leafnodes to superpeers with a low load level it can efficiently avoid overloaded superpeers. Moreover, it again offers the lowest standard deviation which is around 5%.

Another result of our simulations that is not shown in the diagrams is that the minimum and maximum observable load levels of the "piggyback" algorithm are mainly caused by a temporary increased or decreased traffic load of superpeers. The reason is the random nature of lookup paths. However, these deviations change quickly over time, and the load level of an affected superpeer falls quickly back to a value close to the mean value.

6.6 Overhead

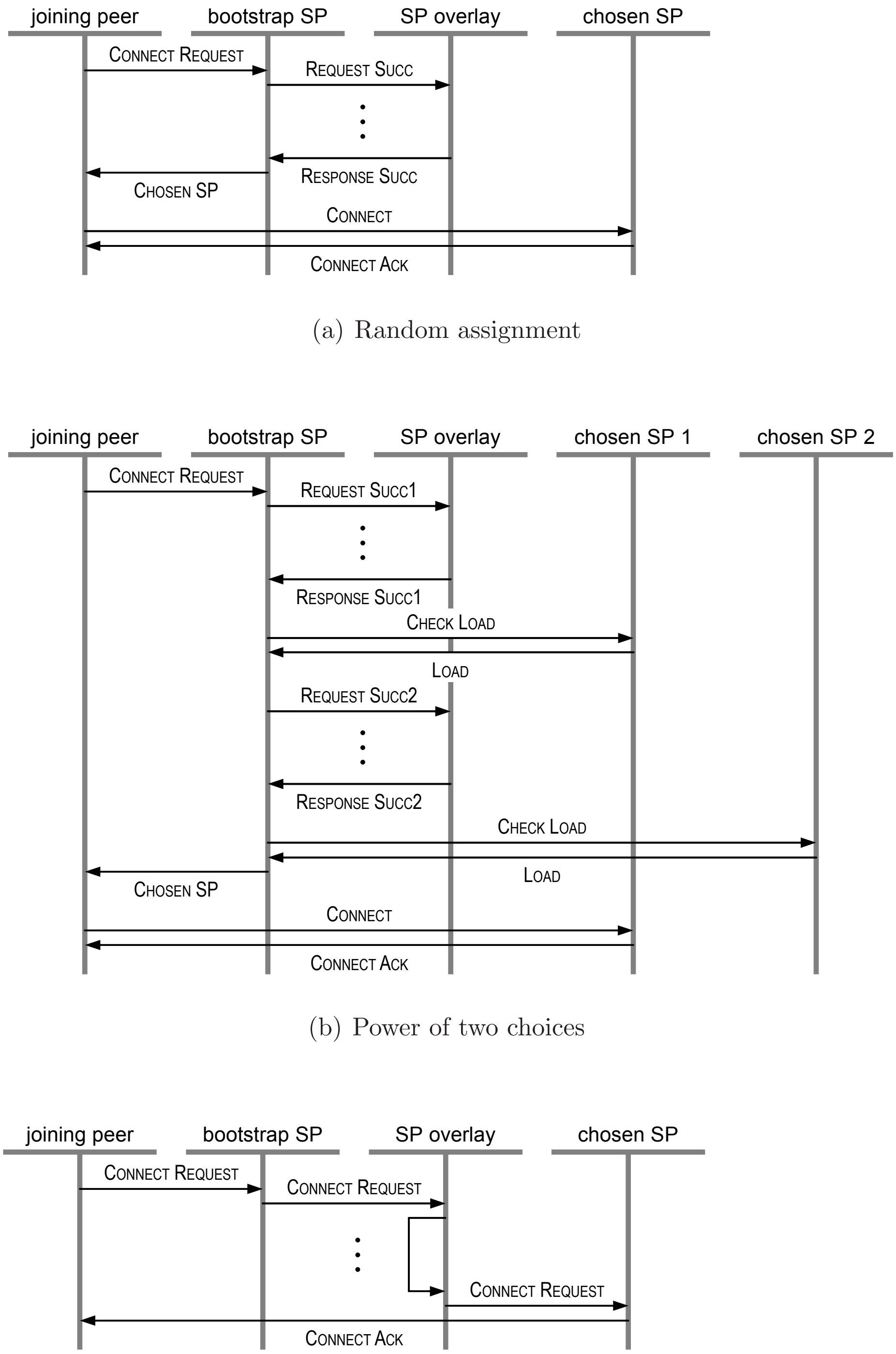
All of the compared load balancing algorithms generate signaling traffic for the involved superpeers. Certainly, we have to take this overhead into account when comparing the algorithms. We therefore calculate the number of messages that are generated on every join procedure. Figures 6.7(a), 6.7(b) and 6.7(c) show the message sequence charts of a join procedure for each algorithm. Independent from the applied algorithm, a join procedure always requires a Connect Request message from the joining peer to the bootstrap superpeer, and a final Connect Ack message from that superpeer that accepts the connection back to the joining peer. This adds an overhead of two generated messages to every join procedure. In the following we evaluate how much additional signaling traffic the load balancing algorithms produce.

The "random assignment" algorithm (Figure 6.7(a)) requires $1/2 \log N_{\mathrm{SP}}$ routing hops on average to determine the responsible superpeer for a randomly selected ID. As we assume iterative routing every hop requires one Request Successor message and one Response Successor message. Afterwards, the address of the responsible superpeer is sent from the bootstrap superpeer back to the joining peer. Finally, before the connection can be acknowledged (cf. above), the joining peer must send a Connect message. Altogether, the "random assignment" algorithm generates the mean overhead of

$$O_{\mathrm{mean(random\ \ assignment)}} = \log N_{\mathrm{SP}} + 4 \tag{6.3}$$

messages on every join procedure.

When we apply the "power of two choices" (Figure 6.7(b)) algorithm for load balancing, we have twice as much signaling traffic as above to find the responsible superpeers for two



(a) Random assignment

(b) Power of two choices

(c) Piggyback

Figure 6.7: Message sequence charts of a join procedure.

randomly chosen IDs. Additionally, before sending the address of the chosen superpeer back to the joining peer (cf. above), the bootstrap superpeer must check the current load of the two superpeers first. This requires two Check Load messages and two responses that contain the current load of each superpeer. Altogether, the "power of two choices" algorithm generates the mean overhead of

$$O_{\text{mean(power of two choices)}} = 2 \cdot \log N_{\text{SP}} + 8 \tag{6.4}$$

messages on every join procedure.

The overhead of the "piggyback" (Figure 6.7(c)) algorithm is generated by forwarding connection requests until the matching superpeer is found. As stated above, two messages are required for the initial Connect Request and the final Connect Ack. Additionally, assuming a maximum of $\log N_{\text{SP}}$ forwardings, we obtain an upper bound of $\log N_{\text{SP}}$ for the number of forwarded Connect Request messages. Altogether, the "piggyback" algorithm generates the maximum overhead of

$$O_{\text{max(piggyback)}} = \log N_{\text{SP}} + 2 \tag{6.5}$$

messages on every join procedure.

From the above analysis we see that our "piggyback" algorithm generates the lowest overhead in the system in terms of generated messages:

$$O_{\text{max(piggyback)}} < O_{\text{mean(random assignment)}} < O_{\text{mean(power of two choices)}} \tag{6.6}$$

However, the "piggyback" algorithm generates additional overhead by including the current load of a superpeer in every message sent to another superpeer. To find out the dimension of this overhead, we first determine the average number of messages per second that a superpeer extends with its current load. Then we multiply this value with the size of the information field (in bit) that contains the superpeer's load. In the following we do not present exhaustive simulation results. Instead, we give a representative sample of a realistic P2P system to estimate this additional overhead. We consider a P2P system with N peers that are organized according to our hierarchical system architecture (cf. Figure 6.1). We assume that peers have an average lookup rate of $q = 1$ lookup per minute and an average number of shared data items of $f = 100$. We further assume that $\alpha = 10\%$ of all peers serve as superpeers, and use the following timer values: $T_{\text{PING}} = T_{\text{STAB}} = 5$ s, $T_{\text{FIX}} = 30$ s and $T_{\text{REP}} = 300$ s. Finally, we assume that an 8 bit field in every message that is exchanged between superpeers is used for piggybacking load information. In the following we rely on the traffic analysis from Section 5.4, and we check which of the analyzed messages contain piggybacked load information.

When a lookup is performed or a shared data item is republished in our hierarchical system, a superpeer can piggyback its load information in those messages that are required for resolving the responsible superpeer $\mathcal{S}_2$ (step 2 in Section 5.4.1), and when forwarding the request to $\mathcal{S}_2$ (step 3). Assuming that every superpeer processes an equal

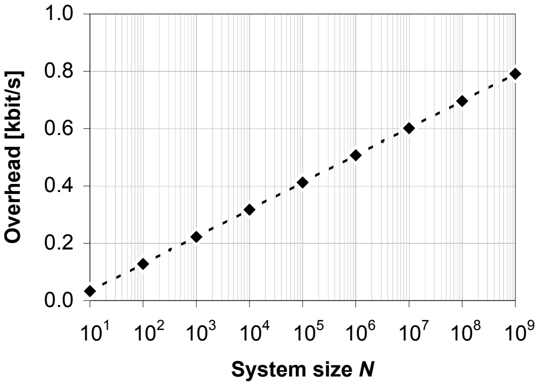


Figure 6.8: Overhead of the "piggyback" algorithm generated by including load information in messages between superpeers.

proportion of this traffic, the number of messages that a superpeer extends with its current load is given by

$$l_{\text{piggybacked}} = \frac{Q}{N_{\text{SP}}} \cdot (\log N_{\text{SP}} + 1) \tag{6.7}$$

$$r_{\text{piggybacked}} = \frac{F}{T_{\text{REP}} \cdot N_{\text{SP}}} \cdot (\log N_{\text{SP}} + 1) \tag{6.8}$$

Also the maintenance messages in the superpeers' top-layer Chord ring contain piggybacked load information. In Section 5.4.2 we have seen that every superpeer exchanges (and thus piggybacks)

$$m_{\text{piggybacked}} = \frac{3}{T_{\text{STAB}}} + \frac{\log N_{\text{SP}} \cdot 2}{T_{\text{FIX}}} \tag{6.9}$$

messages per second for top-layer DHT maintenance.

In total, the number of messages $c_{\text{piggybacked}}$ that a superpeer extends with its current load is the sum of $l_{\text{piggybacked}}$, $m_{\text{piggybacked}}$ and $r_{\text{piggybacked}}$. The overhead that is thereby generated for the superpeer is given by $c_{\text{piggybacked}} \cdot 8$ bit. With the aforementioned parameter settings, we calculate the overhead per superpeer that is depicted in Figure 6.8, depending on the system size N. We notice two important results. First, the overhead increases only logarithmically with the number of peers N. Second, the amount of additional signaling traffic for every superpeer is below 1 kbit/s, even in very large systems with 10^9 peers. Consequently, we state that the overhead generated by extending messages with the current load of a superpeer is negligible for realistic P2P systems. Moreover, in case of high a signaling traffic in the superpeers' DHT, this overhead can further be reduced by including the current load only in every x^{th} message, with $x > 1$. The simulation results in Section 6.4 support this claim, showing that already the maintenance traffic is sufficient to distribute load information efficiently among the superpeers.

Recapitulating the above analysis, we see that the "piggyback" algorithm requires the minimal necessary number of messages on every join event, while its additional overhead for including a superpeer's load as piggyback information in messages exchanged between superpeers is negligible. As a result, not only its load balancing performance is superior to the "random assignment" algorithm and the "power of two choices" algorithm, but also the generated overhead.

6.7 Changing Assumptions

Below, we briefly discuss the assumptions we have made in the previous sections and how the "piggyback" algorithm is affected if we change them. We do not discuss the effects on the "random assignment" algorithm and the "power of two choices" algorithm, as the previous sections have shown that the "piggyback" algorithm is in any case preferable.

In our simulations we can easily limit the number of forwardings of a connection request to $\log N_{\mathrm{SP}}$ as we exactly know the number of superpeers N_{SP} in the system. Certainly, in a real-world system, N_{SP} has to be estimated. However, this estimation can be done e.g. by analyzing the IDs of the superpeers in the routing table, as proposed in [BSH05]. In the following chapter, we have a closer look on this estimation algorithm. Due to the uncertainty of such an estimation we recommend to multiply the estimated value of N_{SP} with an "uncertainty factor", to limit the number of forwardings to a value of at least $\log N_{\mathrm{SP}}$.

Further, we assumed a static top-layer DHT without joining or leaving superpeers. However, when a new superpeer joins the system, either directly or as a promoted leafnode, it initially starts with a load level $\lambda_{\mathcal{S}} \to 0$, as it initially does not manage any leafnodes. We have seen that our algorithm efficiently assigns joining leafnodes to superpeers with a low load level. Therefore, all joining leafnodes are now connected preferably to the new superpeer, until its load level $\lambda_{\mathcal{S}}$ reaches a value close to the average load level of all superpeers. A leaving superpeer does not affect load balancing in the superpeers' DHT. The reason is that all leafnodes that have been connected to the leaving superpeer have to rejoin the system, and a rejoin is similar to the usual join of a leafnode, for which our algorithm has shown good load balancing properties in the previous sections.

Another assumption that we have made was that every leafnode that joins the system initially contacts a bootstrap superpeer. Nevertheless, this assumption has a minor effect on our algorithm. The superpeers that are stored in the FIFO array of the bootstrap superpeer change on every incoming message. Thus, the possible destinations of the second hop of a connection request also vary frequently over time. As a result, there is hardly any difference between initially contacting one bootstrap superpeer or choosing a randomly selected superpeer in the system.

6.8 Conclusion

In this chapter we have proposed a load balancing algorithm targeting hierarchical DHT-based P2P systems. To the best of our knowledge, this class of DHTs is not covered in the state-of-the-art so far. In particular, we have focused on a hierarchical system in which leafnodes are attached to superpeers that are organized in a DHT. We have proposed an algorithm to balance the load among superpeers. Our algorithm is orthogonal to existing load balancing algorithms and can work together with any of them.

We have compared the proposed algorithm to two hypothetical algorithms, the "random assignment" algorithm and an algorithm exploiting the "power of two choices". Our evaluations have targeted a homogeneous scenario, in which we balance the number of connected leafnodes per superpeer, as well as a heterogeneous scenario, where we take different capacities of superpeers and different loads in terms of the number of generated messages into account. The evaluations have shown a superior performance of the proposed algorithm. We have also shown that our algorithm poses less overhead in comparison with the other solutions.

As already mentioned at the end of Chapter 5, our goal is to develop a set of distributed algorithms that enables us to operate our hierarchical system in its optimal operating point, i.e. to dynamically achieve and maintain an optimal superpeer ratio in a distributed manner. The presented load balancing algorithm is the first component of this set of algorithms. In the following chapter, we complete this set with two further distributed algorithms that finally enable us to achieve a cost-optimal operation of the considered hierarchical system.

7 Cost-Optimal System Operation

7.1 Motivation

In the previous chapters we have shown that hierarchical P2P systems are the right choice for P2P systems in real-life heterogeneous communication environments. In particular, we have considered a two-tier DHT-based P2P system with a top-layer DHT (Chord) between superpeers and leafnodes at the lower layer that are connected only to their superpeer. We have developed an analytical cost model that allows us to determine the optimal operating point, i.e. the optimal superpeer-to-leafnode ratio, of any given set of peers. The basic idea of this model is to minimize the number of superpeers in order to reduce the total signaling traffic as far as possible, but at the same time have enough superpeers so that none of them is overloaded.

Chapter 5 shows how to determine the optimal operating point from the relevant system parameters such as the number of peers, their capacity, lookup rate and number of shared data items with a global view on the system. However, in a real-world P2P system we have no global knowledge about the system status. Hence, if we want to apply the theoretical results from Chapter 5 to achieve and maintain a cost-optimal operation of the considered system, we have to rely on the partial view of every single peer on the system. In this chapter we present a set of distributed algorithms that solve this problem. They allow for achieving and maintaining a cost-optimal operation of our hierarchical DHT, while all decisions taken by the peers are based on their partial view on a set of system parameters, describing the current system status.

