Exploiting Heterogeneity in Peer-to-Peer Systems Using Gradient Topologies

Jan Sacha

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

July 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

Jan Sacha

Dated: July 3, 2009

Acknowledgements

First and foremost, I am very grateful to my supervisors for their mentoring, expertise and support. I would like to thank Jim Dowling, who proposed my research topic and without whom this thesis would never have been written, René Meier, who supervised me in the middle of my studies, and Mads Haahr, who guided me during the difficult time of writing up. A special acknowledgement must be given to Raymond Cunningham, my *de facto* fourth supervisor, for his great help in my research. I am also grateful to my reviewers, Anthony Rowstron and Ciaran McGoldrick, for their valuable contributions to this work.

Over the course of my studies, I collaborated with a number of colleagues in the Distributed Systems Group (DSG). I would like to especially mention here Bartek Biskupski, Dominik Dahlem, Andy Edmonds, Serena Fritsch, Anthony Harrington, Marcin Karpiński, Dave McKitterick, Andronikos Nedos, As'ad Salkham, Aline Senart, Kulpreet Singh, Tim Walsh, and Stefan Weber. The DSG has created a phenomenal environment for working, learning and living. I would also like to thank my family, my parents, brother, and fiancee Aja, who continuously supported me despite the physical distance between us.

The research described in this thesis was partly funded by the Information Society Technology Programme of the European Union, Microsoft Research Cambridge, and the Irish Council for Science, Engineering and Technology (IRCSET). Their generous support is greatly appreciated.

Jan Sacha
University of Dublin, Trinity College
July 2009

Abstract

A peer-to-peer system can be defined as an overlay network built by a set of nodes on top of a physical network infrastructure and its operating protocols, such as the Internet. In a peer-to-peer network, each node maintains a limited number of connections with other nodes, called peers, and the graph of peer connections constitutes the overlay's topology. One of the most fundamental properties of existing large-scale peer-to-peer systems is a very high heterogeneity and dynamism of peers participating in the system. Studies show that the distributions of peer characteristics, such as peer session duration, available bandwidth, and storage space, are highly skewed and often heavy-tailed, with small fractions of peers possessing disproportionally large fractions of the total system resources. Such heterogeneity introduces both challenges and opportunities when designing peer-to-peer systems. The use of low-performance or low-stability nodes for maintaining system data or services can easily lead to a poor performance of the entire system, while the placement of critical data and services on the most reliable, high-capacity nodes may improve the overall system stability and performance.

Current state-of-the-art peer-to-peer systems exploit their heterogeneity by introducing two-level hierarchies of peers. High capability peers, so called super-peers, form an independent sub-topology within the existing peer-to-peer network and handle the core system functionality, such as indexing peer data and handling search queries, or relaying traffic on behalf of firewalled peers. Ordinary peers connect directly to super-peers and act as their clients. However, many existing systems lack an efficient, decentralised super-peer election algorithm. In many systems, super-peers are selected manually, through an out-of-band mechanism, or are elected using simple local heuristics, which are likely to generate suboptimal super-peer sets. Sophisticated super-peer election algorithms exist, but they are usually highly specific to particular systems and are not easily portable to other application areas.

This thesis presents a novel class of peer-to-peer topologies, called gradient topologies, which generalise the concept of super-peer networks. In gradient topologies, the position of each peer is determined by a continuous utility function, and the highest utility peers are clustered in a logical centre of the topology, while peers with lower utility are located at gradually increasing distance from the centre. The utility metric captures application-specific peer requirements and reflects peers' ability to contribute resources and services to the system. The gradient structure of the topology has two fundamental properties. Firstly, all peers in the system with utility above a given threshold are located close to each other in terms of overlay hops and form a connected sub-topology. Such high-utility peers can be exploited by higher level applications in a similar fashion to super-peers in traditional two-level hierarchies. Secondly, the information captured in the topology enables a search heuristic, called gradient search, that enables efficient discovery of such high utility peers.

The gradient topologies have been evaluated using a custom-built simulator and compared with state-of-the-art super-peer systems. The evaluation shows that the election techniques based on gradient topologies allow more flexible super-peer criteria specification compared with the other systems. Moreover, the super-peer sets elected using gradient topologies are closer to the theoretical optimum, compared with the other systems, and have a higher average utility and stability. The experiments also show that the maintenance cost of gradient topologies, in terms of generated messages and established connections, is similar to that in the state-of-the-art systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this chapter is to give a general introduction to the area of peer-to-peer system and to the problems discussed in this thesis. It describes briefly the history of peer-to-peer systems and discusses the main characteristics found in these systems. It is shown that existing large-scale peer-to-peer systems are highly heterogeneous. This chapter also describes the thesis goals, explains the motivation behind these goals, and outlines the thesis organisation.

1.1 Peer-to-Peer Systems

Peer-to-Peer (P2P) systems belong to the fastest growing applications in the area of distributed computing. Within a short number of years, P2P systems gained an extreme popularity, attracted millions of users, entered a number of different application areas, and became one of the main contributors of global traffic in computer networks.

Although the first widely-used systems considered P2P appeared already in 1970's, with Usenet and the Network News Transfer Protocol (NNTP) being examples of such systems, most of the modern P2P applications have been invented during the last decade. Particularly, the first application that gained an extraordinary popularity was Napster, a file-sharing application developed and published in 1999. By the end of 2000, Napster had been downloaded by 50 million people around the world and became the fastest growing application on the Web [149]. The success of Napster was quickly followed by other file-sharing P2P systems, such as Gnutella, KaZaA (also known as FastTrack), DirectConnect, eDonkey, Overnet, BitTorrent and many others. Soon, P2P systems became one of the two most dominant applications on the Internet, in terms of global traffic contribution, along with the Web. On a typical day, KaZaA has more than three million active users, sharing over 5,000 terabytes of content [102]. According to the measurements by Tier 1 Internet providers, P2P applications generate between 15% and 20% of the total Internet traffic, and up to 60% of traffic on some Internet backbone links [57, 87]. Moreover, a number of regional Internet service providers, such as university network operators or national operators, estimate that P2P applications account for approximately 30-70% of total generated traffic, and their share is still growing [12, 11, 111, 53].

Freenet [40, 39], which appeared in 1999, was another system that pioneered the development of P2P networks. Freenet was a distributed storage system that allowed for publication, replication, and retrieval of data, while protecting the anonymity of both authors and readers. Along with file sharing and file storage, P2P systems entered a number of other application areas, such as Internet telephony, video conferencing, and multimedia streaming [37, 32, 84, 33, 92, 140, 204]. P2P systems were also used for content distribution [41], for example by the Debian organisation for distributing Linux distribution image files. An extremely large popularity was achieved by Skype [15, 67], an Internet telephony and conferencing application made available in 2003. In 2006, Skype was reported to have 83 million registered users around the world and was used by approximately 3-4 million simultaneous users at any one time [35].

Another initiative that contributed to the development P2P systems was SETI@home [8, 194], a scientific project launched in 1999, whose aim was to exploit unused computing resources, such as CPU idle time, on machines connected to the Internet for running scientific computations, such as analysing radio telescope signals in search of intelligent life outside Earth. Although SETI@home is not generally considered a P2P system, since its nodes communicate with a centralised server rather than with each other, it shares a number of common characteristics with P2P systems, and belongs to a wider class of distributed systems known as Public Resource Computing (PRC) or volunteer computing. Both P2P and PRC systems rely on large and dynamic populations of autonomous nodes that contribute resources to the system.

The SETI@home project received a strong public response, and by 2002, over 3.8 million people registered in the program. This enabled an unprecedented computing power (order of 100 TeraFLOPS), and by 2002 the program performed a total of 1.7e21 floating-point operations, the largest computation on record at that time [8].

As the popularity of P2P grew, P2P systems also became a significant research area within the greater domain of distributed systems. A number of conferences and scientific journals have been devoted solely to P2P systems and their applications, and a large body of work has been published in the area.

A number of independent definitions for P2P system have been proposed [171, 10, 2, 167]. Most definitions describe P2P systems as large-scale, decentralised systems maintained by a large number of autonomic nodes, called peers, which voluntarily contribute resources, such as storage space, processor cycles, bandwidth, or physical human presence, and self-organise in order to provide a useful services to a community of users. The membership in P2P systems is open and dynamic, as peers can freely join and leave during the system's operation. There is no distinction between dedicated servers and clients, as all nodes participating in the system provide services to each other as peers. Every peer can function as both a client and a server. Some definitions also assume that the responsibilities and capabilities of all peers in a P2P system are identical [162, 44, 176].

Due to the scale and dynamism of a P2P system, it is not feasible for each peer to possess and maintain an accurate model of the entire system. Instead, each peer usually has a knowledge of and communicates with a limited number of other peers, called its neighbours. The connections between neighbouring peers form an overlay network, on top of the physical infrastructure, such as the Internet. This overlay network is used by peers for routing messages and exchanging information. Unlike in wireless networks, where the communication range is limited, the nodes in P2P systems are directly reachable and the overlay structure can be adapted arbitrarily depending of the system needs. The graph of peer connections is called the system topology.

1.2 Heterogeneity in Peer-to-Peer Systems

Contrary to the definitions that assume identical roles and capabilities of all peers in the system, measurements show that deployed P2P systems are characterised by a very high diversity of participating peers. The distributions of basic peer characteristics, such as the processing power, storage space, bandwidth, or session duration, are highly skewed and radically different from the normal or uniform distributions. In the normal distribution, practically all values are located within several standard deviations from the mean (e.g., 99.7% of all values are within three standard deviations from the mean) and values beyond this range are extremely unlikely. In P2P systems, it has been observed that the characteristics of peers (i.e., storage, bandwidth, etc.) vary by several orders of magnitude between individual peers (see Table 1.1 for a comparison). Furthermore, while a large majority of peers have relatively few resources, small subsets of peers possess significant fractions of the total system resources. Such distribution is often modelled using the Pareto distribution, also known as the Zipf's law and power law [135, 4], and other heavy-tailed distributions, such as the Weibull and log-normal distributions.

One of the basic peer properties that shows a high P2P system heterogeneity is the session duration, defined as the amount of time a peer stayed (or is expected to stay) on-line in the system without disconnecting. According to a number of independent measurements, the average session duration in existing P2P systems is relatively short, and varies between 1 minute and 1 hour, depending on the system and the measurement method [181, 144, 27, 36, 167]. However, in nearly all studied systems, groups of peers were found which stayed on-line for considerably longer periods. For example, Stutzbach et al. [181] observe that roughly 10%-20% of peers in Gnutella and Kad have an uptime (i.e., amount of time since joining the system) longer than one day, and around 1-3% of BitTorrent peers have an uptime longer that two weeks. Similarly, Pouwelse et al. [144] report that 17% of

BitTorrent peers stay in the system for longer than one hour after they finished downloading, 3.1% of peers stay on-line for at least 10 hours after downloading, 0.34% of peers stay for more than 100 hours after downloading, and their longest observed session is 83.5 days.

Related to the peer session duration is peer availability, a property defined as the fraction of time a peer spends on-line in the system within a longer period. Measurements indicate that peer availability ranges from almost 0% to 100% between peers, and the availability distribution is highly skewed [173, 18]. The majority of peers have a poor availability, while small subsets of peers stay on-line almost all the time. Furthermore, while some peers frequently join and leave the system over time, other peers connect to the system only once and never come back. For instance, Bhagwan et al. [18] discover that in a two-week trace collected from the Overnet system, on each day, new hosts never seen before in the trace comprise over 20% of the peer population. A number of experiments reveal also diurnal patterns in the peer participation in P2P systems [18, 167, 36, 67, 27].

Bandwidth is another example of an unevenly distributed resource between peers. Although technically bandwidth is a property of a network connection between two machines, in practice the bottleneck bandwidth between a peer and the rest of the Internet is determined by the peer's direct link to the Internet [97, 167], and hence, is a property of this peer. Saroiu et al. [167] show that the median upstream bottleneck bandwidth of peers in Gnutella is roughly 1Mpbs, while about 22% of peers have a bottleneck bandwidth below 0.1Mbps, 8% of peers have a bottleneck bandwidth above 10Mbps, and the highest observed bandwidth capacities reach 100Mbps. Similarly, Pouwelse et al. [144] show that while the average download speed of a peer in BitTorrent is 240 Kbps, a number of peers download at much higher rates, with a maximum around 4,000 Kbps. A number of other measurements show consistent results [9, 173, 188].

Other peer properties that have been analysed and shown to follow skewed distributions include peers' computing power and available storage space and memory (RAM) [9]. Moreover, it has been shown that a significant fraction of peers in P2P systems are located behind firewalls or Network Address Translators (NAT) which limit their ability to communicate with other peers [102, 181]. The types of peers' firewalls or NATs add one more dimension to the diversity of peers and hence to the heterogeneity of P2P systems.

Apart from hardware parameters, peers also very between each other in terms of their behaviour. An important peer characteristic is willingness to share resources. Studies show that the number of files shared by Gnutella peers vary between 0 and 10,000 [167, 208]. In particular, a large fraction of users (so called free riders, up to 70% of all users) share no files [5]. Some Gnutella users also take steps to discourage other users from downloading files from them, for example by advertising the lowest possible upload speed (64 Kbps or less) [167]. On the other hand, groups of peers have been observed which exhibit a contrary behaviour. These peers do not download any files but stay on-line and let other peers download from them. It is estimated that 7% of peers in Gnutella together offer

Property	Measurement	Year	System	Value Range
Session duration	Stutzbach [181]	2005	Gnutella, BitTorrent, Kad	1min – 1day
	Pouwelse [144]	2004	BitTorrent	0.1h – 1000h
	Bustamante [27]	2003	Gnutella	20min – 3days
	Chu [36]	2002	Napster, Gnutella	10min – 10,000min
	Saroiu [167]	2001	Napster, Gnutella	0 – 720 min
Availability	Kutzner [95]	2004	Overnet	1 – 192 h*
	Bhagwan [18]	2003	Overnet	1 – 24 h/day
	Sen [173]	2001	FastTrack, Gnutella, DirectConnect	0 – 1440 min/day
Bandwidth	Anderson [9]	2006	SETI@home	20Kbps – 8Mbps
	Pouwelse [144]	2004	BitTorrent	0 – 4 Mbps
	Tutschku [188]	2003	eDonkey	100bps – 1Mbps
	Saroiu [167]	2001	Napster, Gnutella	10Kbps – 100Mbps
	Sen [173]	2001	FastTrack, Gnutella, DirectConnect	10KB/s – 10MB/s
Disk space	Anderson [9]	2006	SETI@home	0.1 – 512 GB
CPU	Anderson [9]	2006	SETI@home	0 – 4,000 Mflop/s
Memory	Anderson [9]	2006	SETI@home	0.1 – 512 GB
Shared files	Zhao [208]	2005	Gnutella	10 – 10,000
	Saroiu [167]	2001	Napster, Gnutella	0 – 10,000
	Adar [5]	2000	Gnutella	1 – 10,000
Downloads	Leibowitz [98]	2003	KaZaA	1 – 1000

(*) within a two-week period

Table 1.1: Heterogeneity in peer-to-peer systems.

more files than all of the other peers combined [167].

The amount of traffic generated by individual peers is also extremely variable. Sen et al. [173] analyse network traffic in a large Tier 1 Internet provider and show that less than 10% of peer IP addresses contribute around 99% of the total traffic volume, and the top 1% of peer addresses transmit 73% of the total traffic. A single peer may transmit over 10GB of data during a one day. Similarly, Leibowitz et al. [98] observe that while the majority of KaZaA peers initiate less than 10 downloads during their life time, a number of peers request several hundred files. Such highly active peers, generating relatively large amounts of traffic, are often called heavy-hitters.

Table 1.1 shows a summary of measurements of peer characteristics in deployed P2P systems.

1.3 Motivation

One of the main challenges in building P2P systems is dealing with the difficult substrate on which these systems are based – the heterogeneous and dynamic population of peers discussed in section 1.2. Since the average peer session duration in a P2P system is short, large numbers of peers continuously join and leave the system. This process is often referred to as *churn*, and the rate of peer arrivals, which is approximately equal to the rate of peer departures in the long run, is also called churn rate [181, 153].

In the presence of high churn, the entries in peers' neighbourhood tables become quickly outdated and may point to peers that no longer participate in the system [93]. This either increases message loss rates, since messages routed using these entries are not delivered, or increases delays in communication, as message failure detection and retransmission is usually slow. In order to keep neighbourhood tables up to date, peers need to frequently exchange messages with their neighbours. However, this increases the overhead related to the overlay network maintenance [113, 100]. For example, it is estimated that in the early versions of Gnutella, keep-alive messages accounted for over 50% of all generated traffic [155]. As the churn rate increases, the system also needs to increase the level of data redundancy (e.g., the replication factor) in order to guarantee a certain level of data availability. This again increases the overlay maintenance cost, since more data need to be transferred when peers are joining and leaving the system and more replicas need to be synchronised when updates are issued [17]. Consequently, a number of P2P systems have been shown to perform poorly in the presence of high churn [153].

Furthermore, P2P systems may suffer poor performance if they do not address their heterogeneity and do not adapt their structure to the properties of individual peers. The lowest performance peers, with poor processing capacity or insufficient network throughput, are likely to become bottlenecks. For example, in August 2000, the entire Gnutella network experienced deteriorated performance, with slow response time and fewer available resources, as it grew to a larger size. This was caused by peers connected by dial-up modems, which became saturated by the increased load and caused network fragmentation. Indirectly, this was caused by Gnutella's lack of ability to control its topology and to adapt it to the capabilities of individual peers [200].

However, the heterogeneity of P2P systems, as well as being a challenge, is also an opportunity that can be exploited. By assigning more responsibilities to stable, high capability peers, a P2P system may improve its overall reliability and performance [167, 110]. The subset containing the most stable peers is less subject to churn, and hence, is more suitable for hosting data or routing traffic [181, 27]. Likewise, a set of peers with the largest amount of resources is most suitable for performing resource-intensive tasks, such as handling search queries [34].

Consequently, nearly all widely used P2P systems today attempt to exploit the diversity of participating peers. In most systems, peers are divided into two categories. The highest capability peers, called *super-peers*, act as servers to the other peers. Usually, they form an independent sub-topology within the system overlay and handle the core system functionality. Ordinary peers maintain connections to selected super-peers and act as their clients [200]. For example, KaZaA uses super-peers (called *supernodes*) to index data stored by clients and to handle the search protocol [102]. A similar concept has been employed by Gnutella, where *ultrapeers* are used for routing search queries [177], and in eDonkey, where *eDonkey servers* are used by client peers as rendezvous points [188, 75]. In Skype, *supernodes* are mainly used for relaying messages between peers whose Internet access is restricted by a firewall or a Network Address Translator (NAT) [15, 67]. While different systems and research documents introduce their own vocabulary for describing peers, they refer in principle to the same general concept of distinguished peers that have higher capabilities and perform more tasks than ordinary peers. For consistency, this thesis uses the term *super-peers* for describing such distinguished peers, and the terms *ordinary peer*s and *client*s for the remaining peers. In most contexts, the terms super-peer, ultrapeer, supernode, and superpeer can be treated as synonyms.

There are other approaches to exploiting heterogeneity in P2P systems, in which the system structure is adapted to the properties of participating peers, but no division between super-peers an clients is made. Such approaches include spanning tree and mesh structure optimisations in streaming systems [150, 6, 139, 112, 21, 168], topology and message flow adaptation in Gnutella [110, 34, 27], and the introduction of virtual servers to distributed hash tables [148, 88]. However, these approaches are highly specific to their application areas, and hence are not as universal as the super-peer based approaches.

The design based on super-peers has two main advantages. First, certain system tasks, such as hosting data or services, can be assigned to stable, high-performance peers, i.e., super-peers, in order to improve the overall system reliability and performance. Second, the use of super-peers allows the system to limit the number of participants in certain distributed algorithms, such as search [109, 199, 187, 99], which do not scale well and become too expensive when the system size is large. Thus, the super-peer design can improve the scalability of a P2P system.

However, the use of super-peers introduces a number of new challenges that need to be addressed. In order to elect super-peers, the system needs to decide on how many super-peers are needed and which peers are most appropriate to take the role of super-peers. Furthermore, the system needs to maintain and continuously adjust the super-peer set in response to peer arrivals and departures, changes in the current load and changes in peer capabilities. The system also needs to distribute clients between super-peers and migrate them when super-peers leave or fail. Ideally, the load between super-peers should be balanced to ensure the system's scalability and fault-tolerance. This must all be performed in a dynamic and decentralised manner, with no central coordination or authority.

It should be noted that traditional election algorithms for distributed systems, such as the bully algorithm [60], and classic approaches to group communication [156, 73, 19], are not applicable to large-scale P2P systems due to their cost and message overhead. Most of these approaches require strong global consensus between all peers in the system [20] and rely on broadcasting messages between all peers in the system [60, 70]. As such, they can be only applied to small systems (order of thousands of nodes [70]) or smaller subsets of peers within a larger system.

Few existing P2P systems attempt to elect super-peers in an efficient, decentralised way. Many P2P systems rely on manual super-peer selection, through an out-of-band mechanism, or employs simple heuristics, which are likely to generate suboptimal super-peer sets. In some systems, the addresses of super-peers are simply hardcoded into the application. Only few P2P systems attempt to elect optimal super-peers sets according to some metric, but most of these systems are specific to particular applications and their super-peer election mechanisms are not easily portable to other domains.

1.4 This Thesis

This thesis describes a novel approach to dealing with heterogeneity in P2P systems. This approach is based on peer *utility metrics* and *gradient topologies*. A utility metric is a function evaluated at each peer locally that reflects peer's ability to contribute resources to the system and to provide services to other peers. A utility metric is domain-specific, and captures peer requirements imposed by the higher-level application built on top of the gradient topology.

A gradient topology is a P2P topology where the highest utility peers are clustered in the logical centre of the topology, while peers with lower utility are found at gradually increasing distance from the centre. In contrast to super-peer topologies or hierarchies, where peers are divided into two or more discrete groups, gradient topologies introduce a continuous spectrum of peers, from the highest utility peers in the centre to the lowest utility peers at the periphery.

Gradient topologies have an elementary property that for any given *utility threshold*, all peers in the system with utility above this threshold are located close to each other, in terms of overlay hops, and form a connected sub-topology within the system overlay. Such high utility peers can be exploited by the system in a similar way as super-peers in traditional two-level hierarchies. Furthermore, high utility peers in gradient topologies can be easily and efficiently discovered by lower utility peers using an heuristic called *gradient search*, which routes messages from outer peers towards the centre of the topology, as in hill climbing and similar techniques based on the notion of gradient.

The main advantage of gradient topologies over the traditional super-peer topologies is that utility thresholds can be increased or decreased, adjusting the number of peers above the thresholds according to the system requirements, without the need to reconfigure any peer connections. For any selected threshold, peers above the threshold are clustered at the centre of the gradient topology and gradient search can be used by low utility peers to efficiently discover them. Moreover, without any additional mechanisms, multiple thresholds can be calculated to elect multiple concentric sets of high utility peers, similar to a hierarchy.

Gradient topologies, together with election algorithms described in this thesis, allow P2P system to select the most suitable peers for performing tasks such as hosting system data, running services, or participating in certain distributed algorithms. By selecting the highest utility peers in the network for these tasks, P2P systems can exploit the heterogeneity in peer populations for improving the overall system stability and performance.