Having such a dynamic cost-optimization in place provides us with a number of advantages in addition to the general advantages of hierarchical P2P systems that have previously been mentioned in this thesis. For the operator of a communication network hosting such a hierarchical P2P system, costs can be reduced to a minimum by dynamically adapting to the current situation. Note that we are focusing on the total costs of operation from a network operator's viewpoint who is willing to minimize its expense while keeping the system in a stable operation mode. This is especially necessary when traffic is not charged any longer by data consumption but on a flat rate basis as currently emerging even in mobile networks. There are other approaches, e.g. [LSMK05], that are not focusing on a reduction of the total costs of the P2P system but assume a certain available data rate at each peer. That data rate can be used in favorable situations to allow faster lookups by adapting the routing table at the costs of additional maintenance traffic, filling the available data rate at each peer. Our approach is not only advantageous for the operator, but also the user benefits from a high quality DHT, as our dynamic maintenance algorithm puts as many high-layer superpeers in the system as needed to avoid critical, i.e. overload situations. A further advantage for operators and users is the self organization paradigm implemented by our solution. The operator does not have any extra maintenance traffic while being able to keep the system in a stable operation mode and at the same time saving costs. This is also true for the user who does not have to worry if his device becoming a superpeer might break down due to overload.

The feasibility of our solution depends on two prerequisites: load balancing, to have a fair distribution of load among superpeers, and proper estimations of global DHT parameters. Again both algorithms have to be performed in a distributed way. The applied load balancing algorithm has been described in Chapter 6. It extends existing solutions by taking into account the presence of hierarchical layers in hierarchical DHTs, which has not been done before. For the distributed parameter estimation we use slightly adapted algorithms from the literature. No extra signaling traffic is caused here as we are piggybacking estimation information with messages that are exchanged among superpeers.

We prove the feasibility of our solution through extensive simulations with a range of realistic settings, including high churn rates. As we will discuss in detail in the following, we can also observe that the rough estimates of global system parameters calculated by each peer independently are sufficient to come very close to the optimal system configuration that can be calculated with a global view on the system.

The main contributions described in this chapter can be summarized as follows:

- We propose a full set of distributed algorithms including load balancing, system status estimation and maintenance of the optimal operating point.
- In particular, we propose a novel distributed algorithm to achieve and dynamically maintain a cost-optimal operation of the system. The algorithm does not incur any additional signaling traffic as all necessary information exchanged between peers is piggybacked with periodic DHT maintenance messages and lookup requests.
- We demonstrate through extensive simulations that the proposed distributed algorithms, relying only on the local estimations of the relevant system parameters, accurately approximate the optimal system configuration, which can be determined only if the exact values of the parameters are globally known. Our simulations show the benefits of the hierarchical system constructed through our algorithms in comparison with a conventional flat DHT.

The remainder of this chapter is structured as follows. Section 7.2 discusses related work. In Section 7.3 we recapitulate the cost model and define the notion of optimal system configuration. Section 7.4 presents our algorithms for achieving and maintaining an optimal configuration. The algorithms are evaluated in Section 7.5. Section 7.6 concludes.

7.2 Related Work

Even though there are many hierarchical DHT-based system architectures proposed in the literature, most of the works that target specifically the problem of building and configuring hierarchical P2P systems deal with unstructured systems. Yang and Garcia-Molina [YGM03] present a thorough study of superpeer-based unstructured P2P systems.

This work is in a sense very close to ours. Basically, we are led by the same questions: How do we benefit from using a hierarchical system design and, based on that, what design parameters should be selected to increase the benefit as much as possible (e.g., “what is the right number of superpeers?”)? However, there are also a number of important differences between our work and [YGM03]. First, we focus on hierarchical DHTs while they study unstructured systems. Second, our modeling is different in that we precisely quantify the benefits of hierarchical systems within a well-defined analytical cost model. This enables us to decide exactly on optimal system configurations and propose a full set of algorithms that brings our system to the most preferable operating state. On the other hand, [YGM03] provides a set of “rules of thumb” for successful system operation, without specifying a precise algorithm how to build a system.

Montresor’s goal in [Mon04] is basically similar to ours - find the optimal number of superpeers in a given system. He proposes an algorithm called SG-1 in which superpeers exchange information about their capacities with randomly selected other superpeers through a gossip protocol, and try to push their leafnodes toward the most powerful discovered superpeers. SG-1 sets up an unstructured P2P system among superpeers. Thus, it cannot determine the currently optimal configuration (as our algorithm can due to the superpeers’ DHT, which allows calculation of the generated signaling traffic), but must rely on gossiping and thus on the information a peer obtains from randomly chosen other peers. To find these other peers in a constantly changing environment, SG-1 requires the maintenance of an underlying neighborhood set (the so-called connected set) containing randomly chosen peers, which adds overhead to the system. Based on this neighborhood set, superpeers can find another (randomly selected) superpeer that is currently online when gossiping. Our algorithm completely avoids such additional overhead. Moreover, SG-1 models the capacity of a peer as the number of leafnodes the peer can handle. Certainly, this allows SG-1 to efficiently determine the number of leafnodes pushed from one superpeer to a more powerful one when indicated. However, we think this modeling is too simple, as the load that leafnodes generate may diverge significantly. In contrast to SG-1, we define capacity as the available upload bit rate of a peer, and model each peer with its individual session duration, lookup rate and number of shared data items.

The SG-2 algorithm proposed in [JMB06] focuses on a proximity-aware superpeer topology that minimizes the latency between leafnodes and superpeers. The round-trip time (RTT) is either measured directly or approximated through a virtual coordinate service that associates every peer with a position in the virtual space. Superpeers broadcast their availability in their area of the virtual coordinate space so that joining leafnodes are able to connect to the most powerful superpeer in their proximity. Again, a superpeer’s capacity is represented by the maximum number of leafnodes it can handle, and uniformly distributed in the range $[1; 500]$ in the simulations. Similar to its ancestor SG-1, the algorithm needs additional signaling traffic but brings considerable advantages for applications that need low RTTs. A future version of our algorithm may also regard proximity measurements for further optimization.

Apart from distributed algorithms, there also exists the possibility to manage the set of superpeers in a centralized way or to have them statically configured. There also exist approaches which suggest that superpeers should be deployed by content distributors or publishers and ISPs [SRBS03]. This enables the provider to influence the peers' behavior to a certain extent by enforcing defined policies at the superpeers. The provider can control the queries in the system or try to exploit topological locality of query results in order to decrease costs. Despite these advantages, this leads in our opinion too far away from the paradigm of a distributed system and is also not flexible enough in the face of load fluctuations that cannot be handled by a static set of superpeers.

There are a number of other works which are relevant to us, although they are not so directly related to our work as the previous ones. For our algorithms to run properly, we need to estimate the current number of superpeers in the system. [MMKG06] presents a solution based on random walk methods. The approach in [Man03] makes use of an appropriate cryptographic hash function to make sure that the peers' IDs are assigned uniformly in the identifier space. The estimation of the total system size is made by measuring the peer density in a part of the identifier space (usually around the ID of the calculating peer). [PKG$^+$05] and [HM03] follow similar approaches.

[BSH05] proposes a Chord-specific algorithm to estimate the size of the P2P system. It first shows that the length of the interval between the ID of an arbitrary peer and the ID of its direct successor on the Chord ring is geometrically distributed with parameter p, which in turn depends on the size of the system. Hence, a peer can use its successor list as a sample set of different realizations of the same random variable and then use the maximum likelihood estimation to get an approximate value of the total number of peers in the system. Our estimation of the system size, presented in Section 7.4.2, is based on this algorithm.

7.3 Determination of Optimal Operating Point

Before introducing the set of distributed algorithms that enables an optimal operation of our hierarchical DHT, in which the top layer is a Chord ring made of superpeers and the lower layer is made of leafnodes attached to the superpeers, we briefly recapitulate all underlying definitions, assumptions and our notion of optimality.

In Chapter 5 we have introduced a formal cost model to judge on the optimality of hierarchical DHTs (flat DHTs are included in the model as a special case). An optimal hierarchical DHT configuration is defined as a function of the total operation costs (in terms of the total signaling traffic) and the individual costs of peers. More specifically, we calculate the number of messages generated during system operation, including lookup traffic, maintenance traffic and republish traffic, and express it as a function of the superpeer ratio α, i.e. the number of top-layer superpeers divided by the total number of peers. The $C(\alpha)$ curve in Figure 7.1 shows an example. This figure corresponds to a hypothetical scenario with a fixed set of peers, i.e. the peer population remains unchanged. At one extreme, for $\alpha = 1/N$, where N is the system size, we have a centralized P2P system that minimizes the total signaling traffic. As the fraction of superpeers increases so does the total signaling traffic. At the other extreme, for $\alpha = 1$ (there are no leafnodes, all peers are superpeers) we have a flat DHT. In this case the total signaling traffic is maximal.

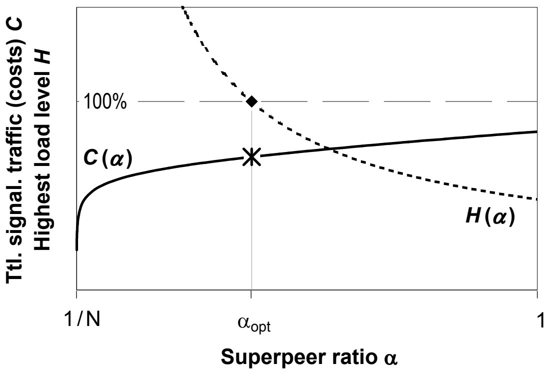


Figure 7.1: Operation costs and highest load level as functions of α.

To determine an optimal value of the superpeer ratio α, we take into account the individual costs of peers. For this purpose, we have defined a load level $\lambda_{\mathcal{P}}$ for every participating peer $\mathcal{P}$. The load level of peer $\mathcal{P}$ is the ratio between the load of $\mathcal{P}$ at a given time instant and the maximum load value peer $\mathcal{P}$ is willing to accept, i.e. the load limit of $\mathcal{P}$. We define the number of uploaded messages as the considered load, i.e., the load level is determined by dividing the actual number of uploaded messages by the upload limit of a peer. This is a reasonable assumption because the upload capacity is normally a more limiting factor than download capacity or disk space. The quantity we call the highest load level H plays an important role in how we determine the optimal system configuration, i.e. the optimal operating point of the system. We have defined H as the highest load level that can be observed across all superpeers:

$$H = \max_{\text{superpeers}} (\lambda_{\mathcal{P}}) \tag{7.1}$$

The $H(\alpha)$ curve in Figure 7.1 gives an example. As the fraction of superpeers grows, the highest load level H drops. The intuition behind is clear, as more peers become superpeers, they share the load in the top-layer DHT and the load of the most heavily loaded superpeer drops. In the scenario from Figure 7.1, we assume that the system is homogeneous in that all peers set the same values for their load limit. In that case, the highest load level shown in the figure is the load level of *any* superpeer, provided that every superpeer processes the same portion of the total signaling traffic. When the system is heterogeneous, the highest load level depends not only on the number of superpeers, but also on *which* peers are superpeers. In this case, the $H(\alpha)$ curve is not smoothly shaped as in Figure 7.1, but exhibits jumps (cf. Figure 5.5 in Section 5.5.3).

We define the optimal operating point of the system as the point in which no superpeer is overloaded, while the total operation costs are as small as possible. In the example from Figure 7.1, this point is denoted by α_{opt}. In other words, our optimality criterion represents a trade-off between a centralized system in which the operation costs is small but the central entity bears most of the load (and may become overloaded), and a fully decentralized system in which the load per peer is relatively small but the total operation costs are high.

As mentioned above, the $H(\alpha)$ curve may have different shapes in homogeneous and heterogeneous systems. Nevertheless, the optimality criterion applies regardless of the properties of individual peers, i.e. select the leftmost point within the region defined by the requirement $H(\alpha) \leq 100\%$. In addition, when our load balancing algorithm presented in Chapter 6 is running in the background, the load levels of superpeers have approximately the same value. As a result, we can turn a heterogeneous system into a homogeneous one with respect to the load levels. The highest load level coincides with the load level of any specific superpeer. This enables us to define a quantity called "system load" that represents the mean load level of all superpeers. According to our definition of the optimal operating point, this system load should be close to 100%, but never exceed this value. In the following section we see how the system load is used as an input parameter of our algorithm that achieves and maintains a cost-optimal system operation.

7.4 Distributed Algorithms for Cost-Optimal System Operation

Our goal is to achieve and maintain the cost-optimal operating point of the considered hierarchical P2P system. This optimal operating point corresponds to the optimal number of superpeers in the system, i.e. the optimal superpeer ratio α_{opt}. In the following, we present a set of distributed algorithms needed for achieving and maintaining such a cost-optimal operation. In particular, we require

- an algorithm to balance traffic load among all superpeers (in terms of equal load levels),
- an algorithm to perform local estimations of all system parameters that are needed to determine the currently optimal superpeer ratio α_{opt}, and
- an algorithm that uses local estimations of system parameters to achieve and maintain a cost-optimal system operation.

Note that the third algorithm is the key contribution of this chapter, and that the first two are rather necessary ingredients.

7.4.1 Load Balancing Algorithm

In the considered system architecture most of the generated signaling traffic is handled by the superpeers. It is essential for a proper system operation to include a load balancing algorithm that ensures an equal load distribution among superpeers. The quantity to balance is load levels of superpeers, i.e. all superpeers should be loaded equally with regard to their (individual) load limit. In Chapter 6 we have presented an appropriate load balancing algorithm to do this. Its key idea is to piggyback load information with every message exchanged between superpeers, and to use this information whenever a joining peer is assigned to a superpeer. Analytically and by simulations we have shown that this algorithm achieves a standard deviation of load levels across superpeers within a few percent, and at the same time generates less overhead than, e.g. the renowned "power of two choices" [BCM03] algorithm. The proposed load balancing algorithm is a first prerequisite for achieving and maintaining a cost-optimal system operation.

7.4.2 Estimation of System Parameters

A second prerequisite is the accurate estimation of global system parameters, based on the partial view of every single peer on the system. Estimations of the following system parameters are required as input parameters of our algorithm to achieve and maintain a cost-optimal system operation which we present in the following section:

- The number of superpeers N_{SP}
- The total number of peers N
- The total number of lookups in the system per time unit, i.e. the system lookup rate Q
- The total number of shared data items in the system F
- The mean superpeer capacity $c_{\mathrm{mean}}^{\mathrm{max}}$

Estimation of N_{SP}. To estimate the current number of superpeers N_{SP} in the top-layer Chord ring, we apply and modify the Chord-specific estimation algorithm presented in [BSH05]. This algorithm relies on measuring the peer density in a peer's successor list, and on extrapolating it to the complete identifier space. The authors show that the distribution $P(d)$ of the distance d between any peer and its direct successor on the ring is

$$P(d) \approx \left(1 - \frac{n}{2^b}\right)^d \cdot \frac{n}{2^b} \tag{7.2}$$

where 2^b is the size of the identifier space and n is the number of peers on the ring. They conclude that the interval size I between a peer and its direct successor is approximately geometrically distributed:

$$I \sim \mathrm{geom}(p) \text{ with parameter } p = n \ / \ 2^b \tag{7.3}$$

According to [SMK$^+$01], every superpeer in the top-layer Chord ring maintains a list with the closest succeeding superpeers on the ring. This successor list provides multiple samples of the random variable I and, based on the maximum likelihood estimation, allows for a good estimation of p and consequently n, which is N_{SP} in our case.

To have an even more precise estimation of N_{SP}, we improve the above algorithm as follows. Every superpeer calculates its local estimation of N_{SP} based on the entries in its successor list. This value is then piggybacked in messages that are exchanged between superpeers. As a result, superpeers can also rely on the estimations of other superpeers when determining the current number of superpeers N_{SP}, simply by building the mean value of their own estimation and the x estimations they recently received from other superpeers. For our implementation, we set $x = 16$. In this way, we can reduce the estimation error to approximately 2%. Note that we configure the superpeers so that they create piggybacks at the same rate. This prevents superpeers with a high traffic volume from piggybacking more messages than superpeers with a lower traffic volume, and therefore from gaining too much importance.

Estimation of N, Q, F and $c_{\mathrm{mean}}^{\max}$. The total number of peers N in the system can be approximated as $N = N_{\mathrm{SP}} \cdot (G + 1)$, where G is the average number of leafnodes per superpeer. Hence, we estimate N by using the estimations of N_{SP} and G. The latter is again obtained from piggybacked neighbor information. Every superpeer includes its current number of leafnodes when sending messages to other superpeers. Then, a superpeer estimates the average number of leafnodes G by building the mean value of its own and the x recently received values from other superpeers.