Gradient topologies are domain-independent and can support many different classes of P2P applications, such as storage systems, name services, file-sharing applications, and semantic registries. As gradient topologies are based on the notion of *peer* utility rather than peer *connection* utility, they are not directly applicable to P2P systems that need to adapt their structures to the properties of the underlying low-level network, such as link latencies and link throughputs. Systems that belong to this category, for example multi-media streaming applications, are not addressed in this thesis.

The main contributions of this thesis are: (i) a number of utility metrics for the characterisation of peers in a P2P system, (ii) neighbour selection algorithms that generate and maintain gradient topologies with desired properties, such as a low degree of peers and a low distance between the highest and the lowest utility peers, growing logarithmically with the system size, (iii) election algorithms, based on decentralised aggregation techniques and adaptive utility thresholds, which create and manage close-to-optimal super-peer sets in the gradient topologies and minimise the number of swappings between super-peers and ordinary peers, (iv) routing heuristics that enable high-utility peer discovery in the gradient topologies.

The proposed algorithms have been validated using a custom-built P2P simulator. In a range of experiments, it is shown that gradient topologies, together with aggregation-based election techniques, generate better-quality and higher-stability super-peer sets, and have similar maintenance cost, compared with state-of-the-art super-peer systems. Moreover, it is shown that gradient topologies offer more flexible and powerful super-peer election mechanisms compared with the existing P2P systems, and thus extend the current state-of-the-art knowledge on super-peers, and more generally, heterogeneity exploitation in P2P systems.

1.5 Thesis Roadmap

The remainder of this thesis is organised as follows.

Chapter 2 reviews a wide range of P2P systems that introduce super-peers, with a particular emphasis on the super-peer criteria and election mechanisms, highlighting the achievements and limitations of each system.

Chapter 3 formally defines the class of gradient P2P topologies and describes their main characteristics.

Chapter 4 presents a collection of utility metrics and algorithms that generate gradient topologies and enable super-peer election and discovery in these topologies.

Chapter 5 evaluates the algorithms introduced in chapter 4 and verifies that they construct topologies defined in chapter 4. It also compares the functionality and performance of gradient topologies with a number of state-of-the-art super-peer election systems.

Chapter 6 summarises the thesis and discusses future work.

Chapter 2

State of the Art

This chapter surveys the area of P2P systems that exploit the diversity in peer characteristics. In the large majority of cases, these systems are based on super-peers. For each reviewed system, the super-peer functions, election mechanisms, and topology maintenance algorithms are described, and the advantages and limitations of each proposed approach are discussed. The last section in this survey covers systems that do not use super-peers but are based on alternative principles.

Systems that exploit heterogeneity between peer *connections* rather than individual peers are not covered in the survey. These systems include in particular streaming and multicasting applications, which optimise dissemination trees, mesh overlays, and other structures based on the latency, bandwidth, and throughput of connections between peers [150, 6, 139, 112, 21, 168]. Due to the different requirements and objectives, these systems cannot be directly compared to the gradient topologies and election strategies described in this thesis.

2.1 Super-Peer Systems

The concept of super-peers has been first studied by Yang and Garcia-Molina in [200], who divide P2P systems into three categories. In *pure* P2P systems, such as Freenet and initial versions of Gnutella, all peers have equal roles and responsibilities in all aspects, and the system's functionality is fully decentralised. In *hybrid* systems, such as Napster, some functionality is handled by a centralised component (e.g., search), but otherwise the system is decentralised (e.g., downloads are performed directly between peers). *Super-peer networks,* such as KaZaA and early versions of Morpheus, present a cross between pure and hybrid P2P systems.

A *super-peer* is a node that acts as a centralised server to a set of clients and as an equal to other super-peers. Clients communicate with their super-peers as with servers in hybrid P2P systems or in traditional client-server architectures. However, super-peers are also connected to each other as peers

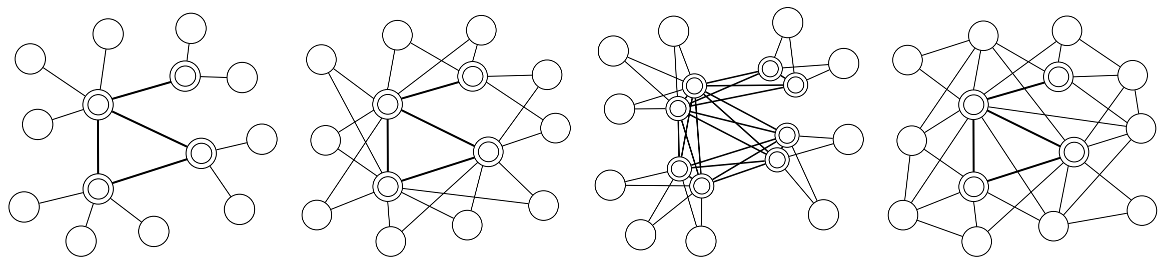

(a) Classic super-peer topology
(b) Redundant client connections
(c) Redundant super-peers
(d) Embedded super-peer topology

Figure 2.1: Four variants of super-peer topologies.

in pure P2P systems, forming a super-peer overlay that handles the core system functionality, such as search. A super-peer together with the set of its clients is called a *cluster,* and the *cluster size* is the number of nodes in the cluster, including the super-peer.

In a sense, the introduction of super-peers enables a trade-off between full decentralisation and partial centralisation in a P2P system. The advantage of super-peer systems over hybrid P2P systems is that they do not have any centralised components. The advantage of super-peer systems over pure P2P systems is that they can exploit the heterogeneity in peers by assigning relevant system functions to high-capability peers and that they reduce the number of participants in expensive algorithms, such as search, by running these algorithms on super-peers only.

2.1.1 Super-Peer Topologies

Super-peer topologies found in the existing P2P systems and literature can be divided into four general types. In classic super-peer topologies, every client is connected to exactly one super-peer, as shown in Figure 2.1(a). These topologies, however, have the drawback that in the case of a super-peer failure, all clients of the failed super-peer become disconnected from the network and isolated.

In order to address this problem, some systems allow clients to maintain connections to multiple super-peers, as shown in 2.1(b). Similarly, for improved system reliability and fault tolerance, Yang and Garcia-Molina [200] introduce a *k-redundant* super-peer topology, where each super-peer, called a *virtual super-peer*, is replicated on k physical peers. Each of the k physical peers maintains connections to all neighbours of the virtual super-peer (clients and other super-peers) and hosts a full copy of the super-peer data. In this way, a virtual super-peer can tolerate up to $k-1$ peer failures without service disruption. A sample 2-redundant super-peer topology is shown in Figure 2.1(c).

Finally, in some systems, any two peers are allowed to establish a connection between each other, including two clients, in which case the super-peer topology is embedded in the overall system topology, as shown in Figure 2.1(d). Such a topology is also known as a hub topology [176].

2.1.2 Super-Peer Election

In order to introduce super-peers, a P2P system needs to decide on how many super-peers are desired and which peers are the best candidates for super-peers. These two problems are correlated, as the desired number of super-peers may depend on the properties of available peers. For example, fewer super-peers may be elected if high-capacity candidates are available in the system, while more super-peers may be required if all peers have low capacity.

As the system elects super-peers, it needs to decide on how to connect the super-peers with each other and how to assign clients to super-peers. The connections between super-peers, as well as the super-peer functionality, are usually application-specific. However, the algorithms for super-peer election and client distribution are often application-independent, and given an optimality criteria, can be directly compared with the election algorithms used in gradient topologies. For that reason, a special emphasis is put on the super-peer election mechanisms used in the systems reviewed in this chapter. Systems with the more elaborated super-peer election algorithms are covered in more detail.

The reviewed systems are divided into four loose categories based on the super-peer election mechanisms. The proposed organisation is not an exhaustive formal taxonomy, and some systems may fit in more than one category. The classification is introduced for convenience only.

The first category, described in section 2.2, comprises systems that do not specify any super-peer election algorithm or have a very simple super-peer election approach. This includes manual or centralised super-peer selection, and selection performed locally at each peer based on a static criteria. Many of the first P2P systems that introduced super-peers belong to this category.

The second class, covered in section 2.3, consists of systems where the population of peers is divided into disjoint groups, and super-peers are elected within each group independently. The groups are usually based on peer properties such as physical location, network proximity, or semantic content.

The third class (section 2.4) contains systems where the super-peer election method is based on a Distributed Hash Table (DHT). The DHTs [179, 149, 162, 46] are a well-known class of P2P systems, with a well-defined functionality (i.e., that of a hash table), which constitutes a coherent subset of P2P systems. Due to the distinctive characteristics of the DHTs, systems that use DHTs for the election of super-peers are considered as a separate class in this review.

Finally, the last category (section 2.5) contains adaptive systems that elect super-peers based on global demand, for example defined as the number of clients, rate of client requests, or current load on super-peers. These systems usually define some optimality criteria and continuously strive to optimise the super-peer set. Table 2.1 lists the systems reviewed in each of the four categories.

Simple Approaches	Group-Based	DHT-Based	Adaptive
2.2.1 OceanStore	2.3.1.1 Crown	2.4.3 SOLE	2.5.1 SG-1
2.2.2 Brocade	2.3.1.2 PASS	2.4.4 Hierarchical DHT	2.5.2 SG-2
2.2.3 Gnutella	2.3.1.3 PoPCorn	2.4.5 HONets	2.5.3 DLM
2.2.4 KaZaA	2.3.1.4 Wolf and Merz	2.4.6 SPChord	
2.2.5 Gnutella 0.6	2.3.2.1 Edutella	2.4.7 Mizrak et al.	
2.2.6 eDonkey	2.3.2.2 ROSA-P2P		
2.2.7 Yang et al.	2.3.2.3 Qiao et al.		
2.2.8 Skype	2.3.3.1 Mastroianni et al.		
2.2.9 JXTA	2.3.3.2 Glare		
2.2.10 LST			
2.2.11 SBARC			
2.2.11 Zhu et al.			
2.2.12 SUPS			
2.2.13 Schelling			

Table 2.1: Systems covered in the review.

2.2 Simple Approaches to Super-Peer Election

This section reviews P2P systems with the simplest super-peer election and management mechanisms. This category includes systems that do not specify any super-peer election method, or rely on external, out-of-band mechanisms, or leave the super-peer election to higher-level applications; systems where the super-peer sets are hardcoded; systems that use centralised components for handling super-peer management; systems where super-peers are selected manually, either by a global system administrator or by the local user at each peer; systems where super-peers are elected based on fixed threshold, such as minimum amount of bandwidth or storage space. A number of research papers reviewed in this section assume that a certain super-peer topology is given and focus on the exploitation of such a topology. The creation and maintenance of the super-peer topology is treated in these papers as a separate research topic. Moreover, a large number of papers specify super-peer election criteria, usually as a combination of peer characteristics such as bandwidth, storage space, or processing power, but do not elaborate on any super-peer election algorithm.

The reviews begin with OceanStore, one of the first P2P systems that postulated the use of super-peers, and Brocade, a system inspired by OceanStore. Next, the section describes a number of file-sharing systems that rely on super-peers, including KaZaA and Gnutella. Finally, the review covers Skype, an Internet telephony system, and a number of general P2P frameworks and algorithms such as JXTA and the Schelling algorithm.

2.2.1 OceanStore

One of the first systems that proposed the use of super-peers, published in 2000, was OceanStore [94, 152], a global-scale distributed storage system for persistent data. OceanStore can be seen as a predecessor of P2P systems, as it was designed to run on a large number of nodes distributed around the world and maintained by multiple independent providers. The nodes in OceanStore are considered unreliable and untrusted, as in P2P systems.

For reliability and performance reasons, data stored in OceanStore is replicated and spread evenly between nodes using a proactive replication algorithm. Every data object has a primary replica, which serialises updates, verifies access control credentials, and constructs a dissemination tree between secondary replicas. The primary replicas are hosted on a selected set of nodes, called *primary tier* or *inner ring*. These nodes function in OceanStore in a similar way to super-peers in many P2P systems.

As stated in [94], "the primary tier consists of a small number of replicas located in high-bandwidth, high-connectivity regions of the network". However, OceanStore does not provide any algorithm for the election of nodes in the primary tier. According to the replication protocol specification [62], nodes that participate in the inner ring are selected manually by system operators.

The data placement and discovery in OceanStore is controlled by a variation of the Plaxton algorithm [143], which later evolved into the Tapestry [207], one of the first distributed hash table systems. In this algorithm, every data item, as well as every peer in the system, is assigned a unique identifier. Data items are assigned to peers based on their identifiers and independently of their physical locations. Peers maintain a system topology and routing tables that enable efficient query routing and data access. Both Tapestry and the original Plaxton algorithm spread data and traffic equally between all peers in the system, and treat all peers in the system as if they possessed uniform resources.

2.2.2 Brocade

In order to address peer heterogeneity and to improve routing performance in OceanStore, an extension to Tapestry has been proposed called Brocade [206]. In Brocade, high-capability super-peers, called *landmark* nodes, are used for routing messages through wide-area networks on behalf of other peers that act as their clients. Landmark nodes maintain a Tapestry network between each other and use the Tapestry routing algorithm. Landmark nodes also maintain lists of their clients.

Brocade elects super-peers from nodes that "have significant processing power, minimal number of IP hops to the wide-area network, and high-bandwidth outgoing links" [206]. Amongst the nodes that satisfy these requirements, super-peers are selected by the Internet Service Providers (ISP) in each local network. According to [206], "gateways routers or machines close to them are attractive candidates" for super-peers. An election algorithm is mentioned, but no details of such an algorithm are provided.

Brocade also includes two mechanisms that associate clients with super-peers. In the first approach, super-peers monitor the traffic in their local networks and intercept all messages destined to peers located in remote networks. These messages are subsequently tunnelled and routed over the Tapestry overlay by the super-peers. However, this approach requires that every local network in the system must have at least one super-peer and the network must be configured in such a way that the super-peers can intercept messages from all local peers. Such a requirement may be a serious obstruction in the system deployment.

The second method relies on the use of predefined names in the Domain Name System (DNS) to identify super-peers. Every super-peer, when elected, binds its address to a fixed name in the local DNS domain. A client can discover its super-peer by resolving the fixed name in its own local DNS domain. If no super-peer is found, the client can become a super-peer itself. However, this approach again requires that at least one super-peer must be created for every local DNS domain in the system, and furthermore, super-peers must be allowed to alter their DNS domains in order to register.

2.2.3 Gnutella

Amongst the first P2P systems that introduced super-peers were file-sharing applications. In these systems, each peer specifies a collection of files that it agrees to share with other peers and each peer can download potentially any file made available by other peers in the system. Most file-sharing systems also provide a search facility that allows peers to discover files shared by other peers that satisfy certain search criteria. In Napster, search was provided by a centralised server that kept track of all files hosted by peers in the system. However, most P2P file-sharing systems provide a decentralised search facility. In these systems, a peer performs search by generating a search query. The query is propagated in the overlay, and peers that store files that match the query reply back to the searching peer.

Early file-sharing systems, such as the first versions of Gnutella, commonly used flooding to propagate queries between peers. Other search techniques often used include random walks, Breadth First Search (BFS), Depth First Search (DFS), and iterative deepening [199, 109, 187]. However, these search strategies, sometimes called *blind* search techniques, have the drawback that they need to disseminate search queries potentially to all peers in the system in order to find matching results. As a consequence, when the size of the system grows and more search queries are generated, an average peer receives more search messages. At certain point, the system does not scale as the search overhead becomes prohibitive.

Gnutella addressed this problem by restricting the maximum number of times a query could be forwarded, limiting the scope of search queries. However, this introduced the problem known as the *search horizon* – peers were able to discover only a subset of files available in the system. Furthermore, Gnutella suffered poor performance due to the fact that the slowest peers in the network easily became overloaded, as the system did not adapt its topology to the peer capabilities.

2.2.4 KaZaA

The scalability problems encountered in Gnutella were solved in KaZaA, the first P2P file-sharing application that introduced super-peers for handling search [102, 200].

In KaZaA, peers are divided into two classes – high-capability super-peers (called *supernodes*) that handle search, and ordinary peers that act as their clients. Each super-peer maintains an index of all files stored on its clients. The search protocol in KaZaA is called FastTrack. A client performs search by submitting a search query to its super-peer. The super-peers disseminates the query to other super-peers, and as it receives results from other super-peers, it forwards the results to the client.

This design has two advantages over traditional search algorithm for P2P systems. First, it limits the number of peers that participate in the search protocol, as search messages are exchanged between super-peers only. This way, search is quicker, less expensive, and more scalable. Second, it prevents low-capability peers from being overloaded, and further improves search performance, as search is handled by selected, high-capability peers.

However, it is not precisely known how KaZaA elects super-peers, since it is a proprietary application and its specification and source code are not publicly available. Based on reverse engineering, it is believed that peers in KaZaA decide to become super-peers using local knowledge about their own characteristics, such as bandwidth, processing power, unrestricted access to the Internet and availability [102]. Furthermore, becoming a super-peer is voluntary, as each peer has the option to suppress the super-peer functionality.

2.2.5 Gnutella 0.6

As KaZaA became popular and the super-peer approach proved valid, super-peers were also introduced in Gnutella version 0.6 [177]. Like KaZaA, Gnutella 0.6 used super-peers (called *ultrapeers*) for indexing files stored by clients (called *leaves*) and for handling search. The introduction of super-peers reduced the load on the lowest-performance peers and improves the scalability of the Gnutella network.

Gnutella 0.6 divides leaves into two categories: ultrapeer-capable and ultrapeer-incapable. The distinction is based on minimum performance requirements. The Gnutella 0.6 protocol [177] specifies that a peer is capable of becoming an ultrapeer if it has a non-firewalled connection to the Internet (allowing incoming TCP connections and UDP packets), minimum of 20 KB/s downstream bandwidth and 10 KB/s upstream bandwidth, an operating system that can handle large numbers of simultaneously open sockets, such as Linux, Windows 2000/NT/XP and Mac OS/X, and a sufficiently high uptime.

Every ultrapeer can accept up to 32 connections from leaves and up to 30 connections from other ultrapeers. In order to become an ultrapeer, a leaf peer must meet the ultrapeer requirements and must connect to an ultrapeer with at least 27 clients (90% of maximum 32). If an ultrapeer-capable leaf connects to an ultrapeer that has fewer than 27 clients, it connects as a leaf; otherwise, it connects an ultrapeer [103].

This way, a new ultrapeer is created when an existing ultrapeer has utilised 90% of its capacity. Generally, ultrapeers are created when new peers are added to the system, and ultrapeers are removed when peers leave the system. Moreover, every Gnutella user can force its peer to act as an ultrapeer or a leaf, disabling the election algorithm.

An interesting feature of the Gnutella 0.6 protocol is that it is backward compatible with earlier versions. For this reason, it is possible for ultrapeers and leaves to coexist in one Gnutella overlay with peers that do not distinguish between super-peers and clients and connect to all of them in the same manner.

2.2.6 eDonkey

eDonkey is another P2P file-sharing application that introduced super-peers, called eDonkey servers, for handling search [188, 75]. eDonkey servers do not share any files and do not initiate downloads, but index files stored on their clients and enable search. Every peer in the eDonkey network is eligible to setup a server, and the decision is made manually by each eDonkey user.

2.2.7 Yang and Garcia-Molina

Yang and Garcia-Molina [200] analyse the performance of super-peer based file-sharing networks and give practical guidelines for designing such networks. They investigate the relationships between the number of super-peers in the network, load on super-peers, and search performance, in order to find the formula for an optimal system configuration. Their model is specific to file-sharing systems and is not easily generalised to other application areas. Moreover, they do not address the problem of super-peer election and super-peer topology maintenance, as they only analyse static properties of super-peer networks.

2.2.8 Skype

Skype [15, 67] is a P2P telephony system that enables voice communication between users using computers connected through a wide-area network. It uses super-peers (called *supernodes*) for relaying traffic between firewalled peers.

Peers are classified into two categories: *firewalled* peers, located behind a firewall or Network Address Translator (NAT), which usually have a private IP address and cannot receive incoming TCP connections and UDP packets, and *non-firewalled* peers, which have full access to the network and can accept all connections. Super-peers are elected amongst non-firewalled peers only.

Depending on the type of the calling peer and the callee, four scenarios are possible. If both the caller and the callee are non-firewalled, they can communicate directly. Similarly, if the caller is firewalled but the callee is non-firewalled, the caller can establish a direct connection with the callee.

Every firewalled peer, in order to receive phone calls, maintains a permanent connection with a super-peer. If a non-firewalled peer attempts to call a firewalled peer, it first contacts the super-peer associated with the callee, the super-peer then notifies the callee, and the callee initiates a connection to the caller. When the connection has set up, the caller and the callee can communicate directly.

In the last possible scenario, where both the caller and the callee are firewalled, the connection procedure consists of two steps. First, the caller connects to the super-peer of the callee, and synchronised by the super-peer, the caller and the callee attempt to establish direct connectivity using STUN [161, 160], a NAT traversal protocol. If this step fails, peers fall back to the TURN protocol [159], where all messages between the caller and the callee are relayed by the super-peer [67].

Although the Skype protocol specification and source-code are not publicly available, it is believed that peers are promoted to super-peers in Skype if they are non-firewalled and have a high amount of bandwidth [67].

2.2.9 JXTA

Juxtapose (JXTA) [65] is a network programming and computing platform, created by Sun Microsystems, specifically designed to be the foundation for creating P2P systems. JXTA standardises a common set of protocols that allow groups of hosts to establish P2P overlay networks. JXTA can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, and many other transport-layer protocols.

JXTA uses two types of super-peers, called *rendezvous* peers and *relay* peers [186].

Rendezvous peers are peers that enable search. They cache so-called advertisement indices, i.e., pointers to client peers that advertise particular resources, and act as directory services. Rendezvous peers also connect to each other and form a semi-structured super-peer network, where each rendezvous peer maintains a loosely-consistent list of all other rendezvous peers in the system. Rendezvous peers forward search queries between each other using a limited-range rendezvous walker algorithm [185].

Relays are used for bridging peers that do not have direct physical connectivity, for example due to firewalls or NAT. Like rendezvous peers, relay peers connect to each other and maintain loosely-consistent lists of all relay peers available in the system. A routing algorithm is used to establish a connection between any two peers in the system through a series of relays.

Furthermore, JXTA uses two sets of peers that act as permanent rendezvous and permanent relays, called seeding rendezvous and seeding relays. The seeding peers are used for bootstrapping peers that join the overlay.

The super-peer election algorithm in JXTA is not fully specified. According to [186], "any peer can become a rendezvous peer assuming it has the right credentials". Similarly, "any peer may become a relay peer assuming it has the right level of credentials and capacity (bandwidth, low churn rate and direct connectivity)". It is up to the higher-level application to decide on the minimum capacity and credentials required for rendezvous and relay peers. The JXTA white paper specifies also that "if none of the seeding rendezvous is reachable after a tunable period (5 minutes), the edge [i.e., client] peer will try to become a rendezvous (if it has the right credential)" [186]. Seeding rendezvous and seeding relays must be hard-coded in the application.

2.2.10 LST

Kleis et al. [90] propose the use of Lightweight Super-Peer Topologies (LST) for routing in P2P networks. Their approach is based on Yao-Graphs and the Highways scheme, which allow the construction of Euclidean minimum spanning tree. Such a tree structure can be used for efficient multicasting. LST relies on super-peers, however, Kleis et al. [90] do not specify any super-peer election mechanism. They only mention that a super-peer "should have *enough* resources to serve other peers" and "should be reliable in the sense that it is not joining and leaving the P2P network frequently". Furthermore, "trust and security incentive schemes could be layered in this election process."