The system lookup rate Q, the total number of shared data items in the system F, and the mean superpeer capacity $c_{\mathrm{mean}}^{\max}$ are estimated in a similar way. Every superpeer measures its lookup rate (generated by own lookups and lookups from its leafnodes), its number of shared data items (including those of its leafnodes) and its capacity, and piggybacks these values as described above. Again, the mean value, calculated from the own value and the x recently received values, is used to estimate each particular parameter.

7.4.3 Achieving and Maintaining Cost-Optimal System Operation

We define the mean value of all superpeers' load levels as the current system load Λ:

$$\Lambda = \sum_{\mathcal{S}} \lambda_{\mathcal{S}} \; / \; N_{\mathrm{SP}} \qquad (7.4)$$

As said earlier, our goal is to have a desired system load Λ_{des} that is close to, but never above 100%. In the following we set $\Lambda_{\mathrm{des}} = 90\%$. This is a reasonable value for the desired mean load level of superpeers because of two reasons. First, the additional traffic load of superpeers generated by churn and load balancing in the top-layer DHT is not considered in our analytical model. Second, we provide a buffer for an unexpected increase of a superpeer's traffic load, e.g. caused by a bursty lookup traffic.

The use of the load balancing algorithm presented in Chapter 6 leads to minor deviations among the individual load levels of superpeers, and therefore to a minor deviation of a superpeer's load level from the system load Λ. Consequently, a superpeer could use its own load level as an approximation of Λ. However, due to the random nature of lookups, a superpeer may temporarily face a higher or lower load level. Thus, we do not only rely on a superpeer's own load level, but take also the x piggybacked load levels that were recently reported from other superpeers into account. We refer to the resulting mean value as the current "experienced system load" Λ_{exp} of a superpeer. In various simulations we have found that the locally measured value of Λ_{exp} is a good approximation of the current system load Λ.

Based on its value of Λ_{exp}, a superpeer can decide whether the system requires more superpeers in order to decrease the system load (if $\Lambda_{\text{exp}} > \Lambda_{\text{des}}$) or if too many superpeers are in the system and thus some of them must be demoted to leafnodes in order to increase the system load (if $\Lambda_{\text{exp}} < \Lambda_{\text{des}}$). In the following we show how the load levels can be adjusted to the desired value by changing the number of superpeers appropriately. So we establish a dependency between Λ and N_{SP}.

In addition to measuring the experienced system load Λ_{exp}, a superpeer can also calculate Λ based on its estimations of N_{SP}, N, Q, F and $c_{\text{mean}}^{\max}$. This is done by estimating the current total signaling traffic processed by all superpeers (in messages per second), and by dividing it by the estimated number of superpeers and the estimated mean superpeer capacity. The total signaling traffic of all superpeers is composed of lookup traffic L_{SP}, maintenance traffic M_{SP} and republish traffic R_{SP}, so we get

$$\Lambda = f(N_{\text{SP}}) = \frac{L_{\text{SP}} + M_{\text{SP}} + R_{\text{SP}}}{N_{\text{SP}} \cdot c_{\text{mean}}^{\max}} \tag{7.5}$$

In Section 5.4, we have performed a detailed study on the generated signaling traffic in hierarchical DHT-based system architectures, including the superpeer-leafnode architecture that we consider in this chapter. (We named it "SiCo" in Chapter 5.) We have derived equations for the lookup traffic (Section 5.4.1), the maintenance traffic (Section 5.4.2) and the republish traffic (Section 5.4.3). In general, we could now use these equations on every superpeer to estimate the lookup traffic L_{SP}, the maintenance traffic M_{SP} and the republish traffic R_{SP} by means of the estimations of N_{SP}, N, Q and F. However, in Chapter 5 we have assumed that references to shared data items are stored on both types of peers, on superpeers and on normal peers (leafnodes). This assumption was necessary to enable a comparison of different intra-group structures. Nevertheless, the results from Chapter 4 and the discussion in Section 5.7.3 show that the system performance increases significantly when storing references only on superpeers. Consequently, we make use of this alternative in the following and recalculate the expressions for L_{SP}, M_{SP} and R_{SP} under this assumption. As said earlier, we focus on the number of uploaded messages.

Figure 7.2 shows the modified lookup procedure when references are stored only on superpeers. In contrast to Figure 5.2, the responsible superpeer $\mathcal{S}_2$ does not forward the lookup request to the responsible group peer $\mathcal{R}$, but sends the corresponding key-value-pair directly back to the peer $\mathcal{I}$ that has initiated the lookup. As a result, each lookup now generates an average upload of $\log N_{\text{SP}} + 2$ messages for the superpeers: $\log N_{\text{SP}}$ messages to resolve the responsible superpeer (step 2), 1 message to forward the lookup

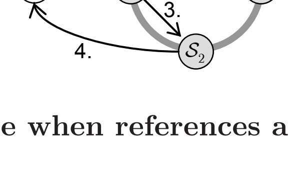


Figure 7.2: Lookup procedure when references are stored only on superpeers.

request from $\mathcal{S}_1$ to $\mathcal{S}_2$ (step 3), and 1 message to send the key-value-pair from $\mathcal{S}_2$ to $\mathcal{I}$ (step 4). Therefore, superpeers can estimate the lookup traffic of superpeers L_{SP} by multiplying the average number of messages that superpeers upload on every lookup with the system lookup rate Q:

$$L_{\mathrm{SP}} = Q \cdot (\log N_{\mathrm{SP}} + 2) \tag{7.6}$$

The maintenance traffic of superpeers M_{SP} is generated by the Stabilize and the FixFingers algorithms for maintaining the superpeers' Chord DHT, and by pinging leafnodes to detect leafnode failures. It is not influenced by the above alternative of storing references. Hence, the expression that allows superpeers to estimate M_{SP} can directly be taken from Section 5.4.2:

$$\begin{aligned} M_{\mathrm{SP}} &= N_{\mathrm{SP}} \cdot \big(m_{\mathcal{S}(\mathrm{TL})} + m_{\mathcal{S}}\big) = \\ &= N_{\mathrm{SP}} \cdot \left(\frac{3}{T_{\mathrm{STAB}}} + \frac{\log N_{\mathrm{SP}} \cdot 2}{T_{\mathrm{FIX}}} + \frac{G-1}{T_{\mathrm{PING}}}\right) = \\ &= \frac{N_{\mathrm{SP}} \cdot 3}{T_{\mathrm{STAB}}} + \frac{N_{\mathrm{SP}} \cdot \log N_{\mathrm{SP}} \cdot 2}{T_{\mathrm{FIX}}} + \frac{N - N_{\mathrm{SP}}}{T_{\mathrm{PING}}} \end{aligned} \tag{7.7}$$

Note that the timer values are global predefined values that are known by every peer.

In Section 5.4.3 we have seen that republishing a shared data item is similar to the lookup of the responsible peer. As a result, the corresponding republish traffic of superpeers can be computed by using the above expression for the lookup traffic, and by replacing the system lookup rate Q by the number of shared data items in the system F that are republished every T_{REP} seconds. Therefore, superpeers can estimate the republish traffic of superpeers R_{SP} by

$$R_{\mathrm{SP}} = \frac{F}{T_{\mathrm{REP}}} \cdot (\log N_{\mathrm{SP}} + 2) \tag{7.8}$$

Eventually, we see that each superpeer can use its local estimations of N_{SP}, N, Q, F and $c_{\mathrm{mean}}^{\mathrm{max}}$ to calculate the system load Λ. The corresponding equation is given by

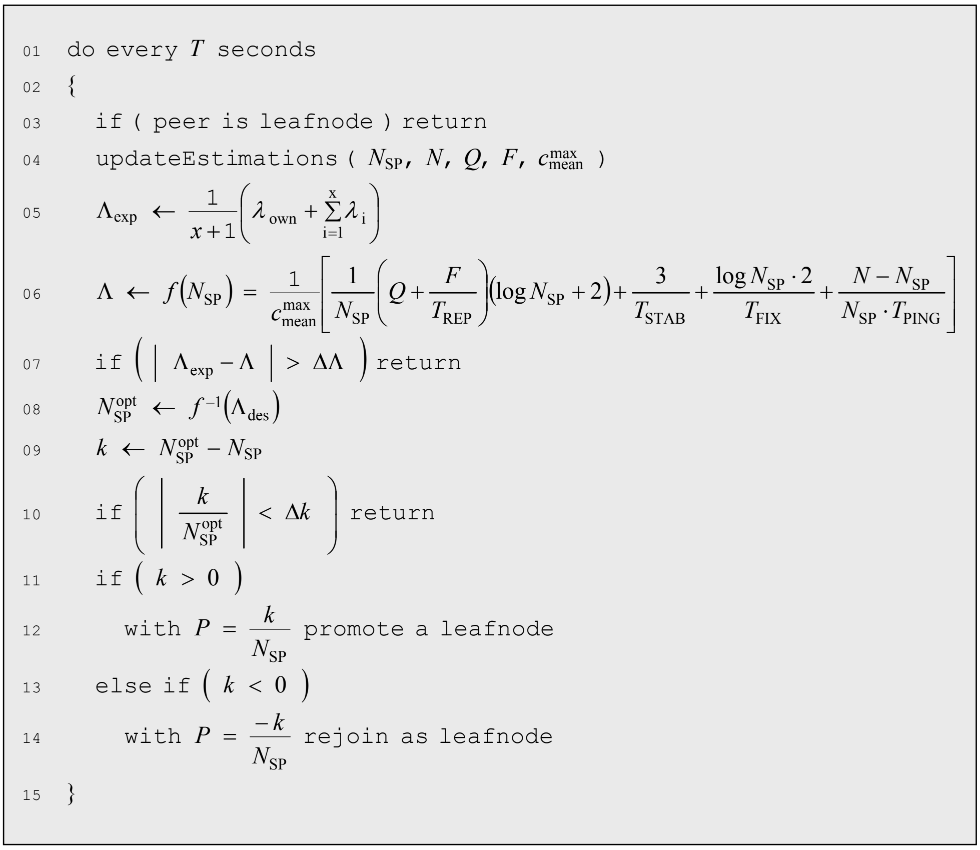
$$\begin{aligned} \Lambda = f(N_{\mathrm{SP}}) &= \frac{1}{c_{\mathrm{mean}}^{\mathrm{max}}} \left[\frac{1}{N_{\mathrm{SP}}} \left(Q + \frac{F}{T_{\mathrm{REP}}}\right) (\log N_{\mathrm{SP}} + 2) + \right. \\ &\left. + \frac{3}{T_{\mathrm{STAB}}} + \frac{\log N_{\mathrm{SP}} \cdot 2}{T_{\mathrm{FIX}}} + \frac{N - N_{\mathrm{SP}}}{N_{\mathrm{SP}} \cdot T_{\mathrm{PING}}}\right] \end{aligned} \tag{7.9}$$

Given equation 7.9, the key idea of our algorithm is that every superpeer computes the inverse function of f given the desired system load

$$N_{\mathrm{SP}}^{\mathrm{opt}} = f^{-1}(\Lambda_{\mathrm{des}}) \tag{7.10}$$

and then, cooperatively with other superpeers, decides how this optimal number of superpeers $N_{\mathrm{SP}}^{\mathrm{opt}}$ can be achieved.

Finding the inverse function of f is not a trivial problem and could in principle be done by means of the Lambert W function [CGH+96]. For simplicity, we prefer an iterative method of resolving $N_{\mathrm{SP}}^{\mathrm{opt}}$ on every superpeer. In equation 7.9, we initially set N_{SP} to the



```
01 do every $T$ seconds
02 {
03   if ( peer is leafnode ) return
04   updateEstimations ( $N_{\mathrm{SP}}$, $N$, $Q$, $F$, $c_{\mathrm{mean}}^{\max}$ )
05   $\Lambda_{\mathrm{exp}} \leftarrow \frac{1}{x+1}\left(\lambda_{\mathrm{own}} + \sum_{\mathrm{i=1}}^{\mathrm{x}} \lambda_{\mathrm{i}}\right)$
06   $\Lambda \leftarrow f(N_{\mathrm{SP}}) = \frac{1}{c_{\mathrm{mean}}^{\max}}\left[\frac{1}{N_{\mathrm{SP}}}\left(Q + \frac{F}{T_{\mathrm{REP}}}\right)(\log N_{\mathrm{SP}} + 2) + \frac{3}{T_{\mathrm{STAB}}} + \frac{\log N_{\mathrm{SP}} \cdot 2}{T_{\mathrm{FIX}}} + \frac{N - N_{\mathrm{SP}}}{N_{\mathrm{SP}} \cdot T_{\mathrm{PING}}}\right]$
07   if $\left( \left| \Lambda_{\mathrm{exp}} - \Lambda \right| > \Delta\Lambda \right)$ return
08   $N_{\mathrm{SP}}^{\mathrm{opt}} \leftarrow f^{-1}(\Lambda_{\mathrm{des}})$
09   $k \leftarrow N_{\mathrm{SP}}^{\mathrm{opt}} - N_{\mathrm{SP}}$
10   if $\left( \left| \frac{k}{N_{\mathrm{SP}}^{\mathrm{opt}}} \right| < \Delta k \right)$ return
11   if $( k > 0 )$
12     with $P = \frac{k}{N_{\mathrm{SP}}}$ promote a leafnode
13   else if $( k < 0 )$
14     with $P = \frac{-k}{N_{\mathrm{SP}}}$ rejoin as leafnode
15 }
```

Figure 7.3: Algorithm to achieve and maintain cost-optimal operation.

currently estimated value. Then we gradually increment (for $\Lambda > \Lambda_{\mathrm{des}}$) or decrement (for $\Lambda < \Lambda_{\mathrm{des}}$) N_{SP} and calculate the resulting value of Λ with equation 7.9. We do this until Λ reaches the desired system load Λ_{des}. In the end, the corresponding value of N_{SP} is the currently optimal number of superpeers (based on the estimations of this superpeer).

Based on the above argumentation, we propose the algorithm from Figure 7.3 to achieve and maintain the optimal number of superpeers $N_{\mathrm{SP}}^{\mathrm{opt}}$ that leads to a system load Λ that is close to the desired system load Λ_{des}:

Periodically every T seconds *(line 01)*, every superpeer *(03)* does the following:

1. Update the estimations of N_{SP}, N, Q, F and $c_{\mathrm{mean}}^{\max}$ *(04)*.

2. Measure the experienced system load Λ_{exp} by building the mean value of the own load level and the x recently received load levels of other superpeers *(05)*.

3. Calculate the system load Λ by means of the estimated parameters according to equation 7.9 *(06)*.

4. Check if the estimations are good enough: Only if Λ deviates at most $\Delta\Lambda$ from Λ_{exp}, proceed with the algorithm. Otherwise, abort this run and wait another T seconds for the next run *(07)*. (Because we do not want to base our algorithm on incorrect estimations, we judge on the quality of the current estimations by comparing the

estimated value of Λ to the measured value Λ_{exp}. Only if both values deviate at most $\Delta\Lambda$ from each other, we consider the current estimations as good enough to proceed with our algorithm.)

5. Define a desired system load Λ_{des}, and determine the optimal number of superpeers $N_{\text{SP}}^{\text{opt}}$ to reach Λ_{des}, as described above. *(08)*

6. Calculate the necessary change in the number of superpeers k *(09)*.

7. Check if the necessary change is significant: If the necessary change in the number of superpeers k is below a certain threshold Δk, abort this run and wait another T seconds for the next run *(10)*. The reason is that small values of k indicate that the current superpeer ratio is close to the optimum, and no system modification is required. In other words, we provide a hysteresis that prevents some superpeers from promoting leafnodes to superpeers while other superpeers leave the system and rejoin as leafnodes.

8. If more superpeers are needed *(11)*, then with probability k/N_{SP} promote a leafnode to a superpeer *(12)*.

9. If too many superpeers are in the system *(13)*, then with probability $-k/N_{\text{SP}}$ (k is negative in this case) leave the system and rejoin as a leafnode *(14)*.

It is extremely important for the proper operation of this algorithm to precisely select the most capable peers as superpeers. In other words, when promoting a leafnode to a superpeer, a leafnode with a high capacity $c_{\mathcal{P}}^{\max}$ must be chosen. To solve this problem, we again distribute information about leafnode capacities by piggybacking. Every superpeer extends messages sent to other superpeers with the address and the capacity of its most capable leafnode. Thus, when deciding that a leafnode has to be promoted, a superpeer can use this information in order to promote the currently most capable leafnode in the system that it is acquainted with (even if this leafnode is connected to another superpeer).