2.2.11 SBARC and Zhu et al.

Another example of a P2P system that does not specify any super-peer election algorithm is the Supernode Based Peer-to-Peer File Sharing System (SBARC) [197]. SBARC improves the routing performance in Pastry DHT [162] by routing messages through high-capability super-peers. Xu and Hu describe a routing algorithm and a data caching scheme that take advantage of super-peers, however, they do not elaborate on the super-peer election mechanism. The following paragraph describes their super-peer election criteria.

> The basic requirement is a high bandwidth connection. [...] The second criteria is it [i.e., super-peer] must have enough computation power because supernodes need to deal with most system workloads. [...] Third, supernodes can not join/leave system frequently, otherwise its efficiency will be greatly reduced. Also it is helpful if supernodes have large amount of storage space. [197]

It is not specified what amount of available bandwidth or computational power or storage space is required for a super-peers, and no algorithm is given that could calculate such minimum super-peer requirements. Furthermore, it is not explained how to estimate peer stability in order to elect super-peers that do not join or leave the system frequently.

A very similar approach is described by Zhu et al. in [209], where the routing performance in a DHT is improved through the introduction of super-peers. However, as in [197], Zhu et al. [209] focus on routing strategies and do not describe any super-peer election algorithm. They only specify the following criteria for super-peers: “significant processing power, high bandwidth, high availability, and large amounts of memory and storage space”. The details of the super-peer election are not discussed “due to space constraints.”

2.2.12 SUPS

Scalable Unstructured P2P System (SUPS) [145] is another system that relies on super-peers but does not contain any super-peer election mechanism. The main goal of SUPS is to produce a low-diameter and balanced super-peer topology. A neighbour selection algorithm is shown, inspired by the theory of random graphs, that connects super-peers in such as way that the topology diameter is $\Theta(ln\,N/ln\,ln\,N)$ and the average super-peer degree is minimised, given a system with N super-peers.

However, [145] does not address the problem of super-peer election. It only mentions that “super-peers are selected from normal peers that have high bandwidth, high computing power, a long resistance time, and a low likelihood of failure [...] the detailed choice of super-peer remains as a separate research topic not addressed in this paper.”

2.2.13 Schelling Algorithm

Singh and Haahr [176] propose a neighbour selection algorithm, inspired by the Schelling’s model, that constructs and maintains *hub topologies*. A hub topology can be defined as P2P topology, where selected peers, called hubs, are highly connected with each other, while ordinary peers are connected to both hubs and other ordinary peers. Hub topologies are considered more robust than classic super-peer topologies, since in the former, a super-peer failure causes an isolation of its clients, while in the latter, ordinary peers are connected to multiple super-peers and can handle hub failures.

The original Schelling’s model is a sociological model proposed to explain the existence of segregated neighbourhoods in the United States. In this model, each agent defines its satisfaction condition, and changes its neighbourhood whenever the satisfaction condition is not met by performing an adaptation procedure.

The customised Schelling algorithm that creates hub topologies is shown in Figure 2.2. A hub is satisfied if it is connected to at least one, but not more than H_{max}, other hubs, where H_{max} is a system constant (lines 1-3). If the number of hub neighbours, h, is above H_{max}, the hub drops one hub connection (lines 5-9). If h is zero, the hub connects to another hub discovered by performing a Depth First Search (lines 10-13). An ordinary peer is satisfied if it is connected to at least one hub (lines 14-17). If this condition is not met, the peer discovers and connects to a hub (lines 23-24). Furthermore, if its number of neighbours is above N_{max}, where N_{max} is a system parameter, it removes a random neighbour (lines 18-22).

```
1: **satisfied()**
2: $h \leftarrow$ number of hub neighbours
3: **return** $(0 < h < H_{max})$

4: **adaptation()**
5: $h \leftarrow$ number of hub neighbours
6: **if** $h > H_{max}$ **then**
7:     $q \leftarrow$ one of hub neighbours
8:     disconnect $q$
9: **end if**
10: **if** $h = 0$ **then**
11:     $s \leftarrow$ search for hubs
12:     connect $q$
13: **end if**
```

(a) Hub

```
14: **satisfied()**
15: $h \leftarrow$ number of hub neighbours
16: **return** $(h > 0)$

17: **adaptation()**
18: $n \leftarrow$ number of neighbours
19: **if** $n > N_{max}$ **then**
20:     $q \leftarrow$ one of neighbours
21:     disconnect $q$
22: **end if**
23: $s \leftarrow$ search for hubs
24: connect $q$
```

(b) Ordinary Peer

Figure 2.2: Schelling algorithm for the generation of hub topologies.

The algorithm does not determine which peers act as hubs. According to [176], hubs are "high-availability and high-capacity peers". They can be be exploited in a similar manner as super-peers in traditional super-peer systems, for example to perform resource-intensive tasks such as "maintaining a directory of resources in the network and processing search queries". The main advantage of the Schelling algorithm is its simplicity and robustness. However, it does not address the super-peer election problem.

2.2.14 Discussion

All systems described in this section employ very simple approaches to super-peer election and management, which are not likely to produce optimal, or close to optimal, system configurations. Centralised approaches introduce reliability and scalability concerns, and are in obvious contradiction with basic principles of P2P systems. Manual super-peer selection is difficult due to the large scale, dynamism, and complexity of P2P systems. A global system administrator is not likely to possess sufficient knowledge about the system to select an optimal super-peer set. A local user at each peer is even less likely to obtain such a knowledge about the system.

Most of the reviewed systems specify criteria for super-peer election. In most cases, these criteria are described as a high amount of available bandwidth, storage space, processing power, and a long session time. However, many systems do not provide any mechanism for the estimation of peer properties in order to apply these criteria. For example, it is not obvious how a peer can estimate its remaining session time. More importantly, given a criteria such as a high amount of property x, many systems do not specify precisely what *high* is.

In many reviewed systems, the super-peer criteria is defined as static threshold, e.g., $x > 5$, such that all peers with property x above the threshold automatically become super-peers and all the remaining peers become clients. This simple approach has the advantage that every peer can make the decision about becoming a super-peer locally, without any communication with other peers. In some cases, the threshold may be directly defined by the requirements of a higher-level application.

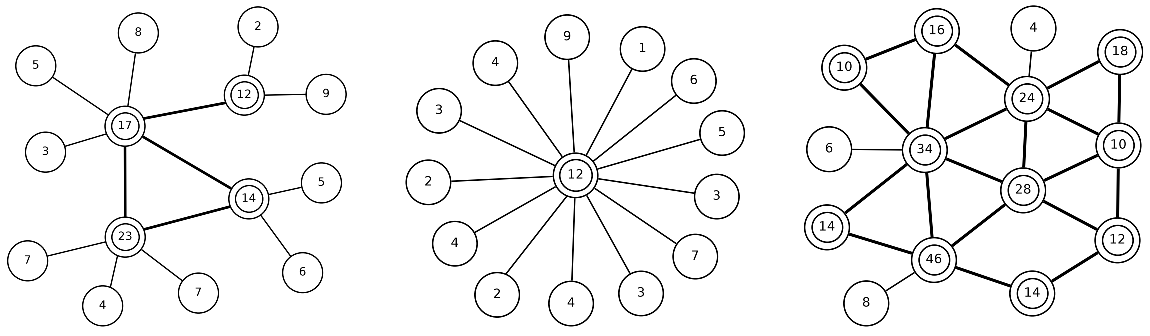
However, this approach has a serious drawback. A static threshold does not allow the system to control the number of super-peers in dynamic populations of peers. If peer properties change, the super-peer sets changes accordingly. This can lead to extreme cases, where no super-peers are elected, if all peers fall below the threshold, or where every peer is a super-peer, if all peers are above the threshold.

In many P2P applications, the number of super-peers is critical for the system performance. For example, in file-sharing applications, the number of super-peers must be significantly smaller than the total number of peers in order to reduce the amount of search traffic to an acceptable level. At the same time, the number of super-peers must be large enough to be able to handle the load associated with serving clients. Often, the optimum number of super-peers in the system depends on the current load, e.g., the number of user requests. A static threshold does not allow the system to adapt the size of the super-peer set to the current demand.

Setting a threshold that limits the number of super-peers to a desired level requires a global knowledge of peer characteristics. In many systems, it is assumed that peer properties follow a certain distribution. Such a distribution can be estimated using domain-specific knowledge, or measured experimentally in deployed systems.

However, there are a number of reasons why the distribution of peer characteristics may change. A natural change may occur over time, as new technologies are developed and the Internet evolves. According to the Moore's Law [129], the capabilities of computers, measured as the number of transistors placed on an integrated circuit, is increasing exponentially, doubling approximately every two years. The global network conditions and the general behaviour of users are another two important and unpredictable factors. Peer properties may depend on the time of the day, day of the week, and external events which may generate flash crowds. Peer properties may also change when the system is deployed in a different environment, for example in a different country, or exposed to a different group of users.

Figure 2.3 shows a sample super-peer election scenario with a static threshold. Each peer is assigned a capacity value, and the super-peer election threshold is 10. In Figure 2.3(a), four super-peers are elected, and the ratio of clients to super-peers is balanced. In Figure 2.3(b), the capacity of all peers is reduced by a half. As the threshold remains static, only one super-peer is elected in the system. Such a single super-peer is likely to become overloaded. In Figure 2.3(c), the capacity of all peers is twice increased. Nearly all peers in the system become super-peers and the system is again



(a) Initial configuration (b) Peer capacity is halved (c) Peer capacity is doubled

Figure 2.3: Threshold-based super-peer election example.

likely to becomes inefficient.

To conclude, the systems reviewed in this section clearly show that there are many applications scenarios where the performance and scalability of a P2P network can be significantly improved by introducing super-peers. Moreover, as the reviewed systems offer relatively simple super-peer election mechanisms, there is a general need for more sophisticated techniques that would allow peers to better control the number of super-peers in the system and to adapt the super-peer set to the changing system conditions. Such approaches to super-peer election are covered in the next three sections.

2.3 Super-Peer Election in Peer Groups

The systems described in this section rely on a common mechanism of peer grouping. In these systems, the population of peers is divided into disjoint groups, based on peer properties such as physical location or data semantics, and a super-peer, or multiple super-peers, are elected within each group independently.

The main advantage of this approach is that the general problem of super-peer election in the system is decomposed into a number of local election subproblems within groups, which are easier to solve since groups are much smaller than the total system size. In the most straight-forward case, the characteristics of all peers in a group are directly compared with the existing super-peer. Furthermore, systems that belong to this class usually impose restrictions on the super-peer topology, such as a certain maximum distance between a super-peer and a client, according to some metric, in order to satisfy application-specific requirements.

In most approaches, super-peers are created on demand, as peers join the system. The general structure of the join procedure consists of the following steps, as shown in Figure 2.4. First, the joining peer, p, determines the group, g, that it belong to (line 2 in Figure 2.4), and contacts the super-peer, s, responsible for group g (line 3). These steps may involve interactions between p and other peers in

```
1:  **Join():**
2:  $g \leftarrow$ group of peer $p$
3:  $s \leftarrow$ super-peer in group $g$
4:  if $s$ is nil then
5:      become super-peer
6:  else
7:      if $s$ higher capability than $p$ then
8:          become client of $s$
9:      else
10:         become super-peer
11:         transfer all clients from $s$ to $p$
12:         request $s$ to become client of $p$
13:     end if
14: end if
```

Figure 2.4: Join procedure at peer p in a group-based system.

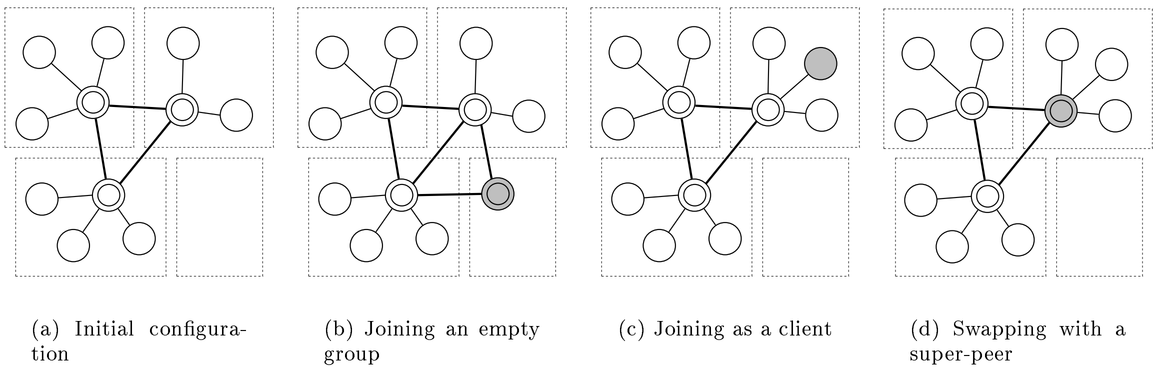

(a) Initial configuration
(b) Joining an empty group
(c) Joining as a client
(d) Swapping with a super-peer

Figure 2.5: Sample join scenarios in a group-based system.

the system, and are system-specific. It is assumed that every peer must know at least one other peer that already participates in the system in order to join.

If no super-peer exists in group g, peer p becomes a new super-peer in group g (lines 4-5). If a super-peer s is found, peer p compares its characteristics with s in order to decide which of the two peers is more suitable to serve as a super-peer for the group (line 7). If peer p has higher capabilities than s, it joins the group as a client of s (line 8). Otherwise, it becomes a new super-peer and swaps its roles with s. In the latter case, all existing clients are transferred from s to p and s besoms a client of p (lines 10-12).

The join procedure is graphically shown in Figure 2.5. An initial configuration consists of four groups, three super-peers and eight clients (a). If a new peer enters an empty group, it becomes a new super-peer in this group (b). If a peer joins a non-empty group, it connects to the existing super-peer in this group (c), compares its capabilities with the super-peer using some application-specific metric, and potentially swaps its role with the super-peer (d).

Apart from peer arrivals, the super-peer election algorithm is run whenever an existing super-peer

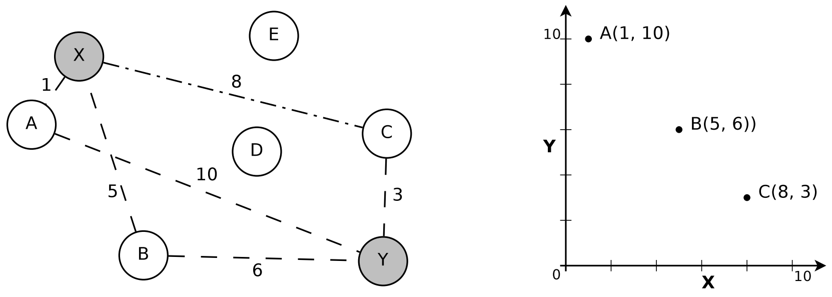


Figure 2.6: Sample coordinate space based on two landmark nodes.

becomes unavailable. Often, the super-peer maintains a list of all peers in the group and periodically broadcast this list to all its clients. The list is ordered based on the characteristics of individual peers, from the highest-capability peer to the lowest-capability peer. In case of a super-peer failure, an election is performed and the first peer on the list becomes a new super-peer. A number of variations of the super-peer election algorithm exist, for example where multiple super-peers are elected within each group, and where groups are dynamically split and merged with each other during the course of the system's operation. If peer characteristics are dynamic, the super-peer may occasionally swap with one of its clients.

The systems reviewed in this section are divided into three subcategories, where peers are organised into groups based on physical location, semantic description, and administration domain (in Grids).

2.3.1 Location-Based Systems

In a number of systems, peers are organised into groups based on their physical location, for example using information about their country, ZIP code, or ISP. Many systems also introduce peer distance metrics, for example defined as the communication latency or number of IP hops on the route between two peers. In these systems, the goal is usually to assign clients to super-peers that are close to them according to the distance metric.

Many systems also use Internet coordinate systems, such as GNP [136] and Vivaldi [45], where every peer is positioned in a k-dimensional virtual coordinate space. The k coordinates of each peer are calculated by measuring the distance of each peer to k well-known *landmark* hosts. If the landmark nodes are evenly distributed in the system, the Euclidean distance between peers in the virtual coordinate space approximates the distance between these peers in the physical network.

Figure 2.6 shows an example of a two-dimensional coordinate space generated using two landmark nodes, X and Y. Peer A measures its distance to X and Y and obtains $(1, 10)$ as its coordinates . Similarly, peer B determines its coordinates as $(5, 6)$. The distance between A and B can be estimated as $\sqrt{(1-5)^2 + (10-6)^2} = 5.6$.

A k-dimensional virtual space can be mapped onto a lower-dimensional space (one-dimensional space in particular) using a Space Filling Curve (SFC), such as the Hilbert space-filling curve and the z-order curve. These curves have the locality-preserving property. Points that are close in the original k-dimensional space are mapped onto points that are also close in the target space.

2.3.1.1 Crown

Crown [192] is a distributed resource management protocol, similar to a distributed hash table. It allows a P2P system to distribute resources, such as files, programs, and services, uniformly between peers, and it enables an efficient resource lookup protocol. Each peer in Crown is assigned two identifiers: a peer identifier, which is unique, and a group identifier, which is potentially shared with other peers. A peer identifier is defined as the SHA-1 hash of the peer's IP address, and a group identifier is defined as the SHA-1 hash of the first $m = 24$ bits of a peer's IP address. It is assumed that peers which share a common IP address prefix are likely to be located in the same local network or autonomous system, and hence, the connections between such peers are likely to have low latency and high throughput.

Peers that belong to each group elect a super-peer. The details of the super-peer election algorithm are not described in [192], but it can be expected that a similar algorithm to the one described in Figure 2.4 can be applied in Crown. The criteria for super-peer election in Crown are: high peer bandwidth, high availability (uptime), large computational power, and a low load. Additionally, users are allowed to manually select which peers become super-peers.

Super-peers connect to all clients in their respective groups and to other super-peers in the system, forming a ring topology that spans all peer groups. Super-peers run a decentralised lookup protocol, which allows them, and their clients, to locate resources available in the system.

2.3.1.2 PASS

A similar approach to super-peer election is proposed in the Peer-to-peer Asymmetric file Sharing System (PASS) [96], a P2P file-sharing application. Peers in PASS are divided into multiple areas based on their geographical location, using information such as ZIP codes or administrative network domain names. PASS does not impose any particular division scheme, but it assumes that the latency between peers that belong to the same area is low compared with the entire system.

Each group of peers independently elects its own super-peers. The super-peers maintain a distributed directory of system data and handle search. As in other file-sharing applications, search requests are propagated between super-peers only and are not forwarded to clients in order to reduce search overhead. Furthermore, one of the super-peers in each area is elected as a Representative SuperNode (RSN), which handles inter-area communication. It is assumed that due to the location-aware division of peers into groups, operations executed locally in a group, such as super-peer elections, directory updates and lookups, are performed at low cost.

As the performance of super-peers (particularly RSNs) is critical for the system's operation, the super-peers are selected from the most stable peers in the system. The first peer that enters an area, becomes a super-peer and RSN for this area. If an existing super-peer becomes overloaded, e.g., due to a high number of clients, the super-peer selects one of its clients, promotes it to a super-peer, and splits the remaining clients between itself and the new super-peer. Furthermore, each super-peer appoints a backup peer amongst its clients; when a super-peer leaves, it is replaced by its backup peer.

2.3.1.3 PoPCorn

A very different approach is followed in PoPCorn [105], which assumes an n-dimensional Euclidean space generated using an Internet coordinate system such as GNP and Vivaldi described above. PoPCorn elects k super-peers and distributes them evenly in the coordinate space. The distribution criteria, *dispersal*, is achieved by maximising the sum of inter-node distances between pairs of super-peers.

Super-peers are elected using k *tokens* exchanged between peers using a repulsion model. The initial token placement is random. Every peer that holds a token performs a scoped gossip with its neighbours in order to notify them about tokens in their vicinity. When a peer receives a gossip message, it updates its model of nearby tokens and calculates a combined repulsion force of these tokens. If the repulsion force exceeds a threshold, TR, the token is passed to another peer, according to the force direction. If a token stays on a peer for a certain number of time steps, T, this peer becomes a super-peer. Each peer calculates its threshold TR based on its capabilities. Peers that are better qualified to serve as super-peers have higher values of the threshold TR, and hence, are more likely to be elected super-peers.

The algorithm description in [105] is very brief and is missing many details. In particular, it it not entirely clear how peers select their neighbours and what topology they maintain, how they perform gossip, and how they discover peers that are closest to the virtual locations defined by the repulsion forces. No evaluation is shown in [105] and there is no evidence that the algorithm generates the desired system configurations. Furthermore, it is not obvious how to deal with peer failures and departures, and in particular, how to address the problem of lost tokens.

2.3.1.4 Wolf and Merz

Wolf and Merz [195] consider a similar problem of constructing a super-peer topology where the distance between super-peers and their clients, as well as between connected super-peers, is minimised. The motivation for this problem is to minimise the total communication cost in the system. Wolf and Merz state that the problem is NP-difficult, but show a heuristic based on evolutionary algorithms enhanced with local search that generates close-to-optimal super-peer topologies. However, the pro-

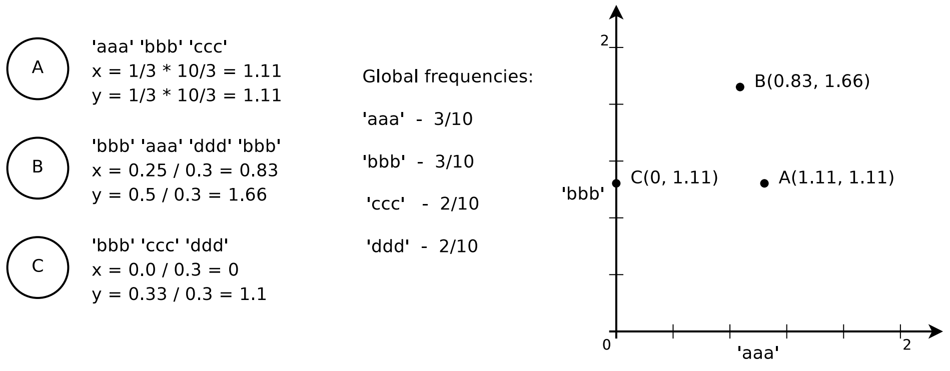


Figure 2.7: Sample coordinate space based on term frequencies.

posed heuristic algorithm requires a global view of the entire network and as such cannot be directly deployed in a P2P system.

2.3.2 Semantics-Based Systems

A number of P2P systems elect super-peers based on peer semantics, for example using tags, text descriptions, or XML descriptions, or ontology concepts associated with peers. A number of standards have been proposed for the semantic description of peers, which include the Resource Description Framework (RDF), Web Ontology Language (OWL), and RDF Schema. Many systems also introduce formal metrics that measure the semantic similarity between peers. Furthermore, virtual coordinate spaces can be generated based on peer semantic properties, as in location-based systems.

Tang et al. [183] describe an approach where peers are mapped onto a coordinate space using the Vector Space Model (VSM). In this model, every peer is associated with a set of text documents, and a fixed set of k terms is used to calculate peers' coordinates. The i'th coordinate of peer p is defined as $\frac{f_p(i)}{f(i)}$, where $f_p(i)$ is the frequency of the i'th term in p's documents, and $f(i)$ is the total frequency of the i'th term in all documents. The distance between peers p and q, assigned coordinates $X = (x_1, x_2, \ldots, x_k)$ and $Y = (y_1, y_2, \ldots, y_k)$ is given as $cos(X, Y)$.

Figure 2.7 shows a sample two-dimensional coordinate space based on terms *'aaa'* and *'bbb'*. Peer A is described by terms *'aaa', 'bbb'*, and *'ccc'*. Given the global frequencies of *'aaa'* as 0.3 and *'bbb'* as 0.3, both coordinates of peer A are equal to $1/3 \cdot 0.3 = 1.11$. Similarly, peer B, associated with terms *'aaa', 'bbb', 'ddd'*, and *'bbb'*, is assigned coordinates $1/4 \cdot 0.3 = 0.83$ and $1/2 \cdot 0.3 = 1.66$, and peer C, with keywords *'aaa', 'ccc'*, and *'bbb'* has coordinates $0 \cdot 0.3 = 0$ and $1/3 \cdot 0.3 = 1.1$.

VSM has the disadvantage that it does not recognise term synonyms and generates coordinate spaces with large dimensions. These problems are addressed in the Latent Semantic Indexing (LSI) algorithm [48], which projects high-dimension vectors generated by VSM onto low-dimension semantic subspaces, selecting the most relevant terms for each peer.

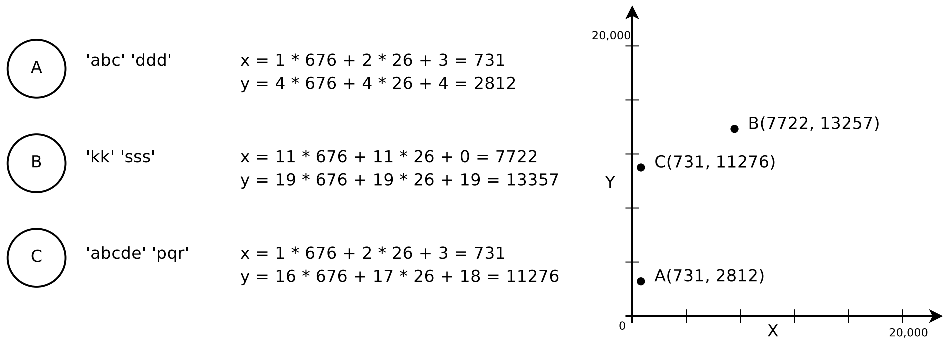

A
'abc' 'ddd'
x = 1 * 676 + 2 * 26 + 3 = 731
y = 4 * 676 + 4 * 26 + 4 = 2812
B
'kk' 'sss'
x = 11 * 676 + 11 * 26 + 0 = 7722
y = 19 * 676 + 19 * 26 + 19 = 13357
C
'abcde' 'pqr'
x = 1 * 676 + 2 * 26 + 3 = 731
y = 16 * 676 + 17 * 26 + 18 = 11276
20,000
B(7722, 13257)
C(731, 11276)
Y
A(731, 2812)
0
X
20,000

Figure 2.8: Sample coordinate space based on fixed-length terms.

Another approach is proposed by Smidth and Parashar [170], where every peer is described by a sequence of k keywords and positioned in a k-dimensional coordinate space. The i'th coordinate of a peer is defined by the i'th keyword of this peer. Each keyword is treated as a d-digit base-b number. Keywords that are longer than d digits are truncated by removing their last letters. Keywords shorter than d letters are padded with a special zero character.

For example, if keywords were constructed over the plain Latin alphabet with 26 letters, each keyword would be a base-26 number, with 'a' representing 1, 'b' representing 2, and so on. Figure 2.8 shows a sample two-dimensional coordinate space ($k = 2$) with three-letter keywords ($d = 3$) over a 26-character alphabet ($b = 26$). Peer A, associated with keywords '*abc*' and '*ddd*' is assigned coordinates $1 \cdot 26^2 + 2 \cdot 26^1 + 3 \cdot 26^0 = 731$ and $4 \cdot 26^2 + 4 \cdot 26^1 + 4 \cdot 26^0 = 2812$. Similarly, peer B with keywords '*kk*' and '*sss*' has coordinates $11 \cdot 26^2 + 11 \cdot 26^1 + 0 \cdot 26^0 = 7722$ and $19 \cdot 26^2 + 19 \cdot 26^1 + 10 \cdot 26^0 = 13357$, and peer C with keywords '*abcde*' and '*pqr*' has coordinates $1 \cdot 26^2 + 2 \cdot 26^1 + 3 \cdot 26^0 = 731$ and $16 \cdot 26^2 + 17 \cdot 26^1 + 18 \cdot 26^0 = 11276$.

A number of other peer distance metrics based on peer semantics are described in [147].

2.3.2.1 Edutella

Edutella [132] is a file-sharing P2P network where every shared file and every peer is described using RDF schemas and RDF metadata. As many other file-sharing networks, Edutella uses super-peers for handling search in order to reduce the overall search cost and overhead [134, 133]. Super-peers are arranged in a hypercube topology, maintained using the HyperCuP protocol [169], which enables efficient routing of search queries, and is also robust to multiple peer failures. In a hypercube with N peers, the path between any two peers has at most $log_2 N$ overlay hops, and the minimum number of peers that need to be removed to partition the network is $log_2 N$, while each peer maintains $log_2 N$ neighbours. A hypercube overlay also allows efficient broadcast and multicast.

In Edutella, every peer is described using a combination of *structuring concepts,* defined through a set of global ontologies, known to all peers in the system [169, 108]. These structuring concepts determine the coordinates of each peer in the hypercube. An ordinary peer connects to the semantically closest super-peer. A super-peer joins the hypercube overlay using the HyperCuP protocol. This way, peers that have similar semantic characteristics are located close to each in the system topology.

According to [134], "super-peers provide the necessary bandwidth and processing power to enable efficient and reliable query processing and answering". However, no algorithm is described that would allow the selection of such high-performance peers. In all algorithms described in [169, 134, 133, 108], it is assumes that every peer joins the system either as a super-peer or an ordinary peer and it is not explained how a peer decides on its role.

2.3.2.2 ROSA-P2P

The Repository of Objects with Semantic Access (ROSA) [26] is an e-learning system, whose purpose is to support teachers in the preparation of didactic materials. ROSA-P2P [26] is a P2P network that provides a distributed storage and search facility for ROSA.

As in Edutella, ROSA-P2P organises peers into groups based on their semantic characteristics. Each group consists of peers that have similar subjects and localisation. Furthermore, each group elects a super-peer, which is responsible for storing data and handling search. The super-peer election criteria are based on peer characteristics such as stability, bandwidth, processing power, available memory and storage capacity. In order to act as a super-peer, a peer needs to satisfy certain performance requirements. Additionally, peers that belong academic institutions automatically become super-peers.

Each peer can declare one of three preferences: to refuse being a super-peer, to accept becoming a temporary super-peer, or to accept being a permanent super-peer. In the first case, the peer never becomes a super-peer. In the second case, the peer becomes a super-peer only if there is no other peer in its group that agrees to be a permanent super-peer. In the third case, the peer is elected a super-peer if it is the highest-capability peer in its group.

When a peer joins the ROSA-P2P network, it first determines the group it belongs to and contacts the super-peer in this group, as in the general scheme shown in Figure 2.4. The list of available groups and corresponding super-peers is obtained from a centralised ROSA portal. The further steps depend on the preferences of the current super-peer and the joining peer. If no super-peer is found in the group, the joining peer is requested to become a super-peer. If it rejects the request, it is either assigned to a different group of is not allowed to join the system. If a temporary super-peer is found in the group, and the joining peer agrees to become a permanent super-peer, the existing super-peer and the joining peer swap their roles. In all other cases, the joining peer becomes a client of the existing super-peer in the group.

Each super-peer periodically compares the characteristics of its clients and selects the highest-capability peer in its group that agrees to serve as a super-peer. If the selected peer has better characteristics than the existing super-peer, the roles of the two peers are swapped. Moreover, each super-peer defines a limit on the maximum number of clients it can simultaneously support. When the number of peers in a group approaches the limit, the super-peer assigns a new super-peer from its clients and splits the group.

Even though ROSA-P2P has been thought as a decentralised P2P system, it is partly centralised, as it relies on a centralised ROSA portal, which maintains the information about the current structure of groups in the system. The centralised portal allows all peers in the system to achieve a synchronised view on the peer groups.

2.3.2.3 Qiao et al.

Qiao et al. [147, 146] propose a P2P system where peers are arranged using a globally-known taxonomy of concepts. Such a taxonomy is a tree, or a hierarchy, where vertices represent concepts, and edges represent relationships between concepts. Every peer is characterised by a number of concepts, but the taxonomy tree is maintained by super-peers only. Each super-peer is responsible for one or multiple subtrees of the taxonomy tree, and maintains an index of all peers that are characterised by the concepts in these subtrees. A super-peer together with associated clients form a semantic cluster.

Super-peers are created on demand. Each concept is associated with a load, defined for example as the frequency of queries related to the concept, or the number of clients or data items associated with the concept. Moreover, every super-peer has a maximum capacity that defines the maximum load it can handle. When the load in a cluster exceeds the capacity of the super-peer, the super-peer selects one of its clients as a new super-peer and splits the cluster.

There are two algorithms for splitting clusters. In the Semantic First Clustering Algorithm (SFCA), a cluster always contains a single subtree of the global hierarchy. When a cluster is divided, the super-peer selects a subtree with load approximately equal to half of the current cluster's load. The resultant clusters always have a parent-child relationship.

The Load Balance First Clustering Algorithm (LBFCA) allows more flexible cluster splitting in order to balance the load between clusters more evenly. A cluster is divided into two subclusters according to the following three rules: (i) if the cluster consists of a single semantic tree, the tree is split into one or multiple subtrees and these subtrees are divided between subclusters; (ii) if the cluster consists of multiple subtrees, these trees are divided between subclusters; (iii) if the cluster consists of a single concept, the load associated with this concept is shared between subclusters.

Figure 2.9 shows a sample hierarchy consisting of concepts A, B, ... K. Each number represents the load associated with a concept. The maximum super-peer load is 20. Initially, the entire hierarchy belongs to one cluster X. When the load associated with every concept is doubled, the cluster is split into clusters X and Y using rule (i) in LBFCA. Next, the load in cluster Y is doubled again, and cluster Y is split into Y and Z using rule (ii). In the last scenario, the load in cluster Y is doubled

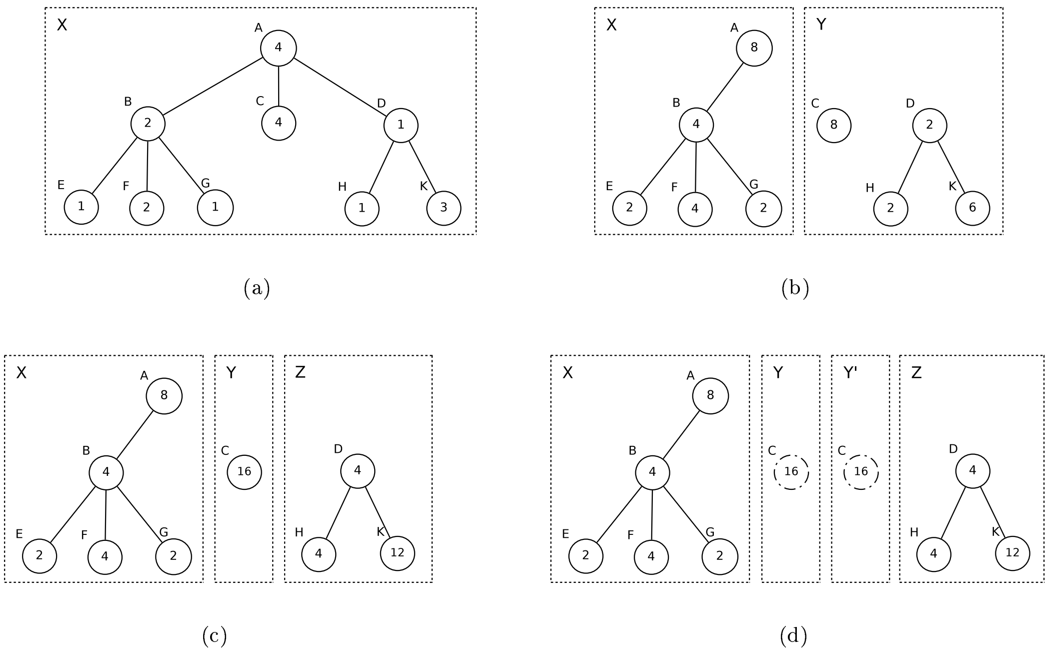


Figure 2.9: Sample division of a semantic hierarchy using LBFCA.

once again, and cluster Y is divided into Y and Y' using rule (iii), so that the single concept A is shared by both clusters Y and Y'.

Although SFCA and LBFCA offer a flexible and adaptive division of the semantic space between super-peers, they raise a number of questions. Both algorithms do not specify how clusters are merged, for example when the load in the system decreases. Furthermore, they require a static and globally known hierarchy of concepts. It is not obvious how such a hierarchy is created and what authority maintains it. One approach would be to adapt a centralised server, as in ROSA-P2P, but this would conflict with the system's decentralisation.

Finally, it can be questioned if a hierarchy is a generally a suitable structure for P2P systems. If a super-peer hierarchy is used for routing or searching, as described in [147], the top-level super-peers may easily become overloaded. Moreover, the failure of the top-level super-peers, potentially due to a malicious attach, may disable the entire P2P system.

2.3.3 Grid Systems

A large-scale Grid can be viewed as a network interconnecting small-scale, proprietary Grids, where each of such small-scale Grids is a network composed of hosts located within one administrative domain, called a Virtual Organisation (VO) [116, 175]. Grid systems and P2P systems, although considered distinct areas, share a number of common characteristics. In particular, both Grid and P2P systems consist of large numbers of nodes physically distributed between different geographical locations, which cooperate with each other in order to provide system-level services.

The main differences between Grids and P2P systems are their scale, dynamism, and the degree to which these systems are controlled. In the currently largest deployed Grids, the numbers of participating nodes are on the order of thousands [117], while in the most popular P2P systems, such as Skype and KaZaA, the numbers of simultaneous users have already exceeded millions [35, 102]. Furthermore, Grid nodes are more stable, as they are usually run by large enterprises and public institutions and are often hosted on dedicated servers. In P2P networks, nodes can freely join and leave the system. Grid systems are also more tightly controlled by system administrators and are more secure than P2P systems.

2.3.3.1 Mastroianni et al.

Mastroianni et al. [116, 117] propose the adoption of super-peers in large-scale Grids in order to improve the efficiency of resource discovery and membership management. In their approach, super-peers maintain metadata about their clients and run a search protocol in a similar manner as in P2P systems. The authors believe that super-peers can be selected manually in Grids, unlike in P2P systems, due to the fact that large-scale Grids are composed of smaller, locally-managed Virtual Organisations. In each VO, local administrators can select a set of the most "powerful" super-peers using local knowledge available within the organisation.

2.3.3.2 GLARE

A more self-managing approach to super-peer election is proposed in the Grid Activity Registration, Deployment and Provisioning Framework (GLARE) [175]. Here, the Grid is divided into multiple peer groups, as in [117], and each group elects a super-peer. Super-peers are used for relaying search queries and caching search results. However, unlike in [117], peer groups are not defined by VOs, but are rather created using the Globus Toolkit 4 (GT4) built-in hierarchical aggregation and indexing mechanism.

One member of each peer group is selected as an election coordinator using the GT4. In order to initiate an election, the coordinator notifies all peers in the group, and peers reply back to the coordinator with their static characteristics, such as processor speed, amount of memory, uptime, and site name. The coordinator ranks the peers and selects the highest-capability peer as a super-peer. In case the group consists of a large number of peers, the coordinator may decide to split the group by appointing multiple super-peers and splitting the group members equally between the super-peers. If a super-peer fails, the first peer that discovers the failure selects the highest rank peer as a new coordinator, who initiates a new election. If the majority of peers in the group confirm that the current super-peer is not available, the highest rank peer becomes a new super-peer.

2.3.4 Discussion

One of the main advantages of dividing a P2P system into peer groups is that many system-level problems can be decomposed into simpler to solve group-level subproblems. In particular, the general problem of super-peer election in the system can be decomposed into simpler subproblems of super-peer election in each group. Additionally, if super-peers are elected locally in each peer group, they are usually located close to their clients in terms of physical distance, communication latency, bandwidth, semantic similarity, or other metrics, depending on the mechanism used to construct peer groups. Such a proximity-aware organisation of super-peers and clients in the overlay topology is often required by higher-level applications.

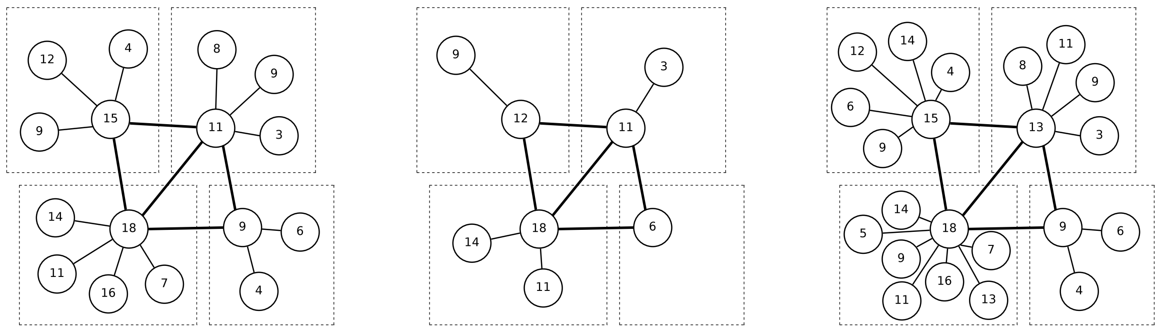
However, the use of peer groups introduces the problem of group creation and maintenance in a decentralised P2P system. Every peer needs to know the group it belongs to.

In the simplest case, a peer determines its group based on its static properties, such as IP address, ZIP code, ISP, administrative domain, or position in some global semantic ontology. This has the advantage that every peer can independently decide on its group, without negotiating with other peers. Moreover, in many systems, every peer group elects exactly one super-peer. This simplifies greatly the super-peer election, as it can be performed using a single coordinator in each group, which collects complete information about every peer in the group and selects a super-peer.

However, such simple approaches have a number of drawbacks. First of all, they do not allow the system to actively control and adapt the number of super-peers to changing system conditions. The number of super-peers is determined by the current number of groups and cannot be tuned at runtime. Furthermore, in many systems, the maximum number of groups is fixed and is determined by the number of possible values of peer properties, such as IP address prefixes, ISPs, ZIP codes, etc. If the total number of peers in these systems is low, groups have relatively few peers, and the overall ratio of clients to super-peers is low. Contrarily, if the number of peers is large, groups consist of large numbers of peers, and the ratio of clients to super-peers is high. For a certain system size, super-peers become overloaded. Inherently, such a system does not scale.

Figure 2.10 shows a sample system with four peer group. The capacity of each peer is marked by a number. In figure 2.10(a), the number of super-peers and clients is balanced. In figure 2.10(b), the overall number of peers is reduced, resulting in a very low client to super-peer ratio. In figure 2.10(c), the number of peers is increased, resulting in a high client to super-peer ratio and a high load on super-peers.

If the distribution of peers between groups is non-uniform, some groups may become highly populated by peers, while others may contain relatively few peers. For example, in systems where peers are assigned to groups based on their physical locations, it can be expected that groups corresponding to high-density, developed areas, such as cities, may contain more peers than other groups. In such



(a) Balanced ratio of clients to super-peers.

(b) Low clients to super-peers ratio.

(c) High clients to super-peers ratio.

Figure 2.10: Sample systems with four peer groups.

cases, the load between super-peers is imbalanced. Moreover, if the distribution of peer capabilities between groups is also skewed, some groups may contain significantly more high-capability peers than other groups. If the goal of the system is to elect the globally highest-capability super-peers, such a skewed distribution may lead to an suboptimum super-peer election. An example of such a scenario is shown in Figure 2.10(a). The highest capability values are 18, 16, 15, and 14. However, the elected super-peers have lower capabilities, i.e., 18, 15, 11, and 9. Peers with capabilities 14 and 16 are not elected super-peers because they are clustered in one group with peer 18.

These shortcomings can be addressed in two ways. First, multiple super-peers can be elected in each group. However, this requires additional mechanisms for the synchronisation and coordination of multiple super-peers elected in one group, and dividing clients between super-peers in a group. Ultimately, as the system size grows to a large number of peers, the problem of multiple super-peer election in a group becomes equivalent to the general problem of super-peer election in a P2P system.

The second approach is to adjust groups in the system dynamically, for example by splitting and merging them. However, this requires dedicated mechanisms for the management of dynamic groups in the system. Every peer needs to be able to determine the group it belongs to, and super-peers need to achieve an agreement between each other on the structure of peer groups. As this is non-trivial in a decentralised system, some systems rely on centralised servers that manage information about peer groups. Furthermore, merging and splitting groups requires an additional algorithm that decides when and which groups to merge and split. In the extreme case, where groups are fully flexible and can be arbitrarily changed, the problem of group construction is equivalent to the general super-peer election problem.

2.4 Super-Peers in Distributed Hash Tables

This section describes the class of P2P systems known as Distributed Hash Tables (DHT). A DHT is a P2P system that distributes *values,* such as objects, chunks of data, and user requests, between peers in the system. Each value is associated with a *key,* and the system provides an efficient and deterministic mapping from keys to peers. DHTs support three operations: an *insert* operation that associates a given key with a given value and stores the key-value pair on a peer in the system; a *lookup* operation that retrieves the value associated with a given key; and a *delete* operation that removes from the system a given key together with its associated value.

Every peer in a DHT is assigned a *unique identifier* and is responsible for the maintenance of a part of the key space. Usually, a peer is responsible for the keys that are numerically closest to its own identifier.

The three operations, insert, lookup, and delete, require multi-hop routing between peers in order to access keys and values stored by the system. Typically, DHTs support routing from a peer to any other peer in the system in $O(\log N)$ overlay hops, while every peer maintains $O(\log N)$ neighbours, where N is the number of peers in the system.

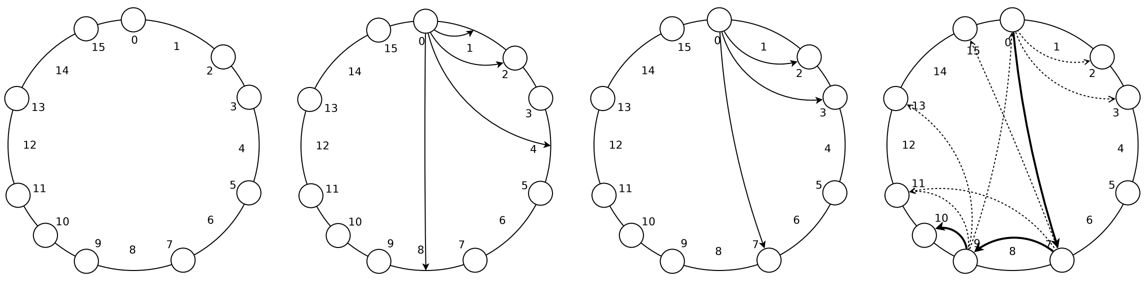
2.4.1 Chord

Chord [179, 180] is one of the the earliest, most often cited, and most popular to date DHT systems. In Chord, every peer calculates its m-bit identifier by applying the SHA-1 hash function to its IP address. Similarly, every key is assigned an m-bit identifier generated by hashing this key. In some variations of Chord, the key's identifier is identical to the key itself and is obtained by hashing the value associated with this key. Due to the properties of the SHA-1 functions, both peer identifiers and key identifiers are scattered evenly in the identifier space, and the probability of generating duplicated identifiers for different peers or keys is extremely low.