7.5 Evaluation

Having introduced our algorithm to reach and maintain an optimal configuration of our hierarchical P2P system, we now evaluate the algorithm in multiple different simulation scenarios. The main goals of the simulations are

- to demonstrate that the given algorithm can accurately approximate the optimal system configuration (that can be determined from global knowledge of the relevant system parameters),
- to show that the optimal system configuration leads to a system load that is close to the desired system load, and
- to measure the benefits of the hierarchical system in comparison to a flat system.

Table 7.1: Modeling of peers.

Parameter	Value	Distribution across peers
Session duration	mean: 1800 s	Negative exponentially distributed
Lookup rate	mean: 1/min	Negative exponentially distributed
Number of shared data items	mean: 20	Negative exponentially distributed
Capacity	1 ... 13 msgs/s	Uniformly distributed

7.5.1 Simulation Setup

For our simulations, we model peers as follows. We assign an individual session duration, lookup rate, number of shared data items and capacity to every peer. The values of these parameters and their distributions are given in Table 7.1. For assigning a peer's capacity, we assume a maximal upload bit rate between 50 kbit/s and 500 kbit/s, and an average message size of 500 Bytes. Further, we assume that 10% of the available upload bit rate are used for signaling. This results in an upload capacity between $50000/(500 \cdot 8) \cdot 10\% \approx 1$ and $500000/(500 \cdot 8) \cdot 10\% \approx 13$ messages per second. For any peer that leaves the system we assume the worst case, i.e. the leaving peer fails without notifying its neighbors.

We use an event-based simulator[1] that is derived from an ns-2 [NS2] implementation of our system. As in Chapter 4, we again improve the simulation performance by simplifying the model of the underlying physical network. Instead of simulating the complete protocol stack of the ISO/OSI reference model, we add a randomly chosen latency between 50 ms and 150 ms to every message exchanged between two peers. We further assume that no computing time is needed to process incoming messages, and, for the detection of failed peers, we set the timeout to recognize a non-responding peer as failed to 1 second.

In a first step, we evaluate our algorithm to see whether it produces superpeer ratios that come close to the theoretical optimum, and whether these superpeer ratios result in a system load that approximates Λ_{des} (Section 7.5.2). Then, in Section 7.5.3, we compare the generated signaling traffic and the success rate and latency of lookups in our hierarchical system with the flat system design. In addition, we evaluate the behavior of both systems in scenarios that are characterized by high churn rates.

We simulate multiple systems with a total number of $N \in \{500; 1000; 2000; 5000; 10000\}$ peers. Each scenario consists of three phases: a "join phase" (duration: 2 hours) where all N peers join the system (note that we start the system with an initial number of 32 superpeers, and that no peers leave during the join phase), a "churn phase" (2 hours) where the number of peers stays approximately constant, and a "leave phase" (1 hour) where the arrival rate of peers is 0 and the active peers leave the system according to their negative exponentially distributed session duration. Obviously, the churn phase can be seen as the normal mode of system operation, where joining peers substitute peers whose session duration is over. The join phase as well as the leave phase are special cases of starting or shutting down the system. They can be seen as a worst case scenario for our algorithm due to the high increase or decrease in the number of peers.

[1] See Appendix A for details on the implementation.

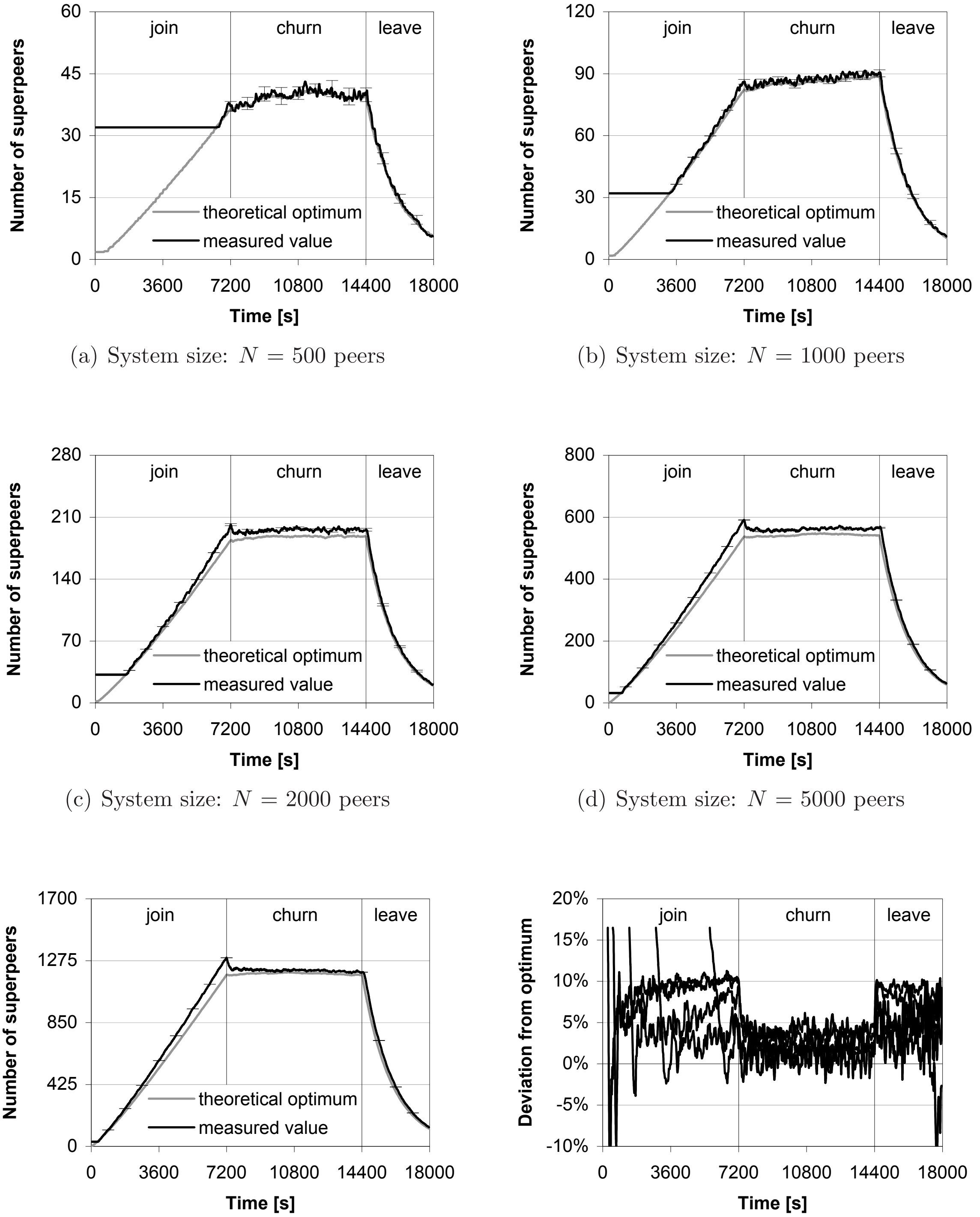
Our algorithm to reach and maintain the optimal superpeer ratio is configured with the following parameters. It is executed every $T = 20$ s on every superpeer. The thresholds $\Delta\Lambda$ and Δk are both set to 10%, and the desired system load Λ_{des} is set to 90%, i.e. the load levels of superpeers should be approximately 90%. Note that we do not claim here that these values represent the optimal input parameters of our algorithm. However, in multiple experiments with varying values for T, $\Delta\Lambda$, Δk and Λ_{des} we found that the above values lead to good results. Further, we set Chord's periodic timer values for Stabilize, FixFingers and republishing shared data items to $T_{\text{STAB}} = 5$ s, $T_{\text{FIX}} = 30$ s and $T_{\text{REP}} = 300$ s, respectively. The periodic Ping/Pong procedure between superpeers and leafnodes is executed every $T_{\text{PING}} = 5$ s.

To have a statistical validation of the results, we simulate each particular scenario 10 times, build mean values and show the 90% confidence intervals.

7.5.2 Algorithm Behavior

In our first simulations we evaluate whether our algorithm produces superpeer ratios that come close to the optimal value, and whether the desired system load can be achieved. Throughout multiple simulation scenarios we log the number of superpeers every 30 seconds, and compare this value to the theoretical optimum that we obtain from a global view on the current system state. We determine the theoretical optimum by tuning the superpeer ratio to a value where the system load takes the desired value of 90%. This is possible as the global view on the system allows us to select the most capable peers as superpeers, and to apply the exact values of the current number of lookups Q in the system and the current number of shared data items F in the system to equation 7.9.

Figures 7.4(a) to 7.4(e) show the measured absolute number of superpeers and the calculated theoretical optimum for different system sizes. In addition, Figure 7.4(f) shows the relative deviation of the measured superpeer ratio from the theoretical optimum. We see that in each particular scenario the measured number of superpeers is always close to the theoretical optimum. This is especially true for the churn phase, i.e. the normal mode of system operation, where the relative deviation is almost always below 5%. During the join and leave phase, i.e. the worst case scenarios, our algorithm produces superpeer ratios that are slightly above the optimal value. Here, the maximum deviation is approximately 10%. The reason is that the high rate of joining or leaving peers leads to some inaccuracy in the estimations, and thus to an increased number of superpeers. However, this is not harmful to our hierarchical system. According to Figure 7.1 in Section 7.3, a superpeer ratio greater than the optimal one leads to an increased signaling traffic and to a decreased load of superpeers. Thus, having slightly more superpeers than actually needed only reduces the system load a little bit, while the small increase in the total signaling traffic is acceptable. Another reason why the measured number of superpeers is almost always above the theoretical optimum is that in practice, it is not always possible to promote the currently most capable leafnode to a superpeer. In contrast to the theoretical optimum, sometimes leafnodes with a lower capacity are selected as superpeer, because the promoting superpeer does not know any other, more capable leafnode. As a result, the mean capacity of all superpeers in the system is slightly



(a) System size: $N = 500$ peers

(b) System size: $N = 1000$ peers

(c) System size: $N = 2000$ peers

(d) System size: $N = 5000$ peers

(e) System size: $N = 10000$ peers

(f) Relative deviation from theoretical optimum.

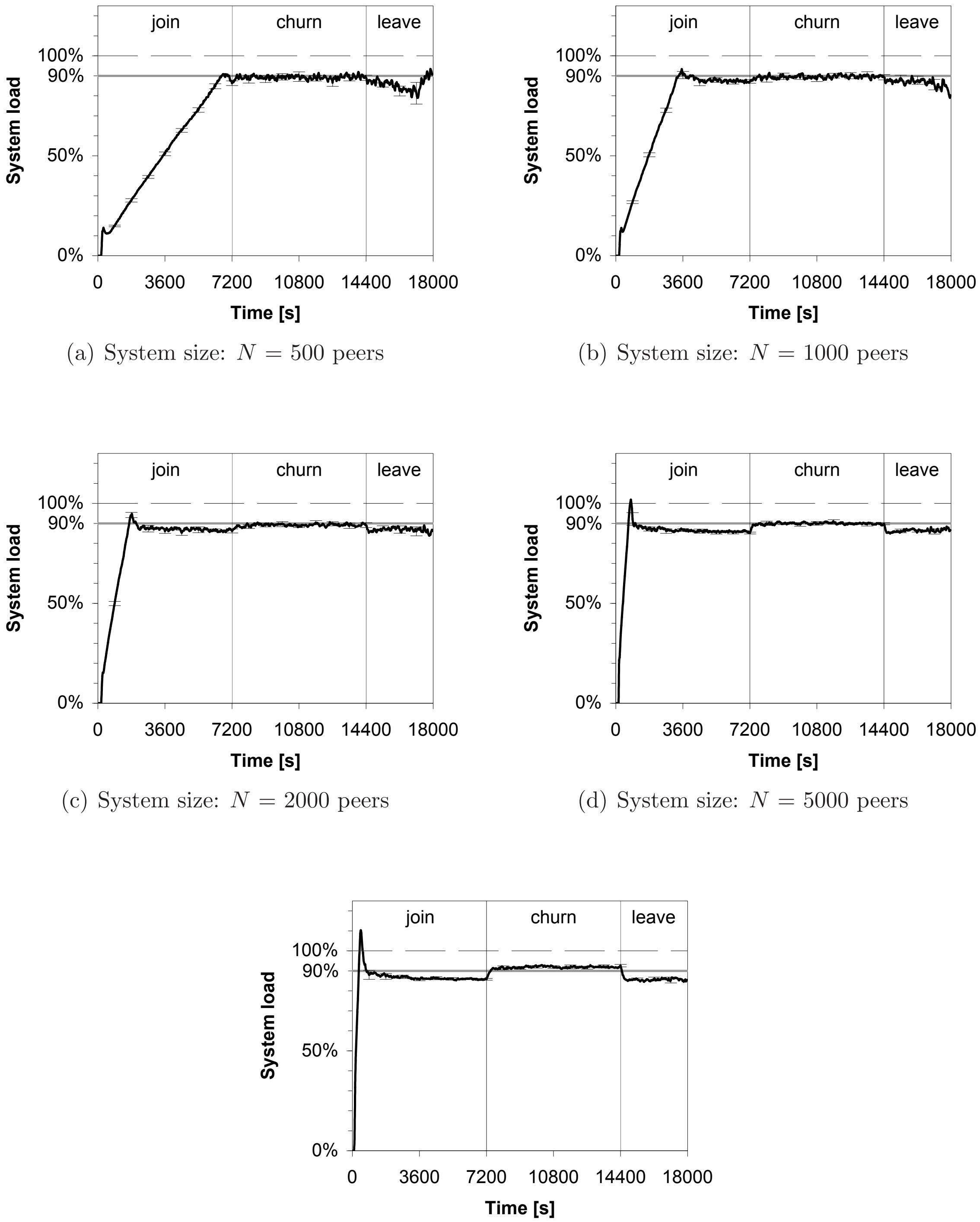
Figure 7.4: Number of superpeers: Theoretical optimum and measured value.

below the maximal possible (= theoretically optimal) value, and thus we need slightly more superpeers than the theoretical optimum to achieve the desired system load.

To verify that the superpeer ratios that our algorithm produces lead to the desired system load, we log the load levels of all current superpeers every 30 seconds. Their mean value is what we have defined as system load Λ, and this value should be close to 90%. In Figures 7.5(a) to 7.5(e), the measured system load is plotted over time. During the churn phase, it always approaches the desired value. This validates the feasibility of our approach and shows the proper operation of our algorithm. During the join and leave phase, we have seen that the number of superpeers may be slightly higher than the optimum. According to the above reasoning, this leads to a decreased system load which can also be seen in the diagrams.

For larger scenarios, we notice peaks in the system load at the beginning of the simulations. We emphasize that these peaks are more a consequence of a peculiarity in our system setup than a real problem. Namely, the system boots with 32 superpeers. As more peers join the system during the join phase, these initial 32 superpeers can for a while handle all the traffic load generated in the system. Later, when the system load raises above 90%, more superpeers are required to distribute the traffic load and reduce the individual load of superpeers. However, the initial 32 superpeers take some time to perceive the increased traffic load, and thus our algorithm does not immediately start to produce new superpeers. During that time, the individual load of the initial 32 superpeers is increased. Larger number of peers joining within the join phase means larger join rates and thus higher peaks in the load of the initial superpeers. To understand this effect (and also other similar dynamics) precisely, we run the following test. We consider a system with $N = 1000$ peers during the churn phase. At time $t = 0$, we triple the lookup rate of all peers in the system (from 1 to 3 lookups per minute per peer) and check how long it takes to promote enough new superpeers to handle the increased load, and how the system load behaves for $t > 0$. The results are shown in Figure 7.6 and Figure 7.7.

In Figure 7.6 we see that the algorithm takes about 150 seconds to produce enough new superpeers and get close to the theoretical optimum. Afterwards, the number of superpeers lies above the theoretical optimum, but the curves converge after around 1000 seconds. As before, the main reason is that it is not always possible to promote the most capable leafnode to a superpeer, as presumed for calculating the theoretical optimum. Figure 7.7 shows how the system load changes after $t > 0$. As the number of superpeers is temporarily below the theoretical optimum, the system load rises above the desired 90%. It stays high for a short time, then drops below 90%, and finally gets close to the desired value after additional 1000 seconds. The most critical observation is that the system load goes above 100%, in this case it reaches 110%, for approximately 150 seconds after the lookup rates have been increased. During that time, superpeers are overloaded. This indicates that sudden changes in the system dynamics may adversely impact the system operation. However, we do not see any serious problem caused by this effect. First of all, we consider artificial scenarios where the traffic load jumps up significantly at a given point in time. Second, the effect can be made less severe by having each peer set some small spare capacity for these sudden changes.