The identifiers are ordered in an *identifier circle* modulo 2^m. The *successor* of identifier k is defined as the first peer whose identifier is equal to or follows k in the identifier circle. Similarly, the *predecessor* of k is the first peer whose identifier precedes k in the identifier circle.

The keys, and the values associated with them, are assigned to peers in the system using consistent hashing. The basic rule is that key k is assigned to the predecessor of k. This scheme has the property that keys are distributed evenly between peers, and the addition or removal of a peer in the system does not change the mapping of keys to peers significantly. In particular, in a system with N peers and K keys, with high probability, every peer is responsible for at most $O(K/N \cdot \log N)$ keys, and when a peer joins or leaves the system, $O(K/N)$ keys are re-assigned between peers, and only between the joining or leaving peer and its neighbours [179].

Figure 2.11(a) shows an example Chord system with a 4-bit identifier space ($m = 4$) and nine



(a) Identifier space and node distribution

(b) Finger identifiers of peer 0

(c) Finger neighbours of peer 0

(d) Routing path from peer 0 to peer 10

Figure 2.11: Sample Chord topology over a 4-bit identifier space.

peers. Since all operations are performed modulo 2^4, identifier 15 is followed by 0. Each identifier is assigned to the closest preceding peer, and hence, peer 0 is responsible for identifiers 0 and 1, peer 2 is responsible for identifier 2, peer 3 is responsible for identifiers 3 and 4, and so on.

Peers in Chord connect to their successors and predecessors by which they organise into a ring topology. In some variants of Chord, peers maintain lists of successors and predecessors, i.e., lists of peers that immediately precede and follow them in the identifier space, in order to improve the system's fault-tolerance. The ring topology enables a simple routing algorithm, where peers forward messages to their successors until each message reaches its destination. However, in this simple routing strategy, a message is forwarded on average between $O(N)$ peers before it is delivered.

In order to improve the routing performance, peers maintain additional neighbours that act as "shortcuts" in the ring topology. These neighbours are selected using a *finger table*, which consists of m identifiers. For a peer identified by n, the finger table contains

$$n + 2^0, \quad n + 2^1, \quad \ldots \; n + 2^{m-1}$$

where all arithmetic is modulo 2^m. In generality, the i'th entry in the finger table of peer n is $n+2^{i-1}$, where $0 \leq i < m$. The entries in the finger table do not change over time, since peer identifiers are constant. Figure 2.11(b) shows a sample finger table in the 4-bit Chord system. For peer 0, the table contains identifiers 1, 2, 4, and 8.

For each identifier k in the finger table, each peer maintains a connection to the peer that is responsible for k, i.e., to the predecessor of k. Such a neighbour, associated with the i'th entry in the finger table, is called the i'th finger neighbour, or simply the i'th finger. Unlike entries in the finger table, finger neighbours depend on the system configuration and may change over time.

The maximum number of finger neighbours of a peer is m. However, if the size of the identifier space (i.e., 2^m) is significantly larger than the total number of peers in the system, it is likely that a peer is responsible for the first entries in its own finger table. In this case, the number of peer's finger neighbours is lower than m. It can be shown that in a system with N peers, the average number of neighbours per peer in Chord is $O(\log N)$.

Figure 2.11(c) shows finger neighbours of peer 0 in the sample Chord system. The neighbours, which are peers 2, 3 and 7, corresponds to identifiers 2, 4 and 8 in the finger table of peer 0, as shown in picture (b). Peer 0 has only three finger neighbours, as no neighbour is associated with the first identifier in the finger table, i.e., 1.

The structure of finger connections enables a very efficient routing algorithm, where a message can be routed between any two peers in the system in $O(\log N)$ overlay hops. A peer routes a message addressed to k by forwarding it to the neighbour that most immediately precedes k in the identifier space. This way, each message is first routed over long (in terms of identifier distance) finger connections, and is gradually forwarded over shorter links, until it finds its destination.

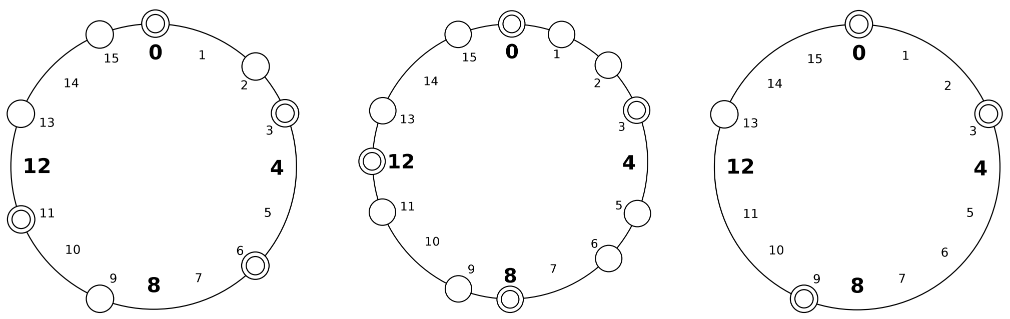
Figure 2.11(d) shows a sample message routed from peer 0 to peer 10. Dashed lines represent peers' finger neighbours, and solid lines indicate the selected routing path. The message is first forwarded to peer 7, using the longest finger connection of peer 0, and is next forwarded to peers 9 and 10 using shorter-distance connections.

The algorithm that maintains the Chord topology, known as the stabilisation algorithm, as well as the details of the routing algorithm, are described in [179].

2.4.2 Other DHT Systems

A number of other DHT systems have been proposed, including Pastry [162], CAN [149], Tapestry [207], Kademlia [118], Kelips [71], P-Grid [1], and Symphony [115]. DHTs have been shown in both theoretical and experimental evaluations to exhibit good performance and stability in the presence of high peer churn, and scalability to millions of participating peers. In particular, DHTs usually provide routing between any peers in the system in $O(\log N)$ hops, given the system size N, and have an expected cost of insert, delete, and lookup operations of $O(\log N)$ message transmissions. Viceroy [114] is the first system that achieved routing in $O(\log N)$ hops with $O(1)$ neighbours per peer. Kaashoek and Karger [85] show that the lower bound for routing in a DHT is $O(\log N)$ hops per lookup request with 2 neighbours per node, and $O(\log \log N / \log N)$ hops per lookup request with $O(\log N)$ neighbours per node. These bounds are met in the Koorde system [85]. A good comparison of several DHT systems can be found in [100] and [153].

Kad, based on Kademlia [118], is considered the largest DHT system that has been implemented and deployed in the Internet. As a part of eMule, a popular file-sharing application, Kad has been reported to support over one million simultaneous users [181]. Bamboo [154], also known as OpenDHT, an open-source implementation of Pastry [162], has been shown to handle higher peer churn rates than other state-of-the-art DHT systems [153]. It has been deployed on PlanetLab [38], a wide-area testbed for P2P systems.



(a) A sample system (b) Increased number of peers (c) Decreased number of peers

Figure 2.12: Super-peer election in a 4-bit Chord system using SOLE.

2.4.3 SOLE

An approach to super-peer election based on a DHT, called Scalable Supernode Selection (SOLE), is described in [105]. The approach relies on the DHT functionality, but does not require any particular DHT scheme and can be customised to any DHT system. It allows a P2P system to elect and maintain K super-peers, where K is a global system constant.

The main idea of SOLE is relatively simple. The system defines K *super-peer identifiers,* and every peer that is responsible for at least one super-peer identifier, according to the mapping provided by the DHT, becomes a super-peer.

The definition of the super-peer identifiers depends on the type of the DHT. If Chord is used as the underlying DHT, the super-peer identifiers can be defined as

$$0,\ \left\lfloor \frac{2^m}{K} \right\rfloor,\ \left\lfloor 2\frac{2^m}{K} \right\rfloor,\ \left\lfloor 3\frac{2^m}{K} \right\rfloor,\ \ldots\ \left\lfloor (K-1)\frac{2^m}{K} \right\rfloor$$

where 2^m is the size of the identifier space. The i'th super-peer identifier is $\lfloor \frac{i}{K} 2^m \rfloor$, for $0 \leq i < K$. This way, the super-peer identifiers divide the DHT identifier circle into K approximately equal arcs, each containing $2^m/K$ identifiers ($\lfloor 2^m/K \rfloor$ or $\lceil 2^m/K \rceil$, to be precise). Each peer p belongs to exactly one arc, and using simple arithmetic, peer p can calculate the super-peer identifier k that directly follows it in the DHT identifier circle. Furthermore, the peer can perform a DHT lookup in order to discover the super-peer responsible for k. If k is assigned to the peer itself, the peer becomes a super-peer. Otherwise, the peer becomes a client of the super-peer responsible for k.

Figure 2.12(a) shows an example Chord system with four super-peers ($K = 4$) elected using SOLE. The super-peer identifiers are 0, 4, 8 and 12. Peer 0 is elected super-peer as it is responsible for super-peer identifier 0. Similarly, peers 3, 6 and 11 are elected super-peers as they are responsible for identifiers 4, 8 and 12, respectively. Figure 2.12(b) shows the same system with extra peers added. As the number of peers is increased, the super-peers are distributed more evenly in the identifier space.

```
 1: Super-Peer:
 2: $q \leftarrow$ lookup super-peer id
 3: if $q \neq p$ then
 4:     request $q$ to become super-peer
 5:     transfer all clients to $q$
 6:     become client of $q$
 7: end if
 8: Client:
 9: if super-peer unavailable then
10:     $q \leftarrow$ lookup super-peer id
11:     if $q = p$ then
12:         become super-peer
13:     else
14:         become client of $q$
15:     end if
16: end if
```

Figure 2.13: SOLE super-peer election algorithm.

Figure 2.12(c) shows the same system with a fraction of peers removed. In this scenario, the number of super-peers is lower than K, as multiple super-peer identifiers are assigned to the a single peer. In particular, both identifiers 4 and 8 are assigned to peer 3.

Lo et al. [105] describe variants of SOLE that use other DHT systems than Chord, such as Pastry and CAN. The general structure of the SOLE algorithm is shown in Figure 2.13. A super-peer periodically checks if it is responsible for its super-peer identifier (line 2). If the identifier is assigned to a different peer q (line 3), the peer appoints q as a new super-peer (line 4), transfers all clients to q (line 5), and becomes a client of q (line 6). A client performs a DHT lookup only when it is not assigned to any super-peer, for example when it is joining the system or when the client's previous super-peer has become unavailable (lines 9). The client determines peer q responsible for the closest super-peer identifier (line 10), and depending on the result, it either becomes a super-peer (line 12), or connects to q as a client (line 14).

Since the super-peer identifiers divide the key space into K equal parts, and due to the properties of Chord, the distance from a peer to its super-peer is $O(\log(N/K))$ overlay hops. Hence, the lookup operation needed to discover a super-peer requires $O(\log(N/K))$ message transmissions, given N peers in the system.

The number of super-peers elected by SOLE does not exceed K, as there are K super-peer identifiers and each of them can be assigned to at most one peer. Furthermore, if every arc in the identifier circle contains at least one peer, the number of elected super-peers is equal to K. Given that peer identifiers are uniformly distributed in the identifier space, the probability that no peers belong to a particular arc is $(\frac{K-1}{K})^N$. The probability that every arch contains at least one peer can be then estimated as greater or equal to $1 - K(\frac{K-1}{K})^N$. Hence, if $N \gg K$, with high probability, the algorithm elects K super-peers.

The main advantages of SOLE are its simplicity and wide-applicability. The use of DHTs assures full system decentralisation, high resilience, and good scalability. The algorithm enables the restriction of the number of super-peers in the system, balances the load between super-peers, and provides an efficient super-peer discovery mechanism.

However, the algorithm has a number of disadvantages. First of all, it elects super-peers based on

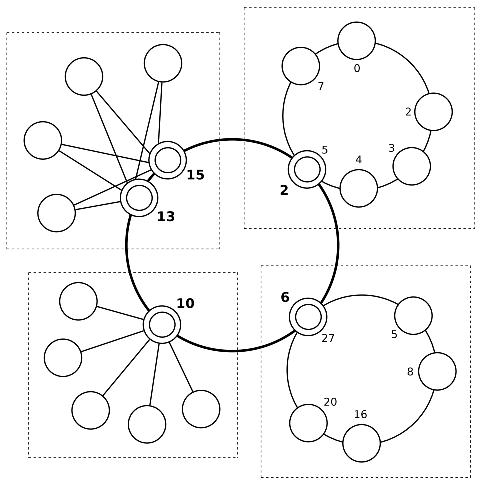


Figure 2.14: Hierarchical DHT.

DHT identifiers rather than their performance or capacity. If peer identifiers are generated randomly or using hash functions, as in many DHTs, the election of super-peer can be seen as a random process, where all peers, including the lowest-performance peers, have equal probability of becoming super-peers. Lo et al. [105] mention criteria for super-peer election, such as high CPU speed, high network connectivity, and high amount of other resources, but it is not clear how the additional super-peer criteria can be combined with the DHT-based super-peer election and discovery algorithms.

Another disadvantage of SOLE is that it elects K super-peers, where K is a predefined value, known to every peer, which cannot be easily changed at runtime. The algorithm does not allow super-peer addition in response to an increased system size or load. Furthermore, SOLE requires that every peer in the system participates in the DHT, which may cause a high load on the lowest performance peers.

2.4.4 Hierarchical DHT

In hierarchical DHT systems [59], peers are organised into groups, and each group maintains its own autonomous overlay network. Each group also elects one or more super-peers, and the super-peers maintain a top-level DHT overlay that connects all peer groups. The communication between peers within groups is handled by the intra-group overlays. The communication between peers located in different groups is relayed by super-peers over the DHT. In order to communicate with peers in other groups, a peer contacts the super-peer in its group, the super-peer routes the message over the DHT to the super-peer in the destination group, and the message is forwarded to the destination peer. Figure 2.14 shows a sample hierarchical DHT system that consists of four groups connected by Chord.

The groups may be heterogeneous. Each group, including the top-level group, may establish a different P2P overlay. In particular, peers in a group may deploy an internal DHT overlay, as in two right-hand-side groups shown in Figure 2.14. The advantage of this approach is that intra-group communication is fully decentralised between all peers in the group, and hence the group can scale to a large number of peers. Alternatively, all peers in a group can directly connect to their super-peer, forming a local star topology, where all intra-group communication, as well as inter-group communication, is handled by the group's super-peer. This configuration is shown in Figure 2.14 in the bottom-left group. The advantage of this approach is that the load on clients is very low, but it has a drawback that the super-peer can easily become a performance bottleneck limiting the size of the group. One more approach is to elect multiple super-peers within each group, as shown in in Figure 2.14 in the top-left group.

Depending on the application-specific requirements, peers in a group may or may not be topologically close to each other. It is assumed that every peer belongs to exactly one group. A peer p is uniquely identified by a pair (id_g, id_p), where id_g is the identifier of peer's group g and id_p is the peer's identifier used for the communication within the group. Garcés-Erice et al. [59] do not specify how peer identifiers and group identifiers are created and how peers are assigned to groups. They only mention that the group identifier of a peer may correspond to the peer's local ISP or university campus. A more self-managing approach would be to generate group identifiers based on peer IP addresses, or using a virtual coordinate system, as described in section 2.3.

Each group elects its super-peers from the "most powerful" peers available in the group. The super-peer election criteria are based on peer properties such as high uptime and good connectivity (primarily), and high CPU power and network connection bandwidth (secondarily). It is assumed that higher uptime peers are more likely to stay on-line in the future. By selecting the most stable and bandwidth-rich peers as super-peers, the system improves the reliability and throughput of the inter-group overlay. Super-peers also cache values frequently accessed in the top-level DHT by the super-peers' clients. If peer groups are organised based on proximity, such a caching strategy can improve data access performance.

When peers join the system, they follow a similar procedure to that described in section 2.3 (as shown in Figure 2.4). In order to join, a peer p is required to know at least one other peer q that already participates in the system. Peer p obtains the address of a super-peer s in its group g by querying peer q, and becomes a client of s. If no super-peer in group g exists, p joins as a super-peer. In a configuration with m super-peers per group, the first m peers that join a group become group super-peers.

Garcés-Erice et al. [59] introduce an interesting approach to handling super-peer failures. Peers run the algorithm shown in Figure 2.15. Each super-peer maintains a list L that contains all peers that belong to the super-peer's group (line 4). The super-peer periodically probes its clients in order to keep the list of peers up-to-date. The list is sorted by peer characteristics, from the best candidates for super-peers to the worst candidates. Furthermore, each super-peer creates a list S that contains

```
1: Super-Peer:
2: loop
3:     wait $\delta$ time units
4:     $L \leftarrow$ list of all peers in the group
5:     $S \leftarrow$ list of super-peer neighbours
6:     broadcast $(L, S)$ to all peers in group $g$
7: end loop
```

(a) Super-Peer

```
 8: Client:
 9: if super-peer unavailable then
10:     $q \leftarrow$ highest uptime peer on list $L$
11:     if $q = p$ then
12:         become super-peer for the group
13:         join super-peer overlay using $S$
14:     else
15:         become client of $q$
16:     end if
17: end if
```

(b) Client

Figure 2.15: Super-peer election algorithm in hierarchical DHT systems.

neighbours of this super-peer in the top-level overlay (line 5). These two lists, L and S, are periodically broadcast by the super-peer to all peers in the group (line 6). Each client also obtains a copy of the two lists when joining its group.

In case of a super-peer failure or departure from the system, each client p checks the first entry on its list L (lines 9-10). The first peer on the list, q, becomes a new super-peer for the group (line 12) and joins the top-level super-peer overlay by contacting super-peers on its list S (line 13). All other peers connect to q and become its clients (line 15). If a super-peer fails and the lists L and S are not available at a peer p, peer p executes a join procedure as if it was joining the system for the first time.

The main open issue in hierarchical DHTs, as they are described in [59], is the mechanism for the group construction and maintenance. Furthermore, although Garcés-Erice et al. assume that multiple super-peers can be elected in a single peer group, they do not specify how clients are distributed between such multiple super-peers and how lists L and S are generated and synchronised between multiple super-peers in a peer group. In many aspects, hierarchical DHTs are similar to other group-based systems described in section 2.3, and most points made in the discussion in section 2.3.4 are also valid for hierarchical DHTs. In particular, hierarchical DHTs with static structures of groups may not scale well with the system size, do not guarantee load balancing between groups, and do not assure optimal super-peer election in terms of super-peer capabilities.

Hierarchical DHTs can be generalised to an arbitrary number of levels. For example, in a three-level hierarchy, each group of peers elects a super-peer, and groups of super-peers elect higher-level super-peers (super-super-peers), which connect through a top-level overlay. However, in this design, not only must peers be assigned to groups, but also super-peers must be assigned to second-level groups (super-peer groups) in order to elect third-level super-peers (super-super-peers). More generally, a mapping must be defined between peers and groups at each level of the hierarchy, excluding the top level. The more groups and levels in the hierarchy, the more knowledge is required at peers to construct and maintain the hierarchy. The problem of group definition and peer assignment to groups is not discussed in [59].

2.4.5 HONet

A similar system to hierarchical DHTs, called Hybrid Overlay Network (HONet), is proposed by Tian et al. in [184]. A HONet system consists of non-overlapping peer groups, called clusters. Every group elects a super-peer, called root node, and the super-peers connect with each other forming a top-level overlay that binds all the groups. Peers in each group also maintain a local overlay, as in hierarchical DHTs. Both the intra-group overlays and the top-level overlay are DHTs with independent identifier spaces. Every peer is uniquely identified by a cluster identifier (CID), which is the local cluster root's identifier in the top-level DHT, and a member identifier (MID), which is the peer's identifier in the cluster-level DHT.

Clusters are created using an Internet coordinate system, such as GNP [136] and Vivaldi [45] described at the beginning of section 2.3.1. When a peer p joins the system, it determines its coordinates in the virtual space, and calculates its identifier in the top-level DHT, *id*, by projecting the k-dimensional coordinates onto the one-dimensional space of the top-level DHT using a locality-preserving Space-Filling Curve, such as the z-order or Hilbert curve described in section 2.3.1. Next, it performs a lookup in the top-level DHT and discovers the closest peer to *id*, denoted q. If the distance in the DHT identifier space between p and q is below a threshold T, p assumes that it belongs to the same cluster as q and becomes a client of q. Due to the properties of the identifier space and the SFC mapping, p and q are located in close physical proximity. If the distance between p and q is higher than T, p joins the system as a super-peer and creates its own cluster. In this case, it sets *id* as its CID.

Apart from DHT connections, every peer maintains random *shortcut* links to peers located in different clusters. The number of such links depends on the peer's capacity, and these links are created using a customised random walk algorithm. The system provides two routing algorithms: *hierarchical routing*, where messages are transferred over both the top-level DHT and cluster-level DHTs, as in the hierarchical DHTs, and *fast routing*, where messages are transferred directly between clusters using shortcut links, if they are available, bypassing the super-peer overlay.

According to [184], "each cluster can choose the most stable member to serve as cluster root, resulting in improved global and local stability". However, it is not obvious how a cluster can change its super-peer from a less stable peer to a more stable peer, since the algorithm described in [184] does not take into account any other peer characteristics than coordinates. Similarly, it is not obvious how the system addresses super-peer failures and departures. If a super-peer is changed in a cluster, the distance between the new super-peer and some of the cluster members may increase above T. In such a case, the cluster may need to elect more than one super-peer in order to cover all peers. Ultimately, the system needs to find a trade-off between electing the highest-capability super-peers in the system

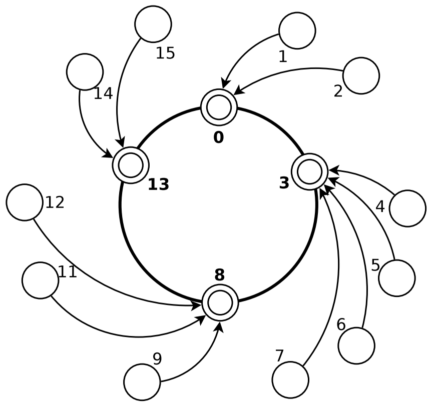


Figure 2.16: Sample SPChord system.

and electing super-peers that are evenly distributed in the DHT identifier space.

This leads to another issue associated with HONet. Almost all DHTs rely on a uniform distribution of peer identifiers in the DHT space. This is required to assure efficient routing and balanced load in the DHT. However, in HONet, peer identifiers are calculated based on peer coordinates, which depend on the physical peer location. If a large number of peers belong to one area, the distribution of DHT identifiers becomes skewed and the DHT may exhibit suboptimum performance.

2.4.6 SPChord

In Super-Peer Chord (SPChord) [101], super-peers maintain a Chord overlay with an m-bit identifier space, and the remaining peers connect to them as clients. Every peer generates its own m-bit unique identifier, including clients, and a client connects to the super-peer that is responsible for this client's identifier in the DHT overlay. Figure 2.16(a) shown a sample SPChord topology with four super-peers and eleven clients distributed between them.

Super-peers are selected from the most stable peers. Peer uptime is used as a predictor for peer stability, and it is assumed that higher-uptime peers are more likely to stay on-line than lower-uptime peers.

Peer identifiers are generated in a two-step process. First, each peer determines its coordinates in a virtual coordinate system, such as GNP [136] and Vivaldi [45] described in section 2.3.1. Second, each peer maps its virtual coordinates onto the m-bit Chord identifier space using space-filling curves that preserve locality, as described in section 2.3.1. This way, super-peers that are close in the physical network are located close to each other in the DHT identifier space, and are associated with clients that are in physical network proximity.

Figure 2.17 shows the algorithm executed at each peer p in SPChord. The join procedure is described in lines 1-8. The first peer that joins the system becomes a super-peer (lines 2-3). All other joining peers perform a DHT lookup in order to determine the super-peer responsible for their

```
 1: Join:
 2: if first peer in the system then
 3:    become super-peer
 4: else
 5:    $q \leftarrow$ lookup own identifier
 6:    become client of $q$
 7:    obtain $L$ and $S$ from $q$
 8: end if

 9: Client:
10: if super-peer unavailable then
11:    $q \leftarrow$ highest uptime peer on list $L$
12:    if $q = p$ then
13:       become super-peer
14:    else
15:       if $p$ follows $q$ then
16:          become client of $q$
17:       else
18:          become client of $q$'s predecessor
19:       end if
20:    end if
21: end if

22: Super-Peer:
23: if overloaded then
24:    $q \leftarrow$ highest uptime client
25:    request $q$ to become super-peer
26:    for each client $c$ do
27:       if $c$ follows $q$ then
28:          transfer $c$ to $q$
29:       end if
30:    end for
31: end if
32: if underloaded then
33:    $r \leftarrow$ predecessor
34:    if $r$ able to handle $p$'s load then
35:       transfer all clients to $r$
36:       become client of $r$
37:       exit
38:    end if
39: end if
40: $L \leftarrow$ list of all peers in the group
41: $S \leftarrow$ list of super-peer neighbours
42: broadcast $(L, S)$ to all clients
```

Figure 2.17: SPChord algorithm at peer p.

identifier (line 5) and become clients of this super-peer (line 6).

SPChord uses the same approach for handling super-peer failures as the hierarchical DHT described in the previous section. Every super-peer maintains a list of its clients, L, and a list of its neighbours in the DHT, S. The lists are periodically broadcast to all clients of the super-peer (lines 36-38), and a joining peer obtains the lists L and S from the super-peer it connects to (line 7). If a super-peer becomes unavailable, the highest-uptime client, q, on list L takes over the role of the super-peer (lines 10-13). All other clients connect either to q or the predecessor of q, depending on their identifiers, so that each client p is associated with the super-peer that is responsible for p's identifier in the DHT (lines 15-19).

The most novel element of SPChord is the algorithm that creates super-peers based on load. If a super-peer p is overloaded (line 23), it selects its highest-uptime client, q, as a new super-peer (lines 24-26), introduces q to the DHT overlay, and splits its remaining clients between itself and q, based on the clients' DHT identifiers (lines 26-30). According to [101], a super-peer is overloaded when the number of its clients exceeds c, where c is a system parameter, equal for all super-peers. The algorithm can be easily extended to allow for individual limits in the number of clients per each super-peer.

A similar algorithm can be used to reduce the number of super-peers when the load is low. When a super-peer p is underloaded (e.g., has fewer clients than $c/2$), the super-peer attempts to merge its cluster with the cluster of its predecessor, r (lines 32-39). However, before p can delegate its clients to r, it must check if r can handle the extra load without splitting its own cluster (line 34). This extra

```
1: Adjust:
2: $q \leftarrow$ select maximum uptime client
3: $id \leftarrow$ generate random DHT identifier
4: $r \leftarrow$ lookup super-peer responsible for $id$
5: if $q$'s uptime higher than $r$'s uptime then
6:     request $q$ to become super-peer
7:     swap $q$'s and $r$'s DHT identifiers
8:     transfer all connections from $r$ to $q$
9:     accept $r$ as client
10: end if
```

Figure 2.18: SPChord topology adjustment algorithm at super-peer p.

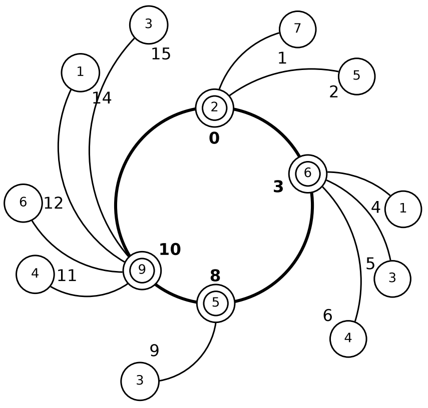


Figure 2.19: Sample configuration generated by SPChord.

check is necessary to avoid a cyclic behaviour of super-peers, where p continuously merges its cluster with r and r continuously splits its cluster and selects p as a super-peer. If r can accommodate p's clients without exceeding its maximum load limit, p migrates all clients to r and becomes a client of r (lines 35-36).

In the absence of peer failures and departures, the peer join procedure and the cluster splitting mechanism guarantee that the highest-uptime peer in each cluster is elected super-peer. However, these algorithms do not guarantee that the uptime of super-peers is globally maximised. For that reason, Liu et al. [101] propose a topology adjustment algorithm, periodically performed by every super-peer p in the system, as shown in Figure 2.18. In this algorithm, each super-peer p selects its highest uptime client, q (line 2), and a random super-peer, r, discovered by performing a DHT lookup on a randomly generated peer identifier (lines 3-4). If q's uptime is higher than that of r (line 5), the roles of q and r are reversed. Client q becomes a super-peer (line 6), the DHT identifiers and neighbour connections of q and r are swapped (lines 7-8), and r is demoted to a client of p (line 9).

The topology adjustment algorithm probabilistically improves the uptime of super-peers, but it also introduces a significant maintenance overhead. Each time a super-peer is swapped with a higher-uptime client, the super-peer needs to notify all its clients and all its DHT neighbours about its identity change. Such frequent changes in the topology may affect the DHT's performance and its stabilisation cost. Since the evaluation of SPChord described in [101] is based on relatively simple simulation experiments, it is not obvious if the proposed algorithm is feasible in a large-scale P2P environment.

As in HONets, peer identifiers in SPChord are calculated based on peer location so that peers that are located close to each other in the physical network also have close DHT identifiers. This way, the system reduces the communication cost for super-peers and their clients. However, many DHT systems, including Chord, require a uniform distribution of peer identifiers in order to provide logarithmic routing and even load distribution between peers. If the distribution of peer locations in the virtual coordinate space is non-uniform, the DHT overlay is likely to suffer reduced performance.

SPChord regulates the number of super-peers according to the system size. When new peers join the system, it creates super-peers, and when peers leave the system, it removes super-peers. Furthermore, it elects super-peers from the highest-uptime clients. However, even in a stable peer population with no arrivals, departures and failures, SPChord does not balance clients evenly between super-peers, hence does not reduce the number of super-peers in the system to minimum, and hence does not guarantee that super-peers have globally-highest uptime.

This is because clients in SPChord are assigned to super-peers based on their DHT identifiers, which imposes additional constraints on the configurations that the system can generate. For example, if two high-uptime peers have very close DHT identifiers, the system can elect both peers as super-peers, but in this case the preceding peer is very likely to have very few or no clients due to the short distance to the other super-peer. Conversely, if the system elects only one of these two peers as a super-peer, the uptime of super-peers in the system is not maximised. Hence, the system can only choose between electing super-peers with maximum uptime, and electing super-peers that are evenly distributed in the DHT space, which is required to balance clients evenly between super-peers and to minimise the number of super-peers in the network. The topology adjustment algorithm can improve the uptime of super-peers, but it does not assign clients uniformly to super-peers.

This is further illustrated in Figure 2.19, which shows a sample SPChord system. The numbers written on peers, within the circles, represent peers' uptimes. The numbers written next to peers, outside the circles, are peers' DHT identifiers. The maximum number of clients per super-peer is 4. Given that there are 14 peers in the system, 3 super-peers are sufficient to handle the remaining 11 peers as clients. However, SPChord elects 4 super-peers, and due to the constraints imposed by the DHT, cannot reduce this number to 3. Super-peer 0 (with uptime 2) cannot merge its cluster with super-peer 3 (uptime 6) or super-peer 10 (uptime 9), as this would require creating a cluster with more than 4 clients. Similarly, super-peer 8 (uptime 5) cannot merge its cluster with the neighbouring super-peers.

Due to the failures of super-peers and merging of clusters, some clients in SPChord may have higher uptimes than their super-peers. In the presented example, the highest-uptime peers in the system are 10, 1, 3 and 12, with uptimes values of 9, 7, 6 and 6, respectively. However, the uptimes of elected super-peers are 9, 6, 5 and 2. The topology adjustment algorithm can gradually swap super-peers with clients and eventually achieve a configuration where peers 10, 1, 3 and 12 are elected super-peers. However, the adjustment algorithm is not able to balance the clients evenly between the super-peers, and hence is not able to reduce the total number of super-peers to 3.

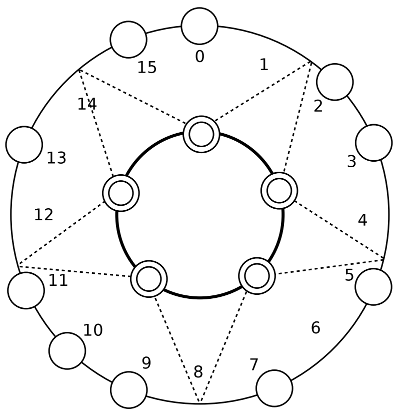


Figure 2.20: Structured Superpeers example.

2.4.7 Structured Superpeers

Structured Superpeers, a system introduced by Mizrak et al. [123], is in many aspects similar to SPChord. In Structured Superpeers, every peer generates a unique identifier and participates in a global Chord overlay called *outer ring*. The system also elects super-peers, which maintain an additional, fully-connected overlay, called *inner ring*. Every super-peer has a full information about all other super-peers in the inner ring. The outer ring is divided into arcs, and each arc is assigned to one super-peer in the inner ring. Figure 2.20 shows an example of such a topology, where the outer ring is split between five super-peers.

The system guarantees routing between any two peers in $O(1)$ overlay hops. In order to send a message from peer p to peer q, peer p forwards the message to its super-peer s. Due to the full-connectivity of the inner ring, super-peer s locates super-peer s' responsible for the arc enclosing q and forwards the message to s'. Super-peer s' delivers the message to q.

The outer ring is not used for routing messages. It's purpose is to improve the network's robustness, as it prevents peer isolation and overlay partitioning in case of a super-peer failure. It is also used by peers to monitor each other and detect potential failures. Moreover, for increased fault-tolerance, super-peers replicate their state on their neighbours in the inner ring.

Super-peers are elected using a *volunteer service.* Mizrak et al. [123] do not specify how the volunteer service is implemented, but only require that every peer joining the system registers in the service, and that the service, when queried, returns the best candidates for super-peers available amongst clients. It is assumed that every super-peer knows its maximum capacity and is able to measure its load. When the load on a super-peer approaches the maximum capacity, two scenarios are possible. If the load on the super-peer's neighbours is below a threshold, the super-peer shares its load with the neighbours using a load-balancing algorithm. Otherwise, the super-peer splits its outer ring arc and creates a new super-peer, selected using the volunteer service. Analogously, when the load on a super-peer is lower than a certain threshold, a super-peer may dismiss one of the existing super-peers and return it to the volunteer service.

The description of Structured Superpeers, published by Mizrak et al. [123], is brief and lacks detail. The main element missing from the description is the algorithm used by super-peers to synchronise their views on the outer ring division. Every super-peer needs to maintain full knowledge of other super-peers in the system and their corresponding outer ring arcs in order to provide constant time routing. Given that super-peers are created and removed on demand, and arcs are dynamically split and merged, it is not known how the information about changes in the arc division is propagated between super-peers. A straight-forward solution would require either a centralised coordinator or an expensive super-peer synchronisation protocol. Furthermore, it is not clear if clients are migrated between super-peers when they are sharing load, and it is not obvious how to implement the volunteer service.

2.4.8 Discussion

Systems reviewed in this section use DHT overlays to elect super-peers. In these systems, peers use the DHT overlay to discover other peers that are located close to them in the DHT identifier space, and super-peers are elected locally amongst peers that are close in the DHT space.

The advantage of DHT-based approaches over group-based approaches is that in the DHT-based systems, peer clusters (i.e., super-peers together with their clients) can be easily split and merged at runtime. The DHT manages the information about current cluster division in an efficient and decentralised manner. The system can dynamically adapt the number of super-peers to the current overlay size or load, while at the same time, every client joining the system is able to discover its super-peer and all super-peers agree with each other about the current division of clients between super-peers.

However, the DHT-based approaches have also disadvantages. Due to the constraints imposed by the DHT, the system cannot at the same time elect the highest capability super-peers and distribute clients evenly between the super-peers. Furthermore, if the clients are not evenly balanced between the super-peers, the total number of super-peers in the system cannot be minimised.

For example, if the best candidates for super-peers have close DHT identifiers, which corresponds to the situation where high-capability peers are located in one group in a group-based approaches, the system is forced to choose between electing the highest-capability super-peers, and electing lower-capability super-peers that evenly divide the DHT space. Such a trade-off is particularly likely if the DHT identifiers are not purely random, but are rather generated based on peer properties, such as peer location.

Finally, in some systems described in this section, such as SOLE and Structured Superpeers, every peer is required to participate in a global DHT overlay. However, running a DHT protocol may introduce a significant overhead on the lowest-performance peers. This is important, since in many

P2P systems, super-peers are introduced in order to reduce the load on the lowest-capability peers, allowing a ordinary peer to have only one connection to a super-peer and letting the super-peers handle more expensive protocols. Systems where the DHT overlays are maintained by super-peers only, such as SPChord, are more consistent with this idea.

2.5 Adaptive Super-Peer Election

This section contains reviews of three systems that elect and optimise super-peer sets according to some well defined criteria. In SG-1 [124], the optimality criteria is based on peer capacities. In SG-2 [83], the optimality criteria is based on peer capacities and distances. In DLM [196], the optimality criteria is derived from a file-sharing systems workload model.

2.5.1 SG-1

The goal of the SG-1 algorithm, proposed by Montresor in [124], is to generate and maintain general-purpose super-peer topologies with the following characteristics

(i) every client is associated with exactly one super-peer,

(ii) super-peers are connected through a pseudo-random overlay network,

(iii) the number of super-peers is minimised.

The last condition, (iii), is based on the notion of peer *capacity*. SG-1 assumes that peers are heterogeneous and associates each peer p with a parameter c_p, called capacity, which determines the maximum number of clients that p can handle if elected super-peer. SG-1 also assumes that every peer is able to calculate its capacity upon joining the system and that peer capacity does not change over time. Condition (iii) states that SG-1 generates a topology with minimum number of super-peers such that the total super-peers capacity is higher than or equal to the total number of clients. A P2P topology that satisfies conditions (i-iii) is called an SG-1 target topology.

As many other P2P systems, SG-1 assumes that all peers are mutually reachable through some lower-level network, such as the Internet, and that any peer can potentially connect to any other peer. The SG-1 algorithm is based on periodic gossipping. Every peer periodically exchanges with selected neighbours its information about its current capacity, numbers of clients, and neighbours. In response to an information exchange, a super-peer may transfer its clients to a higher-capacity super-peer, a super-peer may become a client of another super-peer, and a client may be promoted to a super-peer. The general goal of the algorithm is to migrate clients from lower-capacity super-peers to higher-capacity super-peers and to eliminate superfluous super-peers, i.e., those that have no clients.

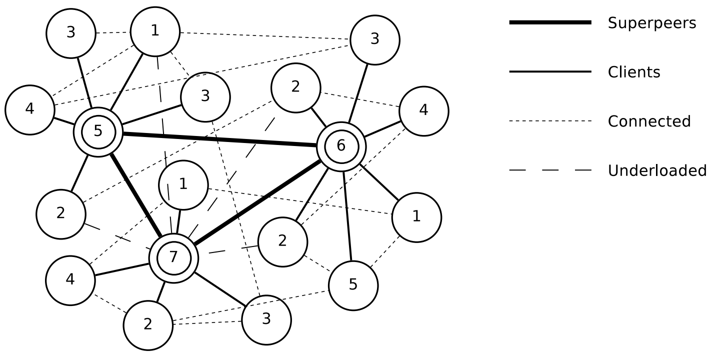


Figure 2.21: Sample SG-1 super-peer topology.

In SG-1, each peer maintains four neighbourhood sets: *connected*, *superpeers*, *underloaded*, and *clients*. The *connected* set contains a random sample of all peers in the system. It is used by peers to exchange information with each other in order to maintain the other three neighbour sets. It also assures full overlay connectivity. The *superpeers* set contains a random sample of super-peers in the system. It is used to establish connectivity between super-peers and is required to generate the target topology. The *underloaded* set contains super-peers that have fewer clients than their capacity value. This set is used in to obtain candidates for client transfers. Finally, the *clients* set manages the relationship between clients and super-peers. For a client, this set contains at most one entry, which represents the current super-peer of this peer. For a super-peer, this set consists of clients currently associated with this super-peer.

Figure 2.21 shows a sample topology generated by the SG-1 protocol. Peer capacity values are indicated by numbers. For clarity, only a subset of peer connections are shown. Super-peers are linked with each other through their *superpeers* neighbour sets, client are linked with their super-peers through *clients* set, and random peers are linked with each other through *connected* sets. Additionally, one super-peer is underloaded (with capacity equal to 7), and a number of peers are connected with this super-peer through their *underloaded* sets.

In order to create and maintain the four neighbourhood sets, each peer periodically runs four neighbour selection algorithms. The *connected* set is maintained using Newscast [80, 82], a gossip-based neighbour selection algorithm that generates approximately random P2P topologies.

2.5.1.1 Newscast

The general structure of Newscast is shown in Figure 2.22(a). Each peer p maintains a limited-size partial view, s_p, which contains its neighbours' descriptors. The maximum size of a partial view, denoted as c, is a system constant. A neighbour descriptor consists of a neighbour address and a timestamp.

It is assumed that a TIMEOUT event is generated every δ time units at peer p, which triggers

```
1:  **loop**
2:      $e \leftarrow$ WaitForEvent()
3:      **if** $e$ is TIMEOUT **then**
4:          $q \leftarrow$ random peer from $s_p$
5:          $s'_p \leftarrow s_p \cup (p, t)$
6:          send request $s'_p$ to $q$
7:      **end if**
8:      **if** $e$ is request $s_q$ from $q$ **then**
9:          add $(p, t)$ to $s_p$
10:         send response $s_p$ to $q$
11:         $s_p \leftarrow$ merge($s_p$, $s_q$)
12:     **end if**
13:     **if** $e$ is response $s_q$ from $q$ **then**
14:         $s_p \leftarrow$ merge($s_p$, $s_q$)
15:     **end if**
16: **end loop**
```

(a) Single-threaded algorithm

```
1:  **Active thread:**
2:  **loop**
3:      wait $\delta$ time units
4:      $q \leftarrow$ random peer from $s_p$
5:      add $(p, t)$ to $s_p$
6:      send $s_p$ to $q$
7:      receive $s_q$ from $q$
8:      $s_p \leftarrow$ merge($s_p$, $s_q$)
9:  **end loop**

10: **Passive thread:**
11: **loop**
12:     receive $s_q$ from a peer $q$
13:     $s'_p \leftarrow s_p \cup (p, t)$
14:     send $s'_p$ to $q$
15:     $s_p \leftarrow$ merge($s_p$, $s_q$)
16: **end loop**
```

(b) Two-threaded algorithm

Figure 2.22: Newscast algorithm at peer p at time t.

the execution of the algorithm. As the event is raised (in line 4 in Figure 2.22(a)), peer p selects a random neighbour q from its partial view (line 5), adds its own address and the current time to the partial view (line 6), and sends a request to the selected neighbour q (line 7). The request contains p's partial view s_p. When p receives a request from a neighbour q (line 9), it adds its own address and the current time to its partial view (line 10), sends a response to q (line 11), and updates its partial view by applying the *merge* operation on s_p and s_q (line 12). The response again contains the partial view of p. Finally, when peer p receives a response from a neighbour q (line 14), it merges its partial view s_p with the received view s_q using the *merge* operation.

Operation $merge(s_p, s_q)$ consists of the following steps. First, all entries in s_p and s_q are combined into one collection, which contains at maximum $2c + 1$ peer descriptors. Second, all duplicated entries are removed from the collection. If multiple descriptors are associated with the same address, only one descriptor with the most recent timestamp is preserved. Finally, the most recent c descriptors are selected from the collection and all other descriptors are discarded.

The Newscast algorithm is also modelled using two threads, as shown in Figure 2.22(b). An active thread initiates a gossip exchange with a random neighbour every δ time units, while a passive thread continuously listens for incoming exchange requests and responds to them. The *merge* operation is identical in both variants of the algorithm. The algorithms are also equivalent in terms of generated topologies and message costs.

Both the *superpeers* and *underloaded* sets are maintained by running two instances of a modified version of Newscast. The modification to the original Newscast algorithm is twofold. First, the neighbour chosen for gossip exchange (in line 4 in both Figure 2.22(a) and Figure 2.22(b)) is selected randomly from the *connected* set, and not from the *superpeers* or *underloaded* set. This way, all peers in the system participate in the dissemination of super-peer and underloaded super-peer information, as the *connected* set contains a random sample of all peers in the system. Second, the peer adds itself to the set of descriptors (in line 5 in Figure 2.22(a) and in lines 5 and 13 in Figure 2.22(b)) only if it satisfies a condition. For the *superpeers* set, this condition is that the peer must be a super-peer, and for the *underloaded* set, the peer must be a super-peer with fewer clients than its capacity.

2.5.1.2 Client sets

The algorithm that maintains the *clients* sets is the most sophisticated component of SG-1. It is run periodically, as the other neighbour selection algorithms in SG-1, and its pseudocode is shown in Figure 2.23(a). Each super-peer periodically invokes the *SuperPeer*() procedure (in line 1 in Figure 2.23(a)), while each client executes the *OrdinaryPeer*() procedure (line 23).

A super-peer p iterates over all entries in its *underloaded* set and attempts to find a candidate for a client exchange (lines 2-10). Such a candidate q must satisfy a number of conditions. First, it must be a super-peer with free client slots. Second, it must either have a higher capacity than that of p, or it must have an equal capacity as p but a higher number of clients than p (lines 3-5).

If a candidate q is found, peer p transfers as many clients to q as q is able to handle (line 13). If p is left with no clients, and q has at least one free client entry (line 14), peer p becomes a client of q (line 15). This way, clients are migrated to a higher-capacity super-peers and the total number of super-peers in the system is reduced.

In the last step, if p still has some clients after an exchange with q (line 16), peer p selects the highest-capacity client of q, denoted as r (line 17), and if r's capacity is higher than the capacity of p (line 18), peer p swaps its role with r (line 19). For this purpose, r becomes a super-peer, p transfers all its clients to r, and p becomes client of r. This step again assures that higher-capacity peers are promoted to super-peers.

The *OrdinaryPeer*() procedure is very simple. Every peer joins the system as a super-peer, and whenever a client loses its super-peer, it becomes a super-peer again (lines 24-26).