(a) System size: $N = 500$ peers

(b) System size: $N = 1000$ peers

(c) System size: $N = 2000$ peers

(d) System size: $N = 5000$ peers

(e) System size: $N = 10000$ peers

Figure 7.5: System load over time.

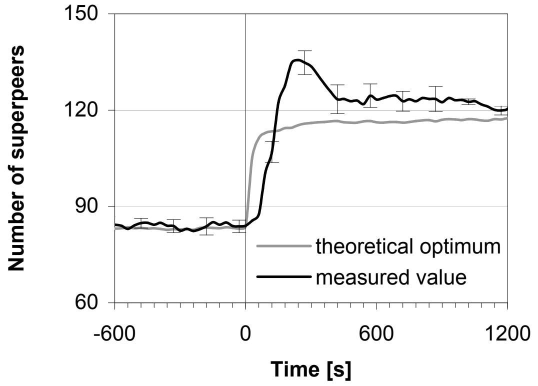


Figure 7.6: Reaction time on changing conditions: Number of superpeers.

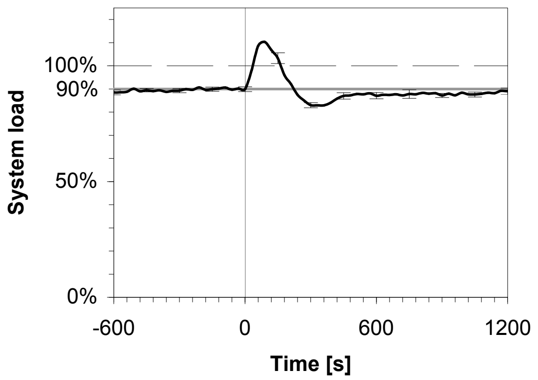


Figure 7.7: Reaction time on changing conditions: System load.

7.5.3 Comparison to Flat System

Having demonstrated the feasibility of the proposed algorithm, we now take one step further and show the benefits of our hierarchical system compared to a conventional flat DHT (Chord in our case). According to the analysis in Chapter 5, we expect our system to generate less signaling traffic than a flat system. Thus, we simulate each scenario also with a flat Chord system, and trace all generated messages in both systems. Further, we divide the analyzed signaling traffic into lookup, maintenance and republish traffic. Figure 7.8 shows the total signaling traffic[2] in both systems depending on the system size, while Figure 7.9 depicts the reduction of the signaling traffic when the hierarchical system is used.

From Figure 7.8 we see that in both systems the total signaling traffic increases almost linearly with the system size, and that the hierarchical system produces significantly less signaling traffic than the flat system. More precisely, the signaling traffic can approximately be halved in all scenarios (bold curve in Figure 7.9). A major part of this reduction is caused by the efficient maintenance algorithms of the hierarchical system.

[2] We show the averaged number of generated messages per second during the join, churn and leave phase.

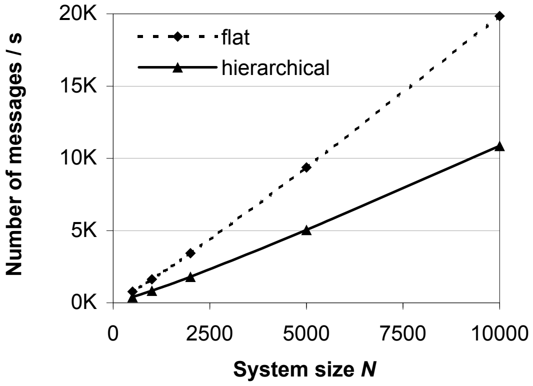


Figure 7.8: Total signaling traffic (confidence intervals are too small to be visible).

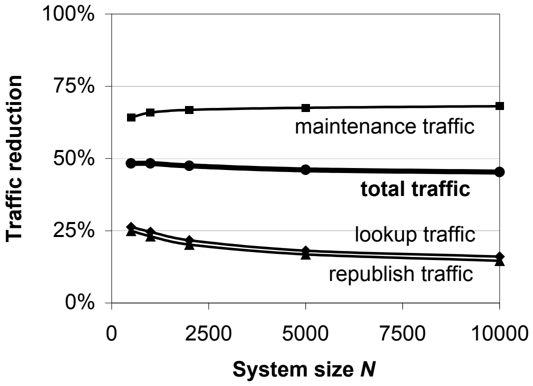
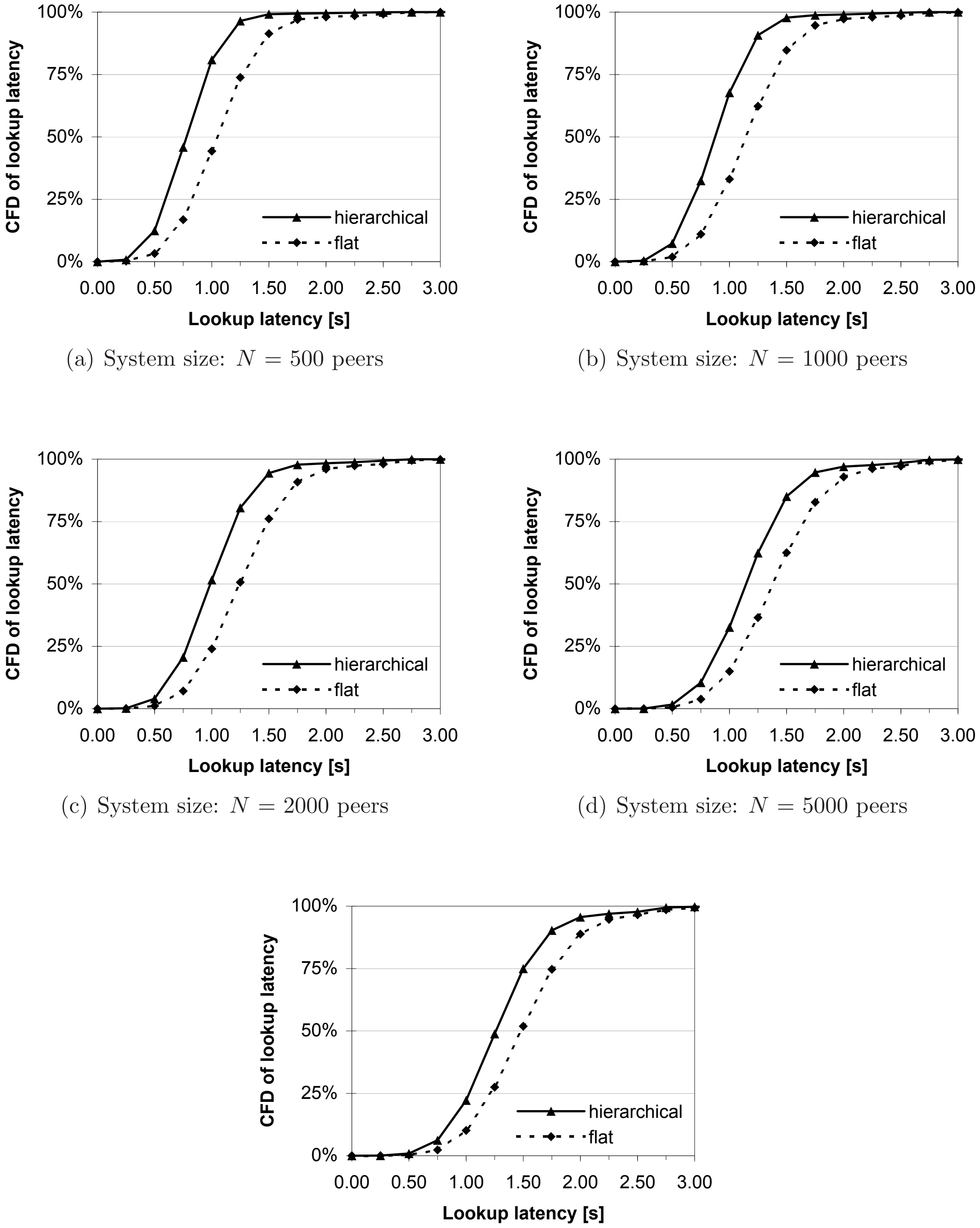


Figure 7.9: Traffic reduction in the hierarchical system (confidence intervals are too small to be visible).

Obviously, a DHT only among superpeers in combination with an economic Ping/Pong algorithm to monitor leafnodes outperforms a large DHT among all peers in the system. The traffic reduction here reaches 68%. However, even the lookup and republish traffic are considerably lower in the hierarchical system. They can be reduced by 15% to 25%.

A further possibility to compare our hierarchical system with a flat Chord system is the responsiveness to lookups in terms of success rate and latency. We here refer to the success rate of lookups as the percentage of lookups that correctly resolve the responsible peer in the system within 5 seconds. Interestingly, we found both systems to be very reliable under the given conditions. Specifying a mean session duration of 30 minutes (see next paragraph for an evaluation of higher churn rates) leads to success rates of more than 99%, independent of the applied system architecture. However, when we take a look at the latency of lookups, we see that the hierarchical system offers a faster resolution. Figures 7.10(a) to 7.10(e) show the cumulative frequency distributions (CFDs) of the lookup latency in the simulated scenarios. Obviously, the more peers participate in the system, the higher is the lookup latency, i.e. the curves of the CFDs are shifted rightwards with increasing N. However, in all cases the CFD curve of the hierarchical system is



(a) System size: $N = 500$ peers

(b) System size: $N = 1000$ peers

(c) System size: $N = 2000$ peers

(d) System size: $N = 5000$ peers

(e) System size: $N = 10000$ peers

Figure 7.10: Cumulative frequency distributions of lookup latency (confidence intervals are too small to be visible).

above the CFD curve of the flat one, indicating that the percentage of lookups resolved within a given time interval is always higher in the hierarchical system. For example, if $N = 10000$, 75% of all lookups in the hierarchical system are resolved within 1.5 seconds, while the flat system reaches only 52%. We further see that the curve of the flat system lies approximately 0.25 seconds right from the curve of the hierarchical system. Thus, we conclude that lookups in the flat system take approximately 0.25 seconds longer on average under the given conditions.

There are two reasons for the faster resolution of lookups in the hierarchical system. First, the number of hops to reach the responsible peer in the hierarchical system is lower than in the flat one, provided that the superpeer ratio α is below 25% (see Appendix B for proof). Second, the smaller DHT in the hierarchical system leads to fewer timeouts during the lookup procedure that are caused by incorrect routing entries. Note that in our simulations, we have not modeled a correlation between the capacity of a peer and its session duration and failure probability. However, we believe that in a real-world scenario such correlation exists. Consequently, when built by capable superpeers that have a high session duration and a low failure probability, the DHT among superpeers is even more stable with less errors in the routing tables. Obviously, in such case the advantages of our hierarchical system are even more significant (cf. also Chapter 4).

We complete our evaluation with a comparison of the hierarchical and the flat system under high churn. Churn has been found to be the most serious challenge for structured P2P systems. Thus, we are interested in whether our hierarchical system also shows a superior performance when churn rates are high. To evaluate this, we consider multiple scenarios with $N = 1000$ peers, vary the mean session duration of peers from 5 minutes down to 1 minute, and judge on both systems' ability to handle churn by analyzing the percentage of lookups that fail during the churn phase (according to the above definition of lookup success). It is important to state that the very low value of 1 minute for the mean session duration is possible only because of the low stabilize period ($T_{\mathrm{STAB}} = 5$ s) that we have specified. If this period is increased, e.g. to save signaling traffic, the effects we describe in the following will emerge also for higher mean session durations.

Figure 7.11 shows the percentage of failed lookups in both systems against a decreasing mean session duration, i.e. an increasing churn rate. We plot it for the worst case where all leaving peers fail without notifying their neighbors (bold curves), and also for the best case where all peers inform their neighbors before leaving the system (normal curves). In all cases, the percentage of failed lookups is below 5% as long as the mean session duration is at least 3 minutes. As mentioned above, this is due to the short stabilization period in the Chord ring. When the mean session duration falls below 3 minutes, the percentage of failed lookups raises. However, the hierarchical system still resolves more than 90% of all lookups even when the mean session duration is only 1 minute and all leaving peers fail. In contrast, nearly one fourth of all lookups in the flat system fail in this case. We can see the same behavior for the best case. Here, approximately 13% of all lookups in the flat system fail when the mean session duration reaches 1 minute, while our hierarchical system still offers an excellent resolution of lookups with a failure rate of only 2%. To sum up, we state that the curve of the hierarchical system is always below the curve of the flat one, showing that the hierarchical system is preferable with regard to reliable resolution of lookups and handling of high churn.

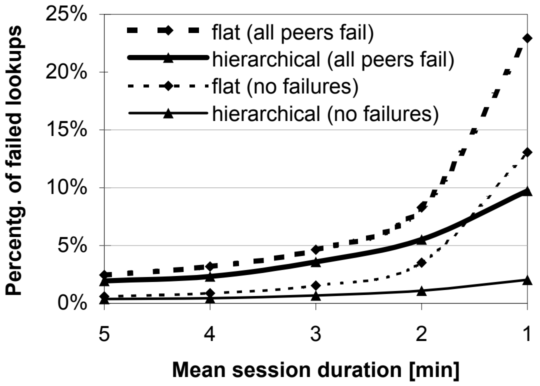


Figure 7.11: Percentage of failed lookups against mean session duration (confidence intervals are too small to be visible).

7.6 Conclusion

This chapter completes our study on the cost-optimal operation of hierarchical DHT-based P2P systems. In the previous chapters, we have

- analyzed different hierarchical DHT-based system architectures,
- performed a detailed traffic analysis of the considered system architectures based on an analytical cost model,
- defined our optimality criterion in terms of the generated signaling traffic and the individual load of each peer,
- found a superior performance of the architecture that connects leafnodes directly and solely with their superpeer, and
- developed an efficient load balancing algorithm that distributes leafnodes properly across the superpeers, considering their individual capacities.

Based on these results, in this chapter we have presented a completely self-organizing hierarchical P2P system whose main design goal is cost-optimal operation. In particular, we have developed a distributed algorithm that achieves and maintains an optimal operating point of the system where the total costs for system operation are minimized while no superpeer is overloaded. The proposed algorithm is self-organizing in the sense that each superpeer estimates a number of relevant system wide parameters based on its local knowledge (and augmented by information obtained from neighboring superpeers), and then takes probabilistic decisions toward the goal. Thus, our algorithm can also be seen as an application of the well-known principle in distributed systems, namely feedback control loop mechanism. Each peer observes the system state and makes a decision that drives the system closer to the state it is supposed to operate in.

The algorithm has been tested in a number of realistic scenarios corresponding to typical settings which we expect the system to be deployed in. The simulations confirm that the algorithm is capable of driving the system from any state to a state which is within a few percents away from the theoretically computed optimum. Further, they show significant advantages of our hierarchical system architecture compared to a flat system design, such as the reduction of total signaling traffic, the reduction of lookup latency or a better handling of churn.

8 Prototypical Implementation

8.1 Introduction

To have a proof of concept of the proposed hierarchical DHT-based P2P system and an experimental platform to evaluate its real-world behavior, we have implemented a prototypical version of the system. In this chapter, we take a look at the test bed configuration in which this prototype is running (Section 8.2), we describe the two available implementations, a full implementation for PCs and notebooks as well as a lightweight implementation for mobile devices (Section 8.3), and we present an exemplary service – a distributed picture sharing service – for our P2P system (Section 8.4). We conclude with a summary of the lessons learned from our prototypical implementation (Section 8.5).

8.2 Test Bed Configuration

8.2.1 Hardware Equipment

The hardware that is available for the prototypical implementation consists of 3 PCs and 2 notebooks in the laboratories at DoCoMo Communications Laboratories Europe GmbH (DoCoMo Euro-Labs) in Munich, Germany, 5 PCs in the laboratories of the Institute of Communication Networks (LKN) in Munich, Germany, and 6 cell phones. The detailed hardware equipment is listed in Table 8.1.

Table 8.1: Hardware equipment of the test bed.