2.5.1.3 Discussion

Contrary to the evaluation described in [124], the original version of the SG-1 algorithm does not converge to the target topology[1]. The following scenario can be given as a counterexample. The highest capacity peer in the system, p, becomes a super-peer and accepts the second highest capacity peer in the system, q, as its client. Subsequently, other peers connect to p and the capacity of p

[1]This fact was acknowledged by Alberto Montresor, the author of SG-1, in a private conversation held with the author of this thesis in March 2008.

```
SuperPeer():
for all $q \in underloaded$ do
  if $c_q \geq c_p$ then
    $l_q \leftarrow$ obtain load from $q$
    if $l_q < c_q \wedge (c_q > c_p \vee l_q > l_p)$ then
      $Transfer(q)$
      exit loop
    end if
  end if
end for

Transfer($q$):
transfer $min(c_q - l_q, l_p)$ clients to $q$
if $l_p = 0 \wedge l_q < c_q$ then
  become client of $q$
else
  $r \leftarrow$ maximum capacity client of $q$
  if $c_r > c_p$ then
    swap roles of $p$ and $r$
  end if
end if

OrdinaryPeer():
if $clients = \emptyset$ then
  become super-peer
end if
```

(a) Original version of SG-1

```
SuperPeer():
for all $q \in underloaded$ do
  if $c_q \geq c_p$ then
    obtain load $l_q$ from $q$
    if $l_q < c_q \wedge (c_q > c_p \vee l_q > l_p)$ then
      $Transfer(q)$
      exit loop
    end if
  end if
end for
for all $q \in superpeers$ do
  $r \leftarrow$ maximum capacity client of $p$
  if $c_r > c_q$ then
    swap roles of $q$ and $r$
  end if
end for

Transfer($q$):
transfer $min(c_q - l_q, l_p)$ clients to $q$
if $l_p = 0 \wedge l_q < c_q$ then
  become client of $q$
else
  $r \leftarrow$ maximum capacity client of $q$
  if $c_r > c_p$ then
    swap roles of $p$ and $r$
  end if
end if

OrdinaryPeer():
if $clients = \emptyset$ then
  for all $q \in underloaded$ do
    $l_q \leftarrow$ obtain load from $q$
    if $l_q < c_q$ and $c_p \leq c_q$ then
      become client of $q$
      exit
    end if
  end for
  become super-peer
end if
```

(b) Extended version of SG-1

Figure 2.23: SG-1 algorithm at peer p.

becomes fully utilised. In the absence of failures and peer departures, the configuration becomes stable, as p does not belong to the *underloaded* sets of other peers and does not participate in client exchanges. Peer p is a super-peer while peer q is a client. However, in the target topology, it is very likely that both p and q should be elected super-peers, since they are the two highest-capacity peers in the system.

This problem can be addressed by extending the *clients* set maintenance algorithm, as shown in Figure 2.23(b). It should be noted, however, that the proposed extension does not belong to the original SG-1 algorithm and is only a suggestion of this thesis' author. In the extended SG-1, each super-peer p selects candidates for client transfers (lines 2-10) and performs the $Transfer()$ procedure (lines 18-27) in exactly the same way as in the original version of SG-1. However, two extra steps are added. In lines 11-16, each super-peer p iterates over its entries in the $superpeers$ set and searches for a super-peer q such that its capacity c_q is lower that the capacity c_r of the highest-capacity client r of peer p. This can be done without incurring any extra communication cost. If a suitable super-peer q is found, the roles of q and r are swapped. Client r becomes a new super-peer, all clients of q are transferred to r, and q becomes a client of r. This way, a higher-capacity client replaces a lower-capacity super-peer.

The second extension to the original algorithm, in lines 30-39, allows peers to join the system as clients rather than super-peers. Every client p that is not associated with a super-peer, either when it is joining the system or when it loses a previous super-peer, attempts to find a new super-peer in its *underloaded* set. If a super-peer q with free capacity is found, which has a higher capacity than that of p, peer p becomes a client of q. Otherwise, peer p becomes a super-peer.

It can be shown that a topology managed by the extended SG-1 algorithm, in the absence of peer arrivals, departures, and failures, always converges to the target topology. If churn is present, the algorithm approximates the target topology. SG-1 is also capable of dealing with catastrophic failures, where a large percentage of super-peers (even 100%) are suddenly removed from the network.

SG-1 allows every super-peer to specify its capacity, i.e., the maximum number of clients it can handle. However, in many applications, the load associated with handling clients may vary between different peers and may also change with time. SG-1 does not model this. More importantly, the load on a super-peer may depend not only on the number of clients directly connected to this super-peer, but also on the general activity of other peers in the system. For example, in systems where super-peers handle search, the load on a super-peer is generated by serving its own clients as well as processing queries received from other peers. SG-1 has the drawback that it does not allow the system to explicitly control the number of super-peers and to adapt the super-peers set to the current total demand in the system.

2.5.2 SG-2

SG-2, proposed by Jesi et al. in [83], is an algorithm inspired by SG-1 that generates proximity-aware super-peer topologies. In such topologies, super-peers are elected from the highest-capacity peers, as in SG-1, but the system imposes additional constraints on the maximum distance between peers using a distance metric. Each peer p is assigned a capacity value $cap(p)$, which represents the maximum number of clients it can handle if elected super-peer. Peer capacity is static and it is assumed that every peer knows its own capacity value, as in SG-1. Additionally, for any pair of peers (p, q) the system defines a *latency* distance $lat(p, q)$. SG-2 generates topologies where:

(i) every client is associated with exactly one super-peer,

(ii) the number of clients of any super-peer s does not exceed $cap(s)$,

(iii) the latency between a client and its super-peer does not exceed tot,

(iv) two super-peers are connected if the latency between them is below $tot + \delta$,

(v) the number of super-peers in the system is minimised.

Parameters tot and δ are configurable system constants. The distance metric $lat()$ is defined as the average round-trip time (RTT) between two peers, and is calculated using the Vivaldi virtual coordinate system [45], described in section 2.3.1. It is assumed that every peer is able to determine its distance to any other peer in the system. A topology described by conditions (i-v) is called a SG-2 target topology.

The target topology can be described using geometrical concepts. Each peer in the system is represented as a point in the virtual coordinate space. The *influence zone* of a peer is an n-dimensional sphere of radius tot centred at that peer. The goal of SG-2 is to cover the virtual space with a minimum number of super-peers in such a way that every peer is either a super-peer or belongs to the influence zone of a super-peer.

Figure 2.24 shows a sample topology generated by SG-2 in a two-dimensional Euclidean space. Five super-peers are elected in order to cover all peers in the space, and their influence zones are marked with circles. Each client is connected to the highest-capacity super-peer in its influence zone.

2.5.2.1 Spherecast

In SG-2, peers communicate with each other using a local broadcast service, which efficiently disseminates messages to all peers within the influence zone of the sender. The local broadcast service is provided by Spherecast, a gossip algorithm based on Newscast [82] and Lightweight Probabilistic Broadcast [54].

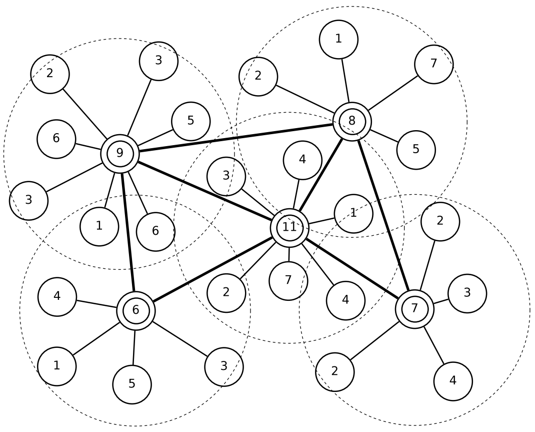


Figure 2.24: Sample topology generated by SG-2 in a 2-dimensional Euclidean space.

In Spherecast, every peer runs an instance of Newscast that manages its set of neighbours. The fan-out set of a peer is defined as the subset of the peer's neighbours that belong to the peer's influence zone. A peer broadcasts a message by sending it to all peers in its fan-out set. When a peer receives a message from a neighbour, it either forwards this message to all neighbours in its fan-out set or drops it. The decision is made probabilistically, based on the number of times the message has been encountered by the peer in the past. A message is dropped with probability $1 - e^{-f/\vartheta}$, where f is the number of previous message occurrences, and ϑ is a constant threshold parameter.

2.5.2.2 SG-2 algorithm

Every client that is not associated with a super-peer periodically broadcasts a "request for super-peer" message, denoted CL-BCAST, in its influence zone. Super-peers reply to these request messages, and clients connect to them. If a client discovers multiple super-peers in its influence zone, it connects to the highest capacity super-peer. Furthermore, a client may probabilistically decide to become a super-peer, depending on its capacity and the frequency of requests received from other clients.

At the same time, every super-peer periodically broadcasts a "super-peer advertisement" message, denoted SP-BCAST, to all neighbours within $tot + \delta$ range in order to announce its presence. Super-peers whose influence zones overlap compete with each other, as clients are migrated from lower-capacity super-peers to higher-capacity super-peers. A super-peer that loses all its clients is demoted to a client.

The three parallel processes, which create super-peers based on demand, transfer clients to high-capacity super-peers, and remove idle super-peers, together generate a topology that approximates the system target topology.

Figure 2.25 shows pseudocode for the SG-2 algorithm. Given that there are two types of peers, super-peers and clients, and two types of messages, CL-BCAST and SP-BCAST, four scenarios of message exchange are possible.

```
1: **Super-peer** $p$ **gets** $\langle$SP-BCAST$, s, cap(s)\rangle$:
2: **if** $cap(p) > cap(s)$ **then**
3:    **for each** client $c$ of $s$ **do**
4:       **if** $lat(c, p) < tot$ **then**
5:          transfer $c$ from $s$ to $p$
6:       **end if**
7:    **end for**
8:    **if** $s$ has no clients **then**
9:       $s$ becomes client
10:      **if** $lat(s, p) < tot$ **then**
11:         $s$ connects to $p$
12:      **end if**
13:   **end if**
14: **end if**

15: **Super-peer** $p$ **gets** $\langle$CL-BCAST$, c\rangle$:
16: accept $c$ as client

17: **Client** $p$ **gets** $\langle$SP-BCAST$, s, cap(s)\rangle$:
18: **if** $p$ has no super-peer **then**
19:    become client of $s$
20: **else if** $cap(super(p)) < cap(s)$ **then**
21:    migrate from $super(p)$ to $s$
22: **end if**

23: **Client** $p$ **gets** $\langle$CL-BCAST$, c\rangle$:
24: become super-peer with probability $\frac{s^2}{s^2+\theta_p^2}$
```

Figure 2.25: SG-2 algorithm.

When a super-peer p receives an SP-CAST message from a super-peer s (line 1 in Figure 2.25), whose capacity $cap(s)$ is lower than $cap(p)$, it requests a client migration (lines 2-14). All clients of s that belong the influence zone of p are transferred to p, provided p has enough free capacity (lines 3-7). If s is left with no clients, it is demoted to a client (lines 8-9), and if it belongs to the influence zone of p, it becomes a client of p (lines 10-12).

When a super-peer p receives a CL-CAST message from a client c, it accepts c as its client, given it has enough free capacity (lines 15-16).

When a client p receives an SP-CAST message from a super-peer s, if p is not currently associated with any super-peer, it connects to s as a client. If p is associated with a super-peer with a lower capacity than the capacity of s, it disconnects from the current super-peer and becomes a client of s, provided s has enough free capacity. Super-peer s can refuse the connection from p if it does not have enough free capacity.

Finally, when a client p receives a CL-CAST message from another client c, it switches its role to a super-peer with probability $\frac{f^2}{f^2+\theta_p^2}$, where f is the number of times p has encountered this message in the past, and θ_p is a threshold variable maintained by p. Intuitively, as the frequency of requests grows, the probability of p becoming a super-peer increases, and as the frequency of requests decreases, the probability of v becoming super-peer decreases. Parameter θ_p is initialised at each peer p as $cap_{max} - cap(p)$, where cap_{max} is the maximum peer capacity in the system, and is periodically updated at each peer according to the following formula

$$\theta_p(t) = \theta_p(t-1) + \alpha(t - t'_p) \tag{2.1}$$

where t is the current time, t'_p is the last time when p became a super-peer, and α is a system parameter. The update formula is used to decrease the probability of role switching between clients and super-peers over time, in order to stabilise the topology and reduce the maintenance overhead.

2.5.2.3 Discussion

According to [83], the problem of finding the target topology, even in a static system, is NP-difficult. In a dynamic environment, with joining and leaving peers and with communication failures, the problem is even more difficult. However, as shown in [83], SG-2 generates topologies where a large majority of clients manage to connect to super-peers in their *tot* latency range and the number of super-peers is close to optimum.

SG-2 uses Spherecast for handling local communication between peers located in physical proximity. However, it is not clear if the Spherecast algorithm scales. When the density of peers in the system grows, more peers belong to each influence zone, and an average peer receives more messages. Similarly, when the range of influence zones is increased, a peer receives on average more messages. If a low-performance peer is located in an area with a high density of other peers, it may easily become overloaded.

On the other hand, when the range of influence zones is decreased, fewer neighbours of a peer belong to the peer's fan-out set. If the system size is large and the range of influence zones is small, the probability that fan-out sets contain any peers may become very low, and as a consequence, Spherecast may stop to work correctly. Thus, the range of peers' influence zones appears to be a critical factor affecting the system's performance. Setting this range appropriately may be non-trivial when deploying an SG-2 system.

2.5.3 Dynamic Layer Management

Another approach to the construction of optimal super-peer topologies is proposed by Xiao et al. in [196]. They ask the following three fundamental questions.

- What is the optimal ratio of super-peers to ordinary peers in a P2P system?
- Which peers should be elected super-peers given an optimal super-peer ratio?
- How to maintain an optimal super-peer ratio in a running P2P system?

In order to address the first question, they introduce a workload model for file-sharing systems, and derive from this model an optimal ratio of super-peers to ordinary peers. In order to address the latter two questions, they introduce Dynamic Layer Management (DLM), a decentralised algorithm that "designates peers with relatively long lifetimes and large capacities as super-peers" and "can maintain an optimal layer size ratio and adaptively elect and adjust peers between superlayer [i.e., super-peers] and leaf layer [i.e., clients]" [196].

2.5.3.1 Workload model

The workload model assumes that every peer stores a collection of files, shared with other peers in the system, and generates a search query with an average frequency f. Super-peers index files stored by their clients and handle search. The model does not depend on any particular search algorithm, but assumes that every search query is propagated to at least p super-peers before the results are returned to the originating peer. A client connects on average to m super-peers, and a super-peer connects on average to k other super-peers. The average durations of client to super-peer and super-peer to super-peer connections are t_l and t_s, respectively.

Ordinary peers are subject to very little message traffic, as they communicate with their super-peers only when updating indices of shared files or issuing search queries or receiving search results. Super-peers are subject to much higher load, as they maintain connections with multiple clients and super-peers, and relay queries received from both their own clients and other super-peers.

Xiao et al. [196] introduce two types of workload. The workload on a super-peer, W_{sp}, is defined as the average message cost incurred by a super-peer when performing a search operation. The workload on the overall network, W_{on}, is defined as the total message cost of an average search operation in the P2P network. Furthermore, Xiao et al. [196] derive the following upper and lower bounds of the two workloads

$$\frac{m\eta}{t_l} + \frac{k}{t_s} + f(p-1) \;\leq\; W_{sp} \;\leq\; \frac{m\eta}{t_l} + \frac{k}{t_s} + f(m\eta + pk) \tag{2.2}$$

$$\frac{n}{1+\eta}\left(\frac{m\eta}{t_l} + \frac{m\eta + k}{t_s} + f(p-1)\right) \;\leq\; W_{on} \;\leq\; \frac{n}{1+\eta}\left(\frac{m\eta}{t_l} + \frac{m\eta + k}{t_s} + f(m\eta + pk)\right) \tag{2.3}$$

where n is the number of peers in the system, and η is the ratio of clients to super-peers. They also introduce a weighted workload, W, as

$$W = \alpha W_{sp} + \beta \frac{W_{on}}{n} \tag{2.4}$$

where α and β are weights such that $\alpha + \beta = 1$. The optimal super-peer ratio, η^*, is defined as a ratio between super-peers and clients that minimises the weighted workload W. It can be estimated using formulas (2.2) and (2.3). For the most efficient search algorithm, which corresponds to the lower bound on the workload, the optimal super-peer ratio is

$$\sqrt{\frac{\beta t_l}{\alpha m}\left(\frac{k-m}{t_s} - \frac{m}{t_l} + f(p-1)\right)} - 1. \tag{2.5}$$

For the least efficient search algorithm, the optimal super-peer ratio is given by

$$\sqrt{\frac{\beta t_l}{\alpha m(1 + f t_l)}\left(\frac{k-m}{t_s} - \frac{m}{t_l} + f(kp - m)\right)} - 1. \tag{2.6}$$

Since m, k and p are defined by the P2P protocol, and t_l and t_s can be measured experimentally, the formula allows the calculation of an optimal super-peer ratio η^*.

```
1: if $p$ is super-peer then
2:     $G_p \leftarrow$ connected clients
3:     $\mu_p \leftarrow log(|G_p|/m\eta^*)$
4: else
5:     $G_p \leftarrow$ known super-peers
6:     $l_p \leftarrow$ number of clients per super-peer in $G_p$
7:     $\mu_p \leftarrow log(l_p/m\eta^*)$
8: end if
9:
10: for all $q \in G_p$ do
11:     if $c_q \cdot X_{c,p} > c_p$ then
12:         $Y_{c,p} \leftarrow Y_{c,p} + 1/|G_p|$
13:     end if
14:     if $a_q \cdot X_{a,p} > a_p$ then
15:         $Y_{a,p} \leftarrow Y_{a,p} + 1/|G_p|$
16:     end if
17: end for
18: if $p$ is super-peer then
19:     if $Y_{c,p} > Z_{c,p}$ and $Y_{a,p} > Z_{a,p}$ then
20:         become client
21:     end if
22: else
23:     if $Y_{c,p} < Z_{c,p}$ and $Y_{a,p} < Z_{a,p}$ then
24:         become super-peer
25:     end if
26: end if
```

Figure 2.26: DLM algorithm.

Using their workload model, Xiao et al. [196] calculate that the optimum number of clients per super-peer in a typical P2P file-sharing application is between 30 and 65. They also notice that this number corresponds to the typical super-peer ratios found in popular file-sharing systems such as KaZaA.

2.5.3.2 DLM algorithm

The Dynamic Layer Management (DLM) algorithm elects super-peers from peers with longer lifetimes and higher capacities and maintains a given ratio η^* of clients to super-peers in a P2P system. Each peer p is assigned a static capacity value, $c(p)$, which reflects the peer's ability to process and relay search queries and search responses, and an age $a(p)$, which is defined as the length of time since the peer joined the system. According to [196], the capacity can be given as a weighted sum of low-level peer properties, such as available bandwidth, CPU speed, and storage space. The age of a peer is used as an estimator of the peer's lifetime. It is expected that peers with higher uptimes are more likely to stay on-line in the future.

In order to elect super-peers and maintain the desired super-peer ratio, each peer performs algorithm shown in Figure 2.26. The algorithm consists of three main blocks. First, each peer collects information about its neighbours, estimates the current super-peer ratio and determines how much it diverges from the optimal ratio η^* (lines 1-8 in Figure 2.26). Second, each peer compares its capacity and age with the corresponding characteristics of its neighbours in order to determine if it is an appropriate candidate for a super-peer (lines 9-16). Finally, each peer decides whether it should become a super-peer or a client (lines 17-25).

Each peer p defines its *related set*, G_p, as a subset of peers in the system that it uses for the estimation of system properties. For a super-peer, the related set is defined as the current set of clients (line 2). For a client, the related set is defined as super-peers that the client knows about (line 5). Xiao et al. [196] suggest that the related set for a client p may be defined as the set of super-peers that p has connected to within the last T time units.

Using G_p, each peer p approximates the average number of client connections per super-peer, l_p. A super-peer simply assumes that l_p is equal to its own number of clients (i.e., $|G_p|$), and a client calculates l_p as an average number of clients per super-peer in G_p (line 6). Given the optimal clients to super-peers ratio, η^*, and the fact that each client connects to m super-peers, the optimal number of client connections per super-peer in the system is $m\eta^*$.

Each peer estimates the divergence of the current system topology from the optimal topology as $\mu_p = log(\frac{l_p}{m\eta^*})$ (lines 3 and 7). A positive value of μ_p indicates that super-peers currently have too many clients, and hence, new super-peers should be elected, while a negative value of μ_p indicates that too many super-peers exist in the system and some of them should be demoted.

Next, each peer p compares its capacity c_p and age a_p with the capacity and age of peers in the related set G_p (lines 10-17). It calculates $Y_{c,p}$ as the fraction of peers in G_p that have a higher capacity than c_p (lines 11-13) and $Y_{a,p}$ as the fraction of peers in G_p that have a higher age than a_p (lines 14-16). In the comparison, the capacity of each peer in G_p is weighted by $X_{c,p}$ (line 11), and the age of each peer in G_p is weighted by $X_{a,p}$ (line 14).

The aim of the scale parameters $X_{c,p}$ and $X_{a,p}$ is to regulate the probability of a super-peer promotion and demotion. However, it is not explained in [196] how $X_{c,p}$ and $X_{a,p}$ are calculated. Xiao et al. only mention that

> $X_{c,p}$ and $X_{a,p}$are adjusted according to the value of μ_p. For a superpeer, if it finds that the system needs more superpeers, it will decrease the possibility of its demotion by decreasing the two scale parameters. Otherwise, it will increase the possibility of its demotion by increasing the scale parameters, while for a leaf peer, if it finds that more superpeers are needed, it will decrease the scale parameters in hoping to increase the promotion possibility; otherwise, it will increase the scale parameters to decrease the promotion possibility. [196]

In the last step, peer p compares its values of $Y_{c,p}$ and $Y_{a,p}$ with threshold values of $Z_{c,p}$ and $Z_{a,p}$ (lines 18-26). A super-peer with both $Y_{c,p}$ and $Y_{a,p}$ higher than $Z_{c,p}$ and $Z_{a,p}$, respectively, becomes a client (lines 19-21). In order to switch its role, it drops all connections to clients and preserves only m connections with selected super-peers. It is not specified in [196] what happens to the disconnected clients and how they find new super-peers to connect to.

A client with the values of both $Y_{c,p}$ and $Y_{a,p}$ above $Z_{c,p}$ and $Z_{a,p}$, accordingly, is promoted to a super-peer (lines 23-25). It preserves its current super-peer connections and starts to accept incoming connections from clients. However, again, it is not entirely clear how $Z_{c,p}$ and $Z_{a,p}$ are calculated.

Xiao et al. state that

> The values of threshold variables, $Z_{c,p}$ and $Z_{a,p}$, are also adjusted according to the value of μ. When more superpeers are needed, superpeers will increase the values of the threshold variables to reduce the demotion tendencies and leaf-peers will reduce the values of the threshold variables to increase the promotion tendencies. For the case there there are too many superpeers, inverse measures will be taken accordingly. [196]

Furthermore, it is not obvious why the algorithm needs both the thresholds variables $Z_{c,p}$ and $Z_{a,p}$ and the scale parameters $X_{c,p}$ and $X_{a,p}$. Their roles appear to be redundant, as they both are used to regulate the probability of super-peer promotion and demotion, and they are both adjusted based on μ_p.

Two extensions to the DLM algorithm, labelled DLM-2 and DLM-3, are proposed in [196]. In DLM-2, all super-peers periodically gossip with each other and exchange information about their clients. This information is used by super-peers to approximate the average number of client connections per super-peer more accurately than in the original version of DLM. An experimental evaluation in [196] shows that DLM-2 achieves better performance than DLM-1 and DLM-3.

In DLM-3, the election algorithm is run only on super-peers in order to reduce load on clients. The decision about a super-peer demotion is made as in the DLM-1 algorithm. Super-peer promotion is managed by existing super-peers. When a super-peer decides a new super-peer is needed, it selects a client q with maximum value of $\gamma_1 c_q + \gamma_2 a_q$, where γ_1 and γ_2 are constant weighing parameters and $\gamma_1 + \gamma_2 = 1$, and promotes q to a super-peer.