Laboratory	Machine	#	CPU	RAM
DoCoMo	PC	3	Intel Pentium® IV 3 GHz	1 GB
	Notebook	1	Intel Pentium® M 1.73 GHz	512 MB
	Notebook	1	Intel Mobile Pentium® III 750 MHz	128 MB
LKN	PC	1	AMD Athlon® 900 MHz	512 MB
	PC	2	Intel Pentium® III 450 MHz	128 MB
	PC	1	AMD K6® 300 MHz	196 MB
	PC	1	AMD K6® 300 MHz	128 MB
variable	Nokia N80 IE	4	ARM-926 220 MHz	64 MB
(cell phones)	Nokia 6670	2	ARM-9 123 MHz	8 MB

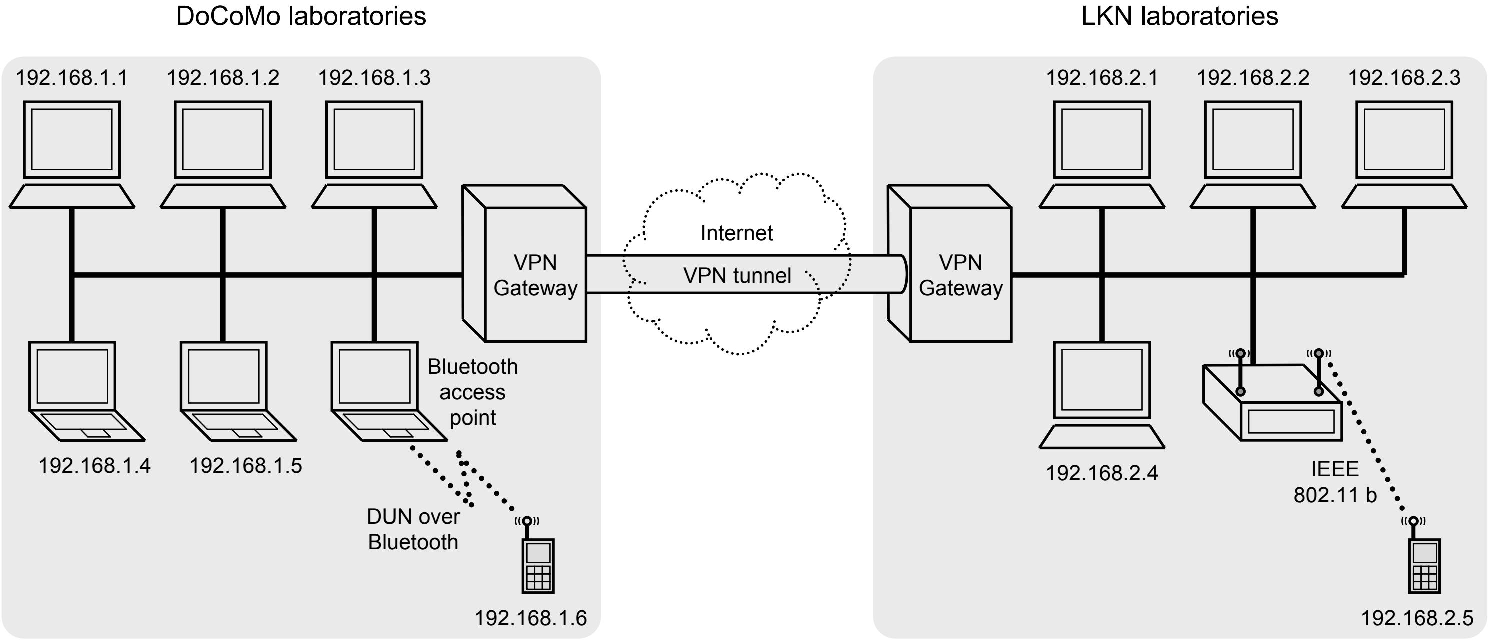


Figure 8.1: Physical network configuration of the test bed.

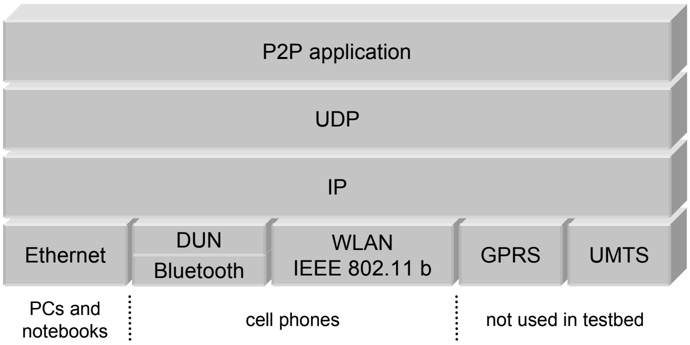


Figure 8.2: Protocol stack of the implementation.

8.2.2 Physical Network Configuration

Figure 8.1 shows the physical network configuration of the hardware components. In the DoCoMo laboratories as well as in the LKN laboratories the PCs and notebooks are connected to a local area network (LAN). In order to allow communication between a peer in the DoCoMo-LAN and a peer in the LKN-LAN, the two LANs are connected by a secured and encrypted VPN tunnel that is established by VPN gateways in both LANs. To connect the cell phones to the other machines in the test bed, we avoid using GPRS or UMTS technology because of the corresponding fees of the mobile network operator. Instead, we emulate the mobile network by the two following solutions:

- We set up a Dial-Up Network (DUN) over a Bluetooth connection between the cell phone and a notebook in the LAN that acts as a Bluetooth access point. This requires a DUN client on the phone and a DUN server on the notebook. Once the connection is established, all IP traffic transmitted from and to the phone is tunneled through the Bluetooth connection.
- We include a WLAN (IEEE 802.11 b) router in the LAN that allows the use of the WLAN interface of cell phones to access the test bed.

However, also the use of “real” GPRS or UMTS networks to connect cell phones to the P2P system is easily possible as we can see from Figure 8.2. It depicts the protocol stack of our implementation. Instead of transmitting IP packets via DUN over Bluetooth or WLAN (IEEE 802.11 b), the cell phones can be configured to transmit IP packets via a GPRS or UMTS connection. This is transparent to the above layers. Note that, in this case, a VPN gateway must be installed on the cell phone to access the LANs at DoCoMo and LKN. Another possibility to allow access for a GPRS- or UMTS-connected cell phone is to equip the machine that acts as superpeer for the cell phone with a second Ethernet interface providing a public IP address.

8.3 System Implementation

There are two implementations of our hierarchical P2P system: a full implementation that provides both superpeer and leafnode functionality and can basically be run on any

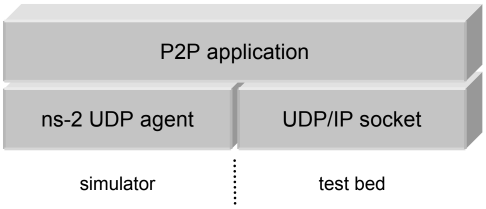


Figure 8.3: Transport layer of simulator and prototype.

type of peer, and a lightweight implementation that provides only leafnode functionality and is designed especially for resource-constrained peers such as cell phones or handheld computers.

8.3.1 Full Implementation

The full implementation of the prototype provides the full functionality that is required for a peer to act as a superpeer and as a leafnode, respectively. This implementation runs on all PCs and notebooks that are part of our test bed.

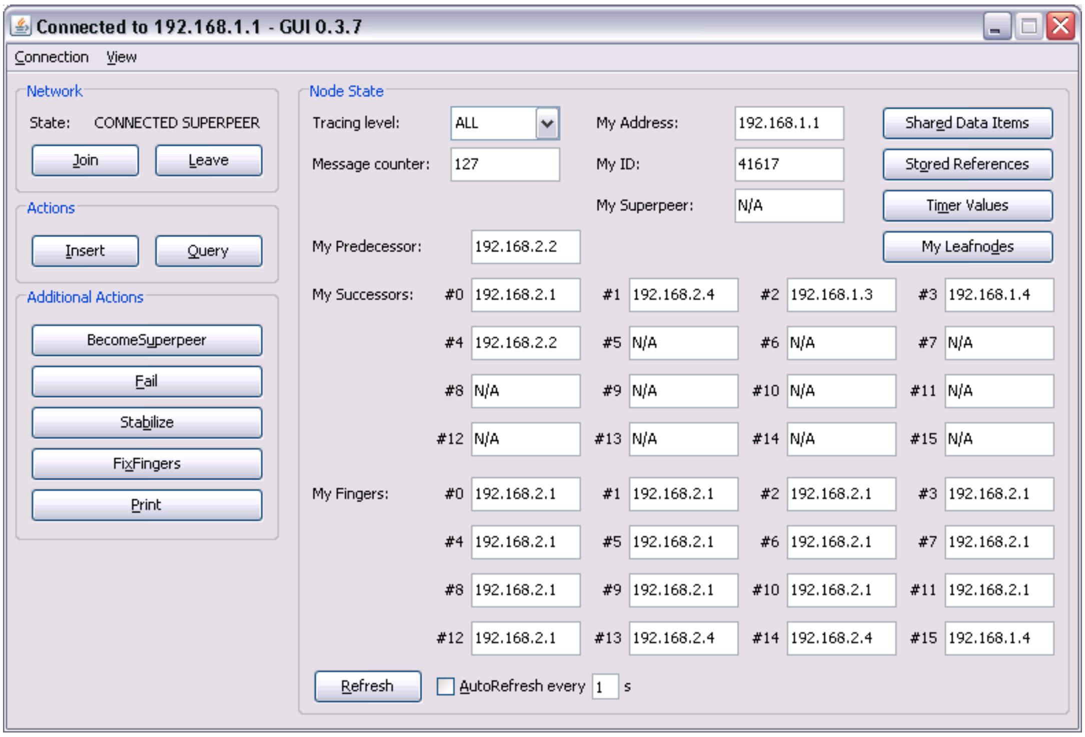
When implementing the prototype, we benefit from the overlay character of P2P systems, since it largely allows the reuse of the existing simulator's source code. In the previous chapters we have used an ns-2 implementation[1] of the system in order to prove the efficiency of the proposed algorithms. Now we reuse this software and develop a running prototype out of it. This can easily be achieved by replacing the ns-2 transport layer agent of the simulator by a real UDP/IP socket, as depicted in Figure 8.3. This is completely transparent to the above P2P application, so it can be reused without any major changes.

A second difference between the simulator and the prototype is the way how the P2P application is controlled. For our simulations, we have generated event files that specify the actions that peers perform at a given time instant. For the prototype, we require a graphical user interface (GUI) that allows the user to control the application. In general, this GUI could be integrated directly into the existing source code as a part of the prototype. However, for our test bed we choose another option. We uncouple the GUI from the P2P application. Therefore, we implement a control interface for the P2P application that can be addressed via a TCP/IP socket, and we develop the GUI as a stand-alone Java [GJSB05] application that communicates with the P2P application through a TCP connection. This offers multiple advantages in comparison to a direct integration of the GUI:

- All P2P applications that are running in the test bed (in the DoCoMo laboratories as well as in the LKN laboratories) can be monitored and controlled from one and the same machine.

- Machines not running the GUI do not have to have a desktop environment installed.

[1] See Appendix A for details on the implementation.



Connected to 192.168.1.1 - GUI 0.3.7

State: CONNECTED SUPERPEER

Tracing level: ALL

Message counter: 127

My Address: 192.168.1.1

My ID: 41617

My Superpeer: N/A

My Predecessor: 192.168.2.2

My Successors:	#0	192.168.2.1	#1	192.168.2.4	#2	192.168.1.3	#3	192.168.1.4
	#4	192.168.2.2	#5	N/A	#6	N/A	#7	N/A
	#8	N/A	#9	N/A	#10	N/A	#11	N/A
	#12	N/A	#13	N/A	#14	N/A	#15	N/A
My Fingers:	#0	192.168.2.1	#1	192.168.2.1	#2	192.168.2.1	#3	192.168.2.1
	#4	192.168.2.1	#5	192.168.2.1	#6	192.168.2.1	#7	192.168.2.1
	#8	192.168.2.1	#9	192.168.2.1	#10	192.168.2.1	#11	192.168.2.1
	#12	192.168.2.1	#13	192.168.2.4	#14	192.168.2.4	#15	192.168.1.4

□ AutoRefresh every 1 s

Figure 8.4: Graphical user interface of the full implementation.

- While the P2P application is programmed in C++, the GUI is programmed in Java and can thus be executed on any Java-compatible machine.

Figure 8.4 shows a screenshot of the GUI. On the left hand side, we see the available actions the user can perform. He can join and leave the P2P system, insert own data items into the system and query for data items that are shared by other peers. Moreover, the GUI offers some additional actions that are implemented for testing purposes, such as intentionally promoting a leafnode to a superpeer or simulating a peer failure. The right hand side shows details about the current state of the peer, e.g. its IP address, its ID in the identifier space and information about neighboring peers. The shared data items of the peer, the references it stores (if it is a superpeer), the timer values it uses, and the leafnodes it manages (if it is a superpeer) can be monitored by calling additional dialogs. The synchronization of the displayed values with the P2P application's current settings (via TCP) can be done actively by pressing the "Refresh" button at the bottom, or periodically by activating the adjoining "AutoRefresh" option.

8.3.2 Lightweight Implementation

The lightweight implementation of the prototype implements only leafnode functionality. It runs on the cell phones that are part of our test bed. Since these devices are too weak to act as superpeers, the lightweight implementation does not provide any superpeer functionality and thus saves resources.

As software platform for the leightweight implementation we have selected Sun's Java Micro Edition (ME) [JME] because of two reasons. First, nearly every mobile device is shipped with Java ME support today. This makes it easy to integrate new devices into the existing test bed. Second, Sun's Java Wireless Toolkit [WTK], a toolbox for developing Java ME applications, allows for a quick and efficient implementation of the software and for testing it locally on an emulated mobile device.

In addition to the basic leafnode functionality, the leightweight implementation includes an interface to the camera software of the cell phone, in order to enable the sharing of pictures that are taken with the phone's camera with other peers in the system. The following section presents details on this exemplary P2P service and shows some screenshots of the leightweight implementation.

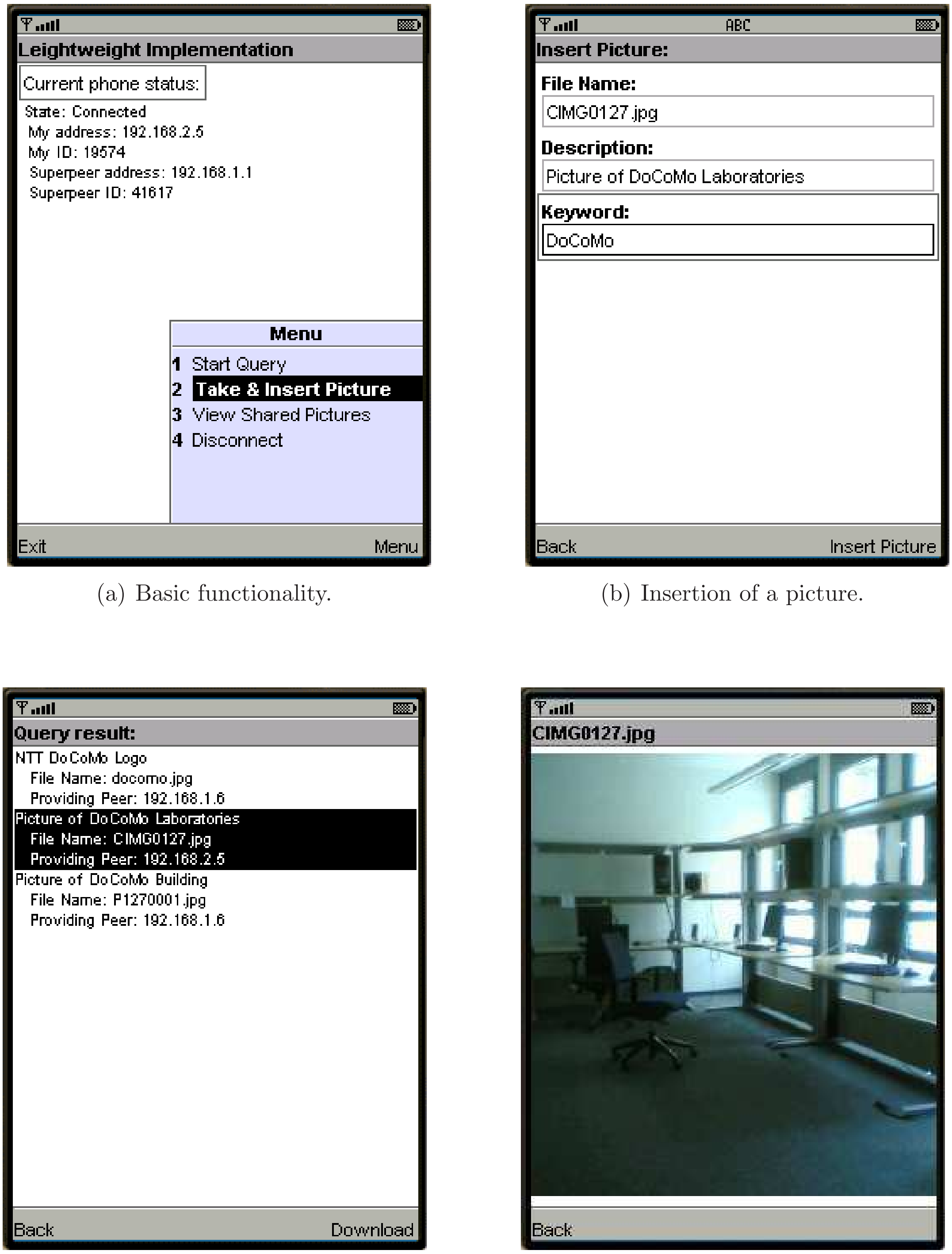
8.4 Service Prototype: Distributed Picture Sharing

To have an example of use of our hierarchical P2P system we have implemented a demonstration application in the form of a distributed picture sharing service. The idea behind it is simple. The cell phones of our test bed are equipped with cameras. We use these cameras to take pictures, then we save the pictures on the phone and share them with other peers in the system. The important difference to a server-based solution is that the pictures remain on the phone, while only a reference to each picture is uploaded onto the responsible superpeer.

The screenshots in Figures 8.5(a) to 8.5(d) depict the typical use of this service. Figure 8.5(a) shows the basic functionality that the leightweight implementation provides after the cell phone has been connected to the P2P system. It allows for initiating a query, inserting new pictures that were taken with the phone's camera, viewing the already shared pictures on the phone, and disconnecting from the system. After taking a picture, the user can share it with other peers. For this purpose, he is presented an "Insert Picture" dialog (cf. Figure 8.5(b)) where he can specify a description and a keyword for his picture. The keyword is used to derive a key for the picture, and thus to determine the responsible superpeer that stores the reference to the shared picture. The description is useful for users that have searched the system and now select the picture(s) that they want to download. For example, a query for "DoCoMo" may result in several query hits for the keyword "DoCoMo", as shown in Figure 8.5(c). By means of the description, the user can select the desired picture, download it from the providing peer, and display it (cf. Figure 8.5(d)). Note that the data transfer is done via a direct HTTP connection between the providing and the downloading peer. Therefore, both implementations (full and leightweight) additionally include standard HTTP server and client functionality.

8.5 Lessons Learned

From our prototypical implementation we have gained insights into the behavior of the proposed hierarchical DHT-based P2P system in a real-world environment. The most



Leightweight Implementation

Current phone status:

State: Connected
My address: 192.168.2.5
My ID: 19574
Superpeer address: 192.168.1.1
Superpeer ID: 41617

(a) Basic functionality.

Insert Picture:

File Name: CIMG0127.jpg

Description: Picture of DoCoMo Laboratories

Keyword: DoCoMo

(b) Insertion of a picture.

Query result:

NTT DoCoMo Logo
File Name: docomo.jpg
Providing Peer: 192.168.1.6
Picture of DoCoMo Laboratories
File Name: CIMG0127.jpg
Providing Peer: 192.168.2.5
Picture of DoCoMo Building
File Name: P1270001.jpg
Providing Peer: 192.168.1.6

(c) Query result.

CIMG0127.jpg

(d) Downloaded picture.

Figure 8.5: Screenshots of the leightweight implementation.

important point here certainly is the observation that the system works as expected. Thus, it validates the feasibility of our approach.

Further, the prototypical implementation shows that a distinction between the different capacities of the participating peers is absolutely essential in a heterogeneous communication environment. We have found that the cell phones are able to participate in our test bed. It is possible to insert pictures taken with the phones' cameras, and to query and to download pictures provided by other peers. However, the constrained resources (especially of the older Nokia 6670) and the slow access to the file system sometimes result in delays of up to several seconds, although the cell phones act only as leafnodes. This underlines the need for further development of the hardware equipment of mobile devices.

A Java ME specific problem has been raised by the numerous different APIs that are necessary for the leightweight implementation on the cell phones, e.g. for Bluetooth support, camera access or file access. Only few cell phones are (currently) available that support all of these APIs. As a result, the possibility of integrating new devices into the existing test bed is (currently) limited. However, this problem will become less important in the future as more and more cell phones will support all necessary APIs.

A further impression that we have received from the leightweight implementation is that the used software development tools Java ME and Java Wireless Toolkit are still at a premature stage. This sometimes resulted in a significantly prolonged software development process. For example, the source code written for a specific cell phone often had to be modified when transferred to another cell phone, even if both cell phones provided the same APIs.

The continuing evaluation of our system is an important research issue for future work. By connecting the cell phones via "real" GPRS and UMTS networks we expect even more realistic insights into the system behavior. In addition, we are planning to extend our local test bed to the global PlanetLab [PLA] test bed, in order to significantly increase the number of peers in the system.

9 Summary

P2P systems are a promising alternative to classical Client-Server systems. Their distributed nature offers a number of advantages compared to centralized systems. Consequently, P2P systems have become a major research topic in recent years and many successful applications based on the P2P paradigm have been developed.

Besides the special case of a centralized P2P system, two main approaches are used to solve the problem of a decentralized, self-organized data lookup: unstructured P2P systems and structured P2P systems. The former approach establishes a random and continuously changing topology and uses a flooding-based search algorithm to locate relevant content. The latter approach sets up a predefined topology and distributes index information about the shared data items across the participating peers, thus enabling their subsequent fast lookup. Especially in highly heterogeneous communication environments, characterized by a large diversity in the provided resources of participants, their transmission data rates, session durations and failure probabilities, we believe that structured P2P systems outperform unstructured P2P systems due to their deterministic lookup path, guaranteed query resolution and their predefined topology that allows for a good estimation of the generated signaling traffic in the system. However, structured P2P systems still pose many challenges that are subject to continual research.

This thesis primarily addresses challenges for structured P2P systems in the context of heterogeneous communication environments. Our main contributions can be summarized as follows. First, we identify the challenges and analyze the requirements of structured P2P systems that are applied in heterogeneous environments. Based on these findings, we develop optimized solutions for their efficient use in such scenarios. In a first step, we propose a novel hybrid design scheme for DHTs. Then, we study hierarchical DHT-based P2P systems. Their hierarchical system architecture turns out to be a solid foundation for P2P systems in heterogeneous environments. We develop a formal analytical cost model that allows us to judge on the optimality of different hierarchical system architectures and thus to compare them. Our comparison shows that a simple hierarchical system architecture composed of a DHT among a carefully chosen set of superpeers and normal peers attached to them is optimal. Further, we propose a novel load balancing algorithm for hierarchical DHT-based P2P systems that shows a superior behavior in comparison to state-of-the-art solutions. Finally, we present a completely self-organizing hierarchical DHT-based P2P system that offers a cost-optimal operation of the system at any time. In addition, we show a prototypical implementation of the system.

The efficiency of our solutions is primarily achieved by distinguishing between the different capacities of the participating peers. The hybrid DHT design scheme presented in Chapter 4 defines two different types of peers (static and temporary) and stores references to shared data items only on the static peers. Thus, it unloads the temporary peers, making it possible for resource-constrained devices such as cell phones or handheld computers to efficiently participate in the P2P system. Further, the hybrid design scheme decreases the maintenance traffic and increases the availability of provided content. The hierarchical DHT-based P2P system developed in Chapters 5 to 7 includes these advantages, but takes one step further by keeping resource-constrained peers out of the DHT and by using the superpeers as proxies for these leafnodes. The distinction between superpeers and leafnodes reflects the varying capacities of peers, thus already forming a suitable architecture for P2P systems in heterogeneous environments. In addition, the proposed algorithms can significantly improve the system performance by dynamically adapting a superpeer-to-leafnode ratio that minimizes the total costs for system operation while not overloading any participating peer.

We believe that the proposed solutions can play an important role in introducing P2P systems in heterogeneous communication environments. Especially with regard to Next Generation Networks (NGN) and Ubiquitous Computing scenarios [Wei91], where any kind of electronic device will be equipped with network access and accessible from all over the world, we think that heterogeneity will become the normal case. Due to their distributed nature and outstanding scalability, P2P systems offer a promising potential for these scenarios.

A Software Implementation of Simulator and Prototype

In order to achieve a better understanding of the simulations performed in this thesis and of the prototypical implementation, we give some details on the software implementation of our system in the following. Figure A.1 shows a simplified class diagram of the implementation.

The class OverlayApp is the core component of the implementation. Each peer creates an instance of this class. It contains attributes like a peer's address, its ID, the address of its superpeer (if it is a leafnode), the leafnodes connected to it (if it is a superpeer) and its load level (if it is a superpeer). Further, the class OverlayApp provides methods that are used for controlling a peer, i.e. methods to join the system, to query for shared data items, to leave the system or simply to fail. These control methods are called by the global Simulator object according to the events specified in the event file (when simulating), or by the control interface of the GUI (when running the prototype). Each OverlayApp has several Manager classes that are responsible for specific tasks that each peer performs: a ContentManager, a FailureManager and a PacketManager. Additionally, if the peer is a superpeer, it has an EstimationManager and a RoutingManager.

The ContentManager manages the shared data items of a peer and provides methods to insert, remove and republish them. If the peer is a superpeer, it also manages the references to shared data items that the peer stores (according to the DHT's mapping rule), and implements a method to transfer references to another peer if necessary.

The FailureManager keeps track of messages sent to other peers that require a response. For this purpose, it holds a list with timers of all monitored messages, and provides methods for starting and stopping these timers. In case of timeouts (indicating a peer failure), the FailureManager executes appropriate failure recovery mechanisms.

The task of the PacketManager is to enable communication between the peers. Thus, its essential methods are *send()* and *recv()*. In addition, the PacketManager allows for creating and evaluating piggybacks that are sent along with messages. In Chapters 6 and 7 we have introduced piggybacks as a requirement for achieving a balanced load among superpeers and for estimating relevant system parameters.

The EstimationManager and the RoutingManager are additional Manager classes that are instantiated only by superpeers. The EstimationManager contains estimations of the system parameters that are needed to determine the currently optimal superpeer ratio, i.e. the number of superpeers, the average number of leafnodes per superpeer, the system lookup rate, the total number of shared data items and the mean superpeer capacity (cf. Section 7.4.2). It further implements methods to update these estimations and to adjust the superpeer ratio by means of the algorithm presented in Section 7.4.3.

The RoutingManager contains all routing information required by superpeers: the predecessor on the Chord ring, the successor list and the finger table. To update this

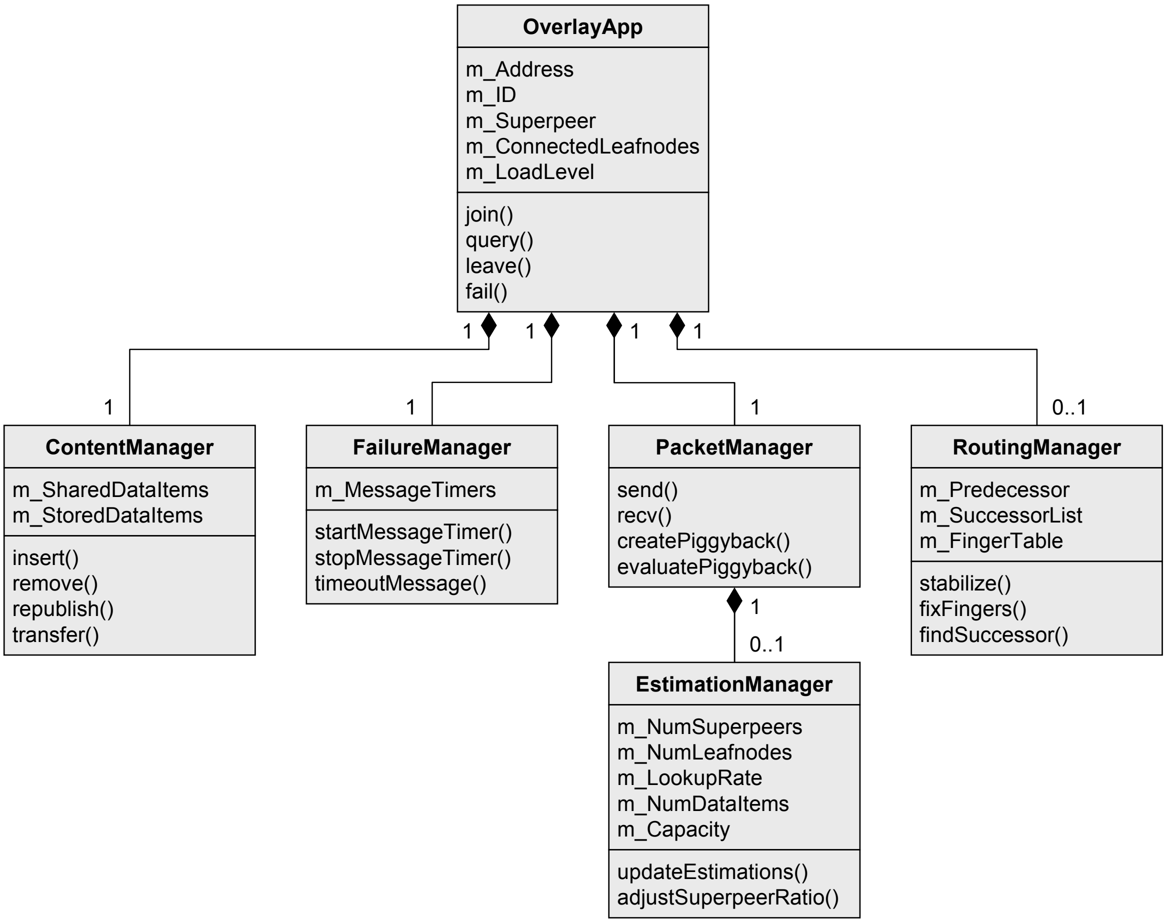


Figure A.1: Simplified class diagram of the implementation.

information, the RoutingManager implements the *stabilize()* and *fixFingers()* methods. To route lookup request to the responsible superpeer on the Chord ring, it further provides the *findSuccessor()* method.

B Routing Performance of Flat and Hierarchical DHT-based P2P Systems

In Section 7.5.3 we have stated that the number of hops to reach the responsible peer in the considered hierarchical system is lower than in the flat one, provided that the superpeer ratio α is below 25%. Below, we prove this claim.

The average number of hops to resolve the responsible peer in a flat Chord DHT is given by

$$h_{\text{flat}} = {}^1\!/_2 \log N \tag{B.1}$$

where N is the number of peers participating in the system. In contrast, the hierarchical system proposed in Chapter 7 requires

$$h_{\text{hierarchical}} = 1 + {}^1\!/_2 \log(\alpha \cdot N) \tag{B.2}$$

hops on average, given that $\alpha \cdot N$ is the number of superpeers in the system. The expression for $h_{\text{hierarchical}}$ reflects the first hop from the peer that initiates the lookup to its superpeer, and the subsequent ${}^1\!/_2 \log(\alpha \cdot N)$ hops to resolve the responsible superpeer in the superpeers' Chord DHT.

We see that the hierarchical system outperforms the flat system if

$$h_{\text{hierarchical}} < h_{\text{flat}} \tag{B.3}$$

and thus

$$\begin{aligned}
1 + {}^1\!/_2 \log(\alpha \cdot N) &< {}^1\!/_2 \log N && \text{(B.4)}\\
2 + \log(\alpha \cdot N) &< \log N && \text{(B.5)}\\
\log(\alpha \cdot N) - \log N &< -2 && \text{(B.6)}\\
\log \frac{\alpha \cdot N}{N} &< -2 && \text{(B.7)}\\
\log \alpha &< -2 && \text{(B.8)}\\
\alpha &< 0.25 && \text{(B.9)}
\end{aligned}$$

C Abbreviations

API	Application Programming Interface
CFD	Cumulative Frequency Distribution
CPU	Central Processing Unit
DHT	Distributed Hash Table
DoS	Denial of Service
DRM	Digital Rights Management
DSL	Digital Subscriber Line
DUN	Dial-Up Network
FTTH	Fiber to the Home
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ISP	Internet Service Provider
LAN	Local Area Network
NGN	Next Generation Network
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PC	Personal Computer
RAM	Random Access Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VPN	Virtual Private Network
WLAN	Wireless Local Area Network

D Mathematical Variables

α	Superpeer ratio
b	Number of bits
$c_{\mathcal{P}}$	Number of messages processed by peer $\mathcal{P}$ per time unit
$c_{\mathcal{P}}^{\max}$	Maximum number of messages peer $\mathcal{P}$ is able to process per time unit (= load limit / capacity of peer $\mathcal{P}$)
C	Number of messages in the system per time unit (= total costs for system operation)
d	Connectivity (= number of connections to other peers)
ε	Number of join and leave events in the system per time unit
$e_{\mathcal{P}}$	Number of republish events of peer $\mathcal{P}$ per time unit
E	Number of republish events in the system per time unit
$f_{\mathcal{P}}$	Number of data items shared by peer $\mathcal{P}$
F	Number of shared data items in the system
G	Group size (= number of peers per group)
H	Highest load level (that can be observed across all superpeers)
k	Key (= identifier of a shared data item)
$\lambda_{\mathcal{P}}$	Load / load level of peer $\mathcal{P}$
Λ	System load (= mean value of the superpeers' load levels)
$l_{\mathcal{P}}$	Number of lookup messages processed by peer $\mathcal{P}$ per time unit
L_{SP}	Number of lookup messages processed by all superpeers in the system per time unit
$m_{\mathcal{P}}$	Number of maintenance messages processed by peer $\mathcal{P}$ per time unit
M_{SP}	Number of maintenance messages processed by all superpeers in the system per time unit
N	Number of peers in the system
N_{SP}	Number of superpeers in the system
N_{LN}	Number of leafnodes in the system
N_{NP}	Number of normal peers in the system
$q_{\mathcal{P}}$	Number of lookups initiated by peer $\mathcal{P}$ per time unit
Q	Number of lookups in the system per time unit
$r_{\mathcal{P}}$	Number of republish messages processed by peer $\mathcal{P}$ per time unit
R_{SP}	Number of republish messages processed by all superpeers in the system per time unit
τ	Number of transferred references in the system per time unit
T_{x}	Time period x
$z_{\mathcal{P}}$	Number of references stored on peer $\mathcal{P}$

List of Figures

List of Tables

Bibliography

Publications by the Author

[ESZK04] J. Eberspächer, R. Schollmeier, S. Zöls, and G. Kunzmann. Structured P2P Networks in Mobile and Fixed Environments. In *International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs)*, Ilkley, UK, 2004.

[KDM$^+$07] W. Kellerer, Z. Despotovic, M. Michel, Q. Hofstätter, and S. Zöls. Towards a Mobile Peer-to-Peer Service Platform. In *Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS)*, Hiroshima, Japan, 2007.

[KKSZ06] W. Kellerer, G. Kunzmann, R. Schollmeier, and S. Zöls. Structured Peer-to-Peer Systems for Telecommunications and Mobile Environments. *AEÜ - International Journal of Electronics and Communications*, 60(1):25–29, 2006.

[ZDK06] S. Zöls, Z. Despotovic, and W. Kellerer. Cost-Based Analysis of Hierarchical DHT Design. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Cambridge, UK, 2006.

[ZDK07] S. Zöls, Z. Despotovic, and W. Kellerer. Load Balancing in a Hierarchical DHT-based P2P System. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, White Plains, NY, USA, 2007.

[ZDK08] S. Zöls, Z. Despotovic, and W. Kellerer. On Hierarchical DHT Systems - An Analytical Approach for Optimal Designs. *Elsevier Journal of Computer Communication - Special Issue: Disruptive Networking with Peer-to-Peer Systems*, 31(3):576–590, 2008.

[ZETK06] S. Zöls, M. Eichhorn, A. Tarlano, and W. Kellerer. Content-based Hierarchies in DHT-based Peer-to-Peer-Systems. In *Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS)*, Phoenix, AZ, USA, 2006.

[ZHDK09] S. Zöls, Q. Hofstätter, Z. Despotovic, and W. Kellerer. Achieving and Maintaining Cost-Optimal Operation of a Hierarchical DHT System. In *IEEE International Conference on Communications (ICC) - Next Generation Networking Symposium*, Dresden, Germany, 2009.

[ZSH+05] S. Zöls, R. Schollmeier, Q. Hofstätter, A. Tarlano, and W. Kellerer. The Impact of Content Distribution on Structured P2P Networks in Mobile Scenarios. In *EUNICE Summer School*, Colmenarejo, Spain, 2005.

[ZSK05] S. Zöls, R. Schollmeier, and W. Kellerer. Making Chord Go Mobile. In *GI/ITG-Workshop "Peer-to-Peer-Systeme und Anwendungen"*, Kaiserslautern, Germany, 2005.

[ZSKD06a] S. Zöls, S. Schubert, W. Kellerer, and Z. Despotovic. Content Availability and Signaling Overhead in DHT Systems for Mobile Environments (Poster). In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, Denver, CO, USA, 2006.

[ZSKD06b] S. Zöls, S. Schubert, W. Kellerer, and Z. Despotovic. Hybrid DHT Design for Mobile Environments. In *International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, Hakodate, Japan, 2006.

[ZSKT05] S. Zöls, R. Schollmeier, W. Kellerer, and A. Tarlano. The Hybrid Chord Protocol: A Peer-to-Peer Lookup Service for Context-Aware Mobile Applications. In *International Conference on Networking (ICN)*, Réunion Island, France, 2005.

General Publications

[AAG+05] K. Aberer, L. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The Essence of P2P: A Reference Architecture for Overlay Networks. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Konstanz, Germany, 2005.

[ACMD+03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-Organizing Structured P2P System. *SIGMOD Record*, 32(3), 2003.

[BAS04] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *ACM SIGCOMM Conference*, Portland, OR, USA, 2004.

[BCM03] J. Byers, J. Considine, and M. Mitzenmacher. Simple Load Balancing for DHTs. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, 2003.

[BMR+06] A. Brampton, A. MacQuire, I. Rai, N. Race, and L. Mathy. Stealth Distributed Hash Table: A Robust and Flexible SuperPeered DHT. In *Conference on Future Networking Technologies (CoNEXT)*, Lisboa, Portugal, 2006.

[BS06] S. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *IEEE INFOCOM Conference*, Barcelona, Spain, 2006.

[BSH05] A. Binzenhöfer, D. Staehle, and R. Henjes. On the Fly Estimation of the Peer Population in a Chord-based P2P System. In *International Teletraffic Congress (ITC)*, Beijing, China, 2005.

[CC05] N. Christin and J. Chuang. A Cost-Based Analysis of Overlay Routing Geometries. In *IEEE INFOCOM Conference*, Miami, FL, USA, 2005.

[CCP02] R. Chakravorty, J. Cartwright, and I. Pratt. Practical Experience with TCP over GPRS. In *IEEE Globecom*, Taipei, Taiwan, 2002.

[CGH$^+$96] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth. On the Lambert W Function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[Coh03] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[CRB$^+$03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *ACM SIGCOMM Conference*, Karlsruhe, Germany, 2003.

[DLS$^+$04] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, 2004.

[GEBF$^+$03] L. Garces-Erice, E. Biersack, P. Felber, K. Ross, and G. Urvoy-Keller. Hierarchical Peer-to-Peer Systems. In *ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, Klagenfurt, Austria, 2003.

[GGG$^+$03] P. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *ACM SIGCOMM Conference*, Karlsruhe, Germany, 2003.

[GGGM04] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, 2004.

[GJSB05] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Third Edition.* Prentice Hall PTR, 2005.

[Gre02] M. Green. Napster Opens Pandora's Box: Examining How File-Sharing Services Threaten the Enforcement of Copyright on the Internet. *Ohio State Law Journal*, 63(2):799–819, 2002.

[HHB+03] R. Huebsch, J. Hellerstein, N. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *International Conference on Very Large Databases (VLDB)*, Berlin, Germany, 2003.

[HJ05] T. Herman and C. Johnen. Strategies for Peer-to-Peer Downloading. *Elsevier Information Processing Letters*, 94(5):203–209, 2005.

[HK03] H. Hsiao and C. King. Bristle: A Mobile Structured Peer-to-Peer Architecture. In *International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, 2003.

[HM03] K. Horowitz and D. Malkhi. Estimating Network Size from Local Information. *Information Processing Letters*, 88(5):237–243, 2003.

[JFY05] Y. Joung, C. Fang, and L. Yang. Keyword Search in DHT-Based Peer-to-Peer Networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Columbus, OH, USA, 2005.

[JMB06] G. Jesi, A. Montresor, and O. Babaoglu. Proximity-Aware Superpeer Overlay Topologies. In *IEEE International Workshop on Self-Managed Networks, Systems and Services (SelfMan)*, Dublin, Ireland, 2006.

[KK03] M. Kaashoek and D. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, 2003.

[KM05] K. Kenthapadi and G. Manku. Decentralized Algorithms using both Local and Random Probes for P2P Load Balancing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, Las Vegas, NV, USA, 2005.

[KR04] D. Karger and M. Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, USA, 2004.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *International Conference on Supercomputing (ICS)*, New York, NY, USA, 2002.

[LSMK05] J. Li, J. Stribling, R. Morris, and M. Kaashoek. Bandwidth-Efficient Management of DHT Routing Tables. In *Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, 2005.

[LV05] J. Li and S. Vuong. Ontology-Based Clustering and Routing in Peer-to-Peer Networks. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dalian, China, 2005.

[Man03] G. Manku. Routing Networks for Distributed Hash Tables. In *ACM Symposium on Principles of Distributed Computing (PODC)*, Boston, MA, USA, 2003.

[MBR03] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, WA, USA, 2003.

[MCKS03] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In *IEEE Workshop on Internet Applications (WIAPP)*, San Jose, CA, USA, 2003.

[MD04] A. Mislove and P. Druschel. Providing Administrative Control and Autonomy in Peer-to-Peer Overlays. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, USA, 2004.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, 2002.

[MMKG06] L. Massoulié, E. Merrer, A. Kermarrec, and A. Ganesh. Peer Counting and Sampling in Overlay Networks: Random Walk Methods. In *ACM Symposium on Principles of Distributed Computing (PODC)*, Denver, CO, USA, 2006.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *ACM Symposium on Principles of Distributed Computing (PODC)*, Monterey, CA, USA, 2002.

[Mon04] A. Montresor. A Robust Protocol for Building Superpeer Overlay Topologies. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Zurich, Switzerland, 2004.

[MRS00] M. Mitzenmacher, A. Richa, and R. Sitaraman. The Power of Two Random Choices: A Survey of Techniques and Results. In *Handbook of Randomized Computing: Volume 1*, pages 255–312. Kluwer, 2000.

[PKG$^+$05] D. Psaltoulis, D. Kostoulas, I. Gupta, K. Birman, and A. Demers. Practical Algorithms for Size Estimation in Large and Dynamic Groups. Technical report uiucdcs-r-2005-2524, University of Illinois at Urbana-Champaign, 2005.

[Pub02] Federal Information Processing Standards Publications. Secure Hash Standard (SHS). *FIPS PUB*, 180-2, 2002.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *ACM/IFIP/USENIX International Middleware Conference*, Heidelberg, Germany, 2001.

[RFH$^+$01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM Conference*, San Diego, CA, USA, 2001.

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowcz. Handling Churn in a DHT. In *USENIX 2004 Annual Technical Conference*, Boston, MA, USA, 2004.

[RIF02] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella Network. *IEEE Internet Computing Journal*, 6(1):50–57, 2002.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. IETF Request for Comments 1321, 1992.

[RLS$^+$03] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, 2003.

[RPW04] S. Rieche, L. Petrak, and K. Wehrle. A Thermal-Dissipation-based Approach for Balancing Data Load in Distributed Hash Tables. In *IEEE Conference on Local Computer Networks (LNC)*, Tampa, FL, USA, 2004.

[RV03] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, 2003.

[SAGPS05] M. Sánchez-Artigas, P. Garcýa, J. Pujol, and A. Góomez Skarmeta. Cyclone: A Novel Design Schema for Hierarchical DHTs. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Konstanz, Germany, 2005.

[SBR04] N. Sarshar, P. Boykin, and V. Roychowdhury. Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Zurich, Switzerland, 2004.

[SGG02] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, 2002.

[SMK$^+$01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM Conference*, San Diego, CA, USA, 2001.

[SMZ03] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *IEEE INFOCOM Conference*, San Francisco, CA, USA, 2003.

[SOTZ05] Y. Shu, B. Ooi, K. Tan, and A. Zhou. Supporting Multi-Dimensional Range Queries in Peer-to-Peer Systems. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Konstanz, Germany, 2005.

[Spi76] F. Spitzer. *Principles of Random Walk*. Springer, 1976.

[SRBS03] S. Singh, S. Ramabhadran, F. Baboescu, and A. Snoeren. The Case for Service Provider Deployment of Super-Peers in Peer-to-Peer Networks. In *Workshop on Economics of Peer-to-peer Systems (P2PECON)*, Berkeley, CA, USA, 2003.

[SS02] R. Schollmeier and G. Schollmeier. Why Peer-to-Peer (P2P) Does Scale: An Analysis of P2P Traffic Patterns. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Linköping, Sweden, 2002.

[SW05] R. Steinmetz and K. Wehrle. What Is This “Peer-to-Peer” About? In R. Steinmetz and K. Wehrle, editors, *Peer-to-Peer Systems and Applications*, pages 9–16. Springer, 2005.

[Wei91] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991.

[WGR05] K. Wehrle, S. Götz, and S. Rieche. Fundamentals of Distributed Hash Tables. In R. Steinmetz and K. Wehrle, editors, *Peer-to-Peer Systems and Applications*, pages 86–89. Springer, 2005.

[XMH03] Z. Xu, R. Min, and Y. Hu. HIERAS: A DHT Based Hierarchical P2P Routing Algorithm. In *International Conference on Parallel Processing (ICPP)*, Kaohsiung, Taiwan, 2003.

[YGM02] B. Yang and H. Garcia-Molina. Efficient Search in Peer-to-Peer Networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2002.

[YGM03] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *International Conference on Data Engineering (ICDE)*, Bangalore, India, 2003.

[ZHS+04] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

Cited Websites

[BD] P. Biondi and F. Desclaux. Silver Needle in the Skype. www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf.

[BEO] BEOLINGUS - Deutsch-Englisches Online-Wörterbuch.
http://www.beolingus.de.

[BIT] BitTorrent.
http://www.bittorrent.com.

[Cho] Y. Chou. Get into the Groove: Solutions for Secure and Dynamic Collaboration.
http://www.microsoft.com/technet/technetmag/issues/2006/10/IntoTheGroove.

[DIC] The National Puzzlers' League - "The Word Lists: 2nd Unabridged Dictionary".
ftp://puzzlers.org/pub/wordlists/unabr2.txt.

[FDB] FreeDB - Database to look up CD information using the Internet.
http://www.freedb.org.

[FTR] FastTrack.
http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=HEAD.

[G04] The Gnutella Protocol Specification v0.4.
http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[G06] Gnutella Protocol Version 0.6.
http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm.

[GNU] Gnutella.
http://rfc-gnutella.sourceforge.net.

[JME] Java Micro Edition.
http://java.sun.com/javame/index.jsp.

[KZA] KaZaA.
http://www.kazaa.com.

[Mic] Microsoft. Peer Name Resolution Protocol.
http://technet.microsoft.com/en-us/library/bb726971.aspx.

[MUT] The Mutella Homepage.
http://mutella.sourceforge.net.

[NAP] Napster.
http://en.wikipedia.org/wiki/Napster.

[NS2] The Network Simulator - ns-2.
http://www.isi.edu/nsnam/ns.

[PLA] PlanetLab.
http://www.planet-lab.org.

[SCR] The National Puzzlers' League - "The Word Lists: The Official Scrabble^{TM}Player's Dictionary".
ftp://puzzlers.org/pub/wordlists/enable1.txt.

[SKY] Skype.
http://www.skype.com.

[WTK] Java Wireless Toolkit.
http://java.sun.com/products/sjwtoolkit.