2.5.3.3 Discussion

DLM and the super-peer election algorithm used in the gradient topology, described later in this thesis, share a number of similarities. Both algorithms introduce metrics that capture peer capabilities and quantify the suitability of individual peers to become super-peers. In both approaches, the optimum number of super-peers in the system is calculated based on estimated system properties. Furthermore, in both approaches, super-peers are elected using adjustable thresholds.

However, unlike the gradient topology, DLM uses relatively simple heuristics for the estimation of global system properties. For example, the average number of clients per super-peer is estimated in DLM as the current number of clients of one super-peer. At the same time, dedicated algorithms exist, such as aggregation algorithms described later in this thesis, that have been specifically designed to efficiently approximate global system properties. These algorithms have been shown to achieve good scalability and performance, with average approximation error decreasing exponentially with time.

The theoretical model for P2P systems proposed in [196] is specific to file-sharing applications and cannot be easily applied to other areas. Furthermore, it requires the knowledge of the average duration of peer connections, which may depend on the system deployment environment, and hence, can only be obtained at runtime. Moreover, the description of DLM in [196] is missing details, which makes the algorithm very difficult to analyse and implement. In particular, [196] does not explain how the threshold parameters, $Z_{c,p}$ and $Z_{a,p}$, and scale parameters, $X_{c,p}$ and $X_{a,p}$, are calculated.

2.6 Gradient Topology Approaches

This short section describes P2P systems that adapt their structure to available peer resources, but do not elect super-peers. The purpose of this section is to complete the state-o-the-art systems review in this chapter by presenting alternative approaches to dealing with P2P system heterogeneity. Some of the concepts introduced in the systems covered in this section are similar to the gradient topology.

2.6.1 Astrolabe

Astrolabe [151] is a large-scale information management system that continuously monitors the dynamically changing state of a collection of distributed resources, reporting summaries of this information to its users. Astrolabe computes these summaries using on-the-fly aggregation controlled by SQL queries.

Astrolabe has a hierarchical structure that can be viewed as a tree. Each node in this tree represents a zone. A zone is "recursively defined to be either a host or a set of non-overlapping [i.e. not having any hosts in common] zones" [151]. The leaves in the tree represent individual hosts. For each zone, Astrolabe computes aggregates of information from all hosts that belong to this zone (i.e. host being descendants of this zone). These aggregates are computed by nodes continuously gossipping with each other within their administrative zones. Additionally, each zone has representative agents that gossip with representatives agents of other zones on behalf of these zones. This way, the information is summarised and gradually propagated from tree leaves to the root node, which receives the system-wide aggregates of all hosts.

Astrolabe is similar to the gradient topology in that it has a hierarchical structure. The representative nodes in each zone are elected using the same gossipping mechanism that produces data aggregations. Thus, Astrolabe can exploit the most stable and best performing nodes for representing zones at higher hierarchy levels. However, Astrolabe does not provide any specific mechanisms that allows nodes to actively manage and optimise the zone structure. Zones in Astrolabe are implicitly defined by node names. Each node that enters the system autonomously chooses its own name and joins the corresponding zones, creating new zones if necessary. For example, a node identified */USA/Cornell/pc3* belongs to the root zone */*, the */USA* zone, and the */USA/Cornell* subzone withing */USA*. Thus, the zone tree grows spontaneously.

2.6.2 Gia

An interesting approach to address P2P system heterogeneity is proposed in Gia [34, 110]. Gia extends the original Gnutella protocol in order increase its scalability. It is based on four main principles. First, it introduces a dynamic topology adjustment mechanism that increases the degree of high-capacity nodes. Node capacity is calculated based on node properties such as "processing power, disk latency, and access bandwidth" [34]. Each node occasionally compares its capacity with that of its neighbours, and computes its *level of satisfaction*. The satisfaction level is low if the total capacity of node's neighbours, normalised by their degree, is lower than the node's own capacity. In such a case, the node runs a topology adjustment algorithm, where it discovers and connects a new neighbour.

The second main principle in Gia is one-hop replication. All nodes in Gia maintain pointers to the content stored by their immediate neighbours. This mechanism, together with the topology adjustment algorithm, allows high-capacity nodes to act as hubs, which can resolve queries on behalf of other nodes. Third, Gia replaces the flooding-based search in Gnutella with a biased random walk, which directs queries to high-capacity nodes and allows the utilisation of hubs. Finally, Gia uses an active flow control algorithm that balances the load in the overlay by probabilistically routing queries towards nodes with more available capacity. By avoiding hot-spots (i.e., overloaded peers), Gia can significantly reduce query latency and drop rate.

The similarity between Gia and the gradient topology is that higher-capacity nodes are promoted in the system structure such that they handle more system traffic and load. However, Gia is specifically designed for keyword query processing and its design principles cannot be easily ported beyond the file-sharing domain.

2.6.3 Virtual Nodes

A powerful mechanism to exploit high-capacity nodes in a P2P system is proposed in Chord [179]. In this system, a single physical host can run multiple *virtual nodes*, i.e., instances of the P2P protocol, in order to utilise its available capacity. By creating, migrating, and removing virtual nodes, the system can balance the load between physical hosts and effectively use available resources. A number of virtual node allocation algorithms, based on Distributed Hash Tables, are proposed in [148, 88].

The main advantages of virtual nodes are their great simplicity and very high applicability. Virtual nodes can be used in a straight-forward way in many different P2P applications. However, virtual nodes also have a number of disadvantages. First, they usually increase the system overhead. For example, in a DHT overlay of size N, a node typically maintains $O(\log N)$ neighbours. When a host runs k virtual nodes, it needs to maintain $O(k \log N)$ DHT neighbours, and hence, must generate more background traffic to detect neighbour failures and to keep its routing tables up-to-date. Generally, virtual nodes increase the number of nodes in a P2P protocol, while super-peers can be used to reduce the number of participants in a P2P protocol. If the protocol does not scale well, generating virtual nodes may become nonviable. Similarly, if node stability is concerned, virtual nodes are of limited use, since they can only increase the number of stable nodes in the system but do not exclude the non-stable nodes from the P2P protocol.

2.7 Summary

This chapter surveys the area of heterogeneous P2P systems. These systems are based on super-peers in the large majority of cases, but examples are also given for systems that are based on other design principles. The covered domains include storage systems (OceanStore, Brocade), file-sharing systems (Gnutella, KaZaA, eDonkey), telephony and video-conferencing systems (Skype), e-learning systems (Edutella, ROSA), Grid systems (GLARE), and distributed hash tables (HONets, SPChord, Structured Superpeers, and others). A large part of the reviewed systems can be classified as general P2P frameworks and algorithms.

The functions assigned to super-peers, as well as the overlays run by the super-peers, are generally application-specific. File-sharing, e-learning, and Grid systems use super-peers for indexing files (and other resources) and handling search protocols. Grid systems also use super-peers for managing membership information. Skype uses super-peers for relaying traffic between firewalled peers. OceanStore introduces super-peers for coordinating updates on replicated objects. Brocade, and many other systems based on DHTs, use super-peers for routing messages.

Most reviewed systems attempt to elect super-peers that have certain desired characteristics. These characteristics are often described as high stability, high processing capability, large available storage space, and a high-quality network connection. Peer stability is usually estimated using peer's current uptime. The processing capability is often defined as a function of peer's CPU clock speed and amount of RAM. The quality of a peer's network connection is typically measured using the upstream or downstream bandwidth of the peer's Internet link. Furthermore, some systems, such as Gnutella, require that super-peers have non-firewalled Internet connections and run a certain version of the operating system.

The reviewed systems vary in the requirements for the desired number of super-peers in the network. In PoPCorn and SOLE, the goal is to elect a fixed number of super-peers, given as a system parameter. In more adaptive systems, such as SG-1, SG-2, DLM, HONets and SPChord, the number of super-peers is regulated according to the current system size and load. In particular, DLM maintains a fixed ratio of super-peers to clients, and both SG-1 and SG-2 elect super-peer sets that have sufficient capacity to handle the remaining peers as clients. In some systems, such as Crown, PASS, eDonkey and Grids, the numbers of super-peers is not strictly controlled and depends on external factors such as peer IP addresses (Crown), peer locations (PASS), and local users or administrators (eDonkey, Grids). Furthermore, some systems introduce additional constraints on super-peers and clients, such as a maximum distance in a virtual coordinate system or semantic space.

The super-peer election methods in the reviewed systems can be divided into four general categories. The first category, comprising the simplest approaches, includes systems where super-peers are hardcoded, configured manually, or elected based on fixed thresholds. Centralised approaches are not scalable and introduce security and reliability risks. Manual or static selection is not likely to produce optimal super-peer sets due to the complexity and dynamism in most P2P systems. Fixed thresholds can only be applied to systems where the distribution of system-wide peer characteristics does not change significantly in time and is known to the system designer or administrator.

In systems that belong to the second category, peers are divided into groups based on properties such as physical location, position in a virtual space, semantic content, or membership in a Virtual Organisation. This approach has the advantage that the super-peer election problem can be decomposed into local election subproblems which are solved independently in each group. It also allows the system to bind clients with super-peers that are close to them according to a system metric. However, this approach introduces the problem of group management. Simple schemes, based on peer properties such as IP address or ZIP code, do not allow peers to actively control the number of super-peers in the system. Furthermore, election in groups does not guarantee that all peers with globally-highest capability become super-peers, and does not guarantee that clients are evenly distributed between super-peers.

The third category consists of systems that elect super-peers using a DHT overlay. In these systems, super-peer clusters can be dynamically split and merged, and the number of super-peers can be regulated based on the current network size and load. However, due to the constraints imposed by the DHT, these systems cannot guarantee that the elected super-peer sets are optimal in terms of size and super-peer capabilities.

The last category contains systems that continuously optimise super-peer sets according to a formally defined criterion. SG-1 generates a topology with minimum number of super-peers such that the total super-peer capacity is equal to the total number of clients. SG-2 extends SG-1 and introduces additional constraints on the maximum distance between super-peers and clients. Finally, DLM derives an optimal ratio of super-peers to clients from a file-sharing workload model, and provides a mechanism for maintaining such an optimal ratio in a P2P system, electing high-uptime and high-capacity super-peers.

In summary, the systems reviewed in this chapter clearly show that there is a general need for introducing super-peers in P2P applications in order to improve their overall performance and scalability. However, the majority of existing systems offer simple and limited mechanisms for the super-peer election, and only a handful of systems attempt to optimise super-peer sets according to well-define criteria. These few sophisticated systems are specific to particular application scenarios.

The remaining chapters in this thesis show that gradient topologies, in combination with aggregation-based election techniques, extend the current state-of-the-art knowledge on super-peers, and allow more flexible and adaptive super-peer election in large-scale heterogeneous systems. In particular, gradient topologies can generate super-peer sets equivalent to that in SG-1, DLM, and many other reviewed systems, and can adapt super-peer sets according to the requirements in the system.

Chapter 3

Gradient Topologies

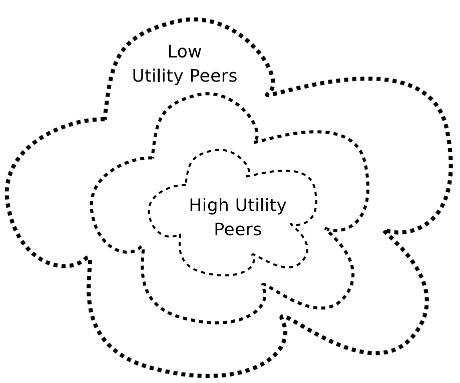
This chapter formally defines gradient topologies (GT) and describes their main properties. It then introduces a subset of gradient topologies, called tree-based gradient topologies (TGT), which have a number of attractive, formally proven properties, such as a diameter growing logarithmically with the overlay size. Due to these properties, the thesis mainly focuses on the TGTs, although many features are common to both GTs and TGTs. The last section presents a brief design of two large-scale applications, a P2P storage systems and a P2P name service, that take advantages of the TGTs. The purpose of this last section is to show, at a high-level of detail, how the TGTs can be used in practical application scenarios. The algorithms that generate TGTs and elect super-peers are presented in the next chapter.

3.1 Topology Properties

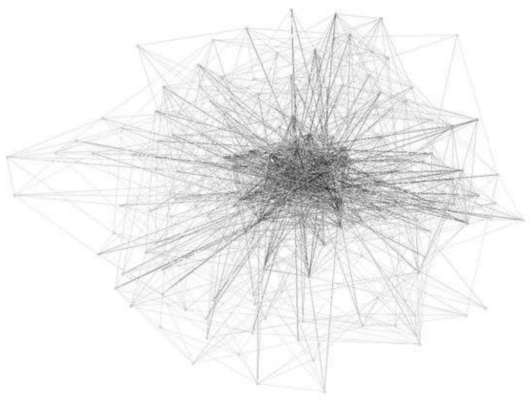
Gradient Topologies (GT) are a class of P2P overlay topologies, where peers are arranged according to their utility such that the highest utility peers are clustered in the logical centre of the topology (also called *core*) while lower utility peers are located at gradually increasing distance. The higher the utility of a peer, the closer this peer is, in terms of overlay hops, to the maximum utility peers in the system.

Peer utility metric is defined by the application that runs on top of the gradient P2P topology. It is assumed that the higher-level application requires the selection of the highest-utility peers in the network for its application-specific purposes. For example, in a content distribution network, the utility may be defined as a function of a peer's maximum upstream bandwidth. In a P2P storage system, the utility may combine a peer's available storage space and bandwidth, while in a grid computing system, the utility may be a function of a peer's processing speed and expected availability.

The gradient topology is independent from the higher-level application in the sense that all al-



(a) Conceptual diagram



(b) Visualisation obtained from a P2P simulator

Figure 3.1: Gradient topologies.

gorithms used for the construction of the topology, message routing, and election of super-peers, do not make any assumptions about peer utility. They only require that for every peer p in the system, a utility value, $U(p)$, is defined. Thus, the utility metric encapsulates application-specific peer requirements.

Formally, gradient topologies can be defined as P2P topologies where for any two peers, p and q, if $U(p) \geq U(q)$ than $\text{dist}(p, p_0) \leq \text{dist}(q, p_0)$, where $\text{dist}(x, y)$ is a peer distance metric defined as the shortest path length between two peers x and y, and p_0 is the highest utility peer in the system. Figure 3.1(a) shows a conceptual diagram of a gradient topology, and 3.1(b) shows a sample visualisation of a gradient topology generated in a P2P simulator [163]. Darker nodes and darker edges indicate higher-utility peers and connections between high-utility peers.

Gradient topologies have two main properties. First, all peers in a gradient topology with utility above a given *utility threshold* form a connected sub-overlay, which is itself a gradient topology and is concentric with the total system topology. Such high-utility peers can be exploited in a similar manner as super-peers in traditional P2P systems.

Second, the information captured in the gradient topology enables efficient routing of messages from low-utility peers to high-utility peers. This is achieved by forwarding messages at each peer to the highest-utility neighbour, as in hill-climbing and similar search heuristics. This strategy, called *gradient search*, guarantees that messages are eventually delivered to the highest-utility peers in the system.

Assuming that a higher-level application uses the highest-utility peers in the system for running certain services, gradient search allows lower-utility peers to discover these high-utility peers in order to access the services hosted by them. Gradient search does not require any global knowledge at peers, as it requires only that each peer estimates the utility of its immediate neighbours. Gradient search is also deterministic, and it does not require peers to duplicate message, as in flooding and parallel random walks.

3.1.1 Utility Thresholds

Given that peers are characterised by a common utility metric U, the super-peer election problem in a gradient topology can be solved by calculating a super-peer utility threshold. All peers with utility above the selected threshold become super-peers, and the remaining peers become clients. The super-peer set elected this way is optimal in the sense that the utility of peers in the set is maximised. Each super-peer has a higher utility than any client.

The number of elected super-peers is directly controlled by the super-peer utility threshold. By decreasing the threshold, the system can increase the number of super-peers, and by increasing the threshold, the system can decrease the number of super-peers. Due to the structure of the gradient topology, no peer connections need to be reconfigured as super-peers are added or removed, since super-peers always are clustered at the centre of the topology and can be discovered by clients using gradient search.

A number of different criteria can be applied when calculating super-peer thresholds. In the simplest case, the threshold can be explicitly given by a higher-level application. However, as discussed in section 2.2.14, setting a threshold that limits the number of super-peers to a desired level requires a global knowledge of peer utility. Furthermore, if the super-peer election threshold is fixed, the system is not able to adapt the number of super-peers to the existing demand and is likely to generate a suboptimal super-peer set if the characteristics of peers in the system significantly change over time.

3.1.1.1 Top-K Threshold

A *top-K threshold* is defined as a utility value, t_K, such that exactly K peers in the system have utility equal or above t_K and all remaining peers have utility below t_K. Given the cumulative peer utility distribution in the system, $D : \mathbb{R} \to \mathbb{R}$, where $D(u)$ is the number of peers with utility above u,

$$D(u) = \Big|\{p : U(p) \geq u\}\Big| \tag{3.1}$$

the top-K threshold must satisfy the following equation

$$D(t_K) = K. \tag{3.2}$$

Assuming that peers have a knowledge of the utility distribution D, a top-K threshold allows a precise restriction of the number of super-peers in a dynamic system. It has the property that, regardless of the system size (as long as $N \geq K$) and utility of participating peers, it elects exactly K super-peers, and the utility of these super-peers is maximised.

3.1.1.2 Proportional Threshold

Similarly, a *proportional threshold* is defined as a utility value, t_Q, such that a fixed fraction Q of peers in the system have utility greater than or equal to t_Q. In a system with N peers, a proportional threshold is described by the following equation

$$D(t_Q) = Q \cdot N. \tag{3.3}$$

A proportional threshold allows peers to adapt the number of super-peers to the total system size. As the system grows and shrinks in size, the proportional threshold increases and decreases, adjusting the number of super-peers in the system so that the ratio of super-peers to ordinary peers remains constant.

3.1.1.3 Capacity Threshold

In many applications, the desired number of super-peers depends not only on the system size but also on the capabilities of available peers. For example, systems such as SG-1, SG-2 and DLM introduce the notion of peer *capacity* and generate super-peer sets that have sufficient capacity to handle all remaining peers as clients.

Assuming that a capacity value $C(p)$ is defined for each peer p, a *fixed-capacity threshold* can be introduced as a utility value, t_C, such that peers with utility above t_C have a total capacity of C. A fixed-capacity threshold can be calculated using a cumulative peer capacity distribution, $D^c : \mathbb{R} \rightarrow \mathbb{R}$, where $D^c(u)$ is the total capacity of peers with utility above u,

$$D^c(u) = \sum_{U(p)>u} C(p). \tag{3.4}$$

The threshold must satisfy the following formula, similar to the top-K threshold definition

$$D^c(t_C) = C. \tag{3.5}$$

3.1.1.4 Clients Threshold

In order to elect super-peers that have a total capacity equal to the number of clients in the system, as in SG-1 and SG-2, a *clients threshold* is defined as a utility value, t, such that

$$D^c(t) = N - D(t). \tag{3.6}$$

In the above equation, $D(t)$ is the number of elected super-peers, $N - D(t)$ is the number of clients, and $D^c(t)$ is the total super-peer capacity. If peer utility is defined as peer capacity, i.e., $U(p) = C(p)$ for every peer p, the super-peer set generated using a clients threshold is equivalent to that in the SG-1 target topology. If $U(p) \neq C(p)$, super-peers are selected from the highest-utility peers in the system, and the number of super-peers is determined by the system size and super-peer capacity.

3.1.1.5 Load-Based Threshold

A more general approach to elect super-peers is based on the concepts of peer capacity and peer *load*. Depending on the higher-level application, load can represent connected clients, stored data, network transfers, handled requests, running jobs, or other application-specific concepts. The goal of the super-peer election is to generate a super-peer set that has a sufficient total capacity to handle the load generated in the system.

More formally, the capacity $C(p)$ of a peer p is defined as the maximum load peer p can handle at a time, and $L(p)$ represents load that peer p is currently handling. Any peer can generate load, and the total amount of system load fluctuates over time. A *load-based threshold* is defined as a utility value, t, such that peers with utility above t have a total capacity equal to the total system load, i.e.,

$$D^c(t) = \sum_p L(p). \tag{3.7}$$

The utilisation of peer p is defined as the ratio of a peer's current load to the peer's capacity, i.e., $\frac{L(p)}{C(p)}$. If the super-peer threshold is calculated using formula 3.7, all super-peers must achieve a full utilisation (i.e., equal to one) in order to handle the total system load. This requirement can be relaxed by allowing super-peers to have a higher total capacity than the system load. In order to maintain an average super-peer utilisation of W, where $0 < W \leq 1$, the super-peer election threshold, t_W, must satisfy the following formula

$$D^c(t_W) \cdot W = \sum_p L(p). \tag{3.8}$$

This way, super-peers maintain a margin of spare capacity and can accommodate extra load in case of a rapid load level increase. Moreover, when the average super-peer utilisation is reduced, more super-peers in the system have free capacity, and the distribution of load between super-peers that have free capacity becomes easier.

3.1.1.6 Composite Threshold

Finally, multiple criteria for super-peer sets can be combined into one threshold. For example, if the super-peer set must satisfy two conditions, to have a minimum size of K, and a minimum capacity of C, a composite threshold, $t_{K,C}$, is defined as $\min(t_K, t_C)$, where t_K and t_C are derived from formulas 3.2 and 3.5, respectively. Similarly, in order to elect a super-peer set that has a size of K *or* capacity of C, the super-peer election threshold is set to $\max(t_K, t_C)$.

3.1.2 Super-Peer Utility

Many measurements on existing P2P systems show that peer characteristics, such as session times, availability, and bandwidth, are closely approximated by Pareto distributions [173, 144, 181], and many theoretical models of P2P systems assume that peer properties follow Pareto distributions, also called power-law [135, 4, 14, 7]. Typically, the Pareto shape parameter, k, in P2P systems is approximately equal to two [181]. Given a system with Pareto distributed peer utility, the following theorem describes the utility of super-peers.

Theorem 3.1 *In a system where peer utility follows a Pareto distribution with exponent k, and the highest-utility peers are elected super-peers and constitute a fraction Q of all peers in the system, the ratio of mean peer utility to mean super-peer utility is $\sqrt[k]{Q}$.*

Proof Given that peer utility follows a Pareto distribution, the probability that a peer p has a utility value above x is

$$P(U(p) > x) = \left(\frac{m}{x}\right)^k \tag{3.9}$$

where m is the minimum peer utility and k is a constant system parameter, $k > 0$. In order to maintain a ratio Q of super-peers to the total number of peers, the super-peer utility threshold, t, must satisfy

$$\left(\frac{m}{t}\right)^k = Q. \tag{3.10}$$

Hence, $t = m \sqrt[-k]{Q}$. The utility of super-peers also follows a Pareto distribution, but the minimum super-peer utility is t. The probability that a super-peer s has a utility value above x is then given by

$$P(U(s) > x) = \left(\frac{t}{x}\right)^k. \tag{3.11}$$

The mean of the peer utility is $\mu = \frac{km}{k-1}$. The mean of the super-peer utility is

$$\mu' = \frac{kt}{k-1} = \frac{km \sqrt[-k]{Q}}{k-1} = \mu \sqrt[-k]{Q} \tag{3.12}$$

and the ratio between mean peer utility, μ, and mean super-peer utility, μ', is then $\frac{\mu}{\mu'} = \sqrt[k]{Q}$. □

3.2 Tree-Based Gradient Topologies

This section introduces tree-based gradient topologies and describes their main properties, including average peer degree, diameter, path lengths, and average distance to a super-peer.

Given a utility metric U, peers can be ordered according to their utility, from the highest-utility peer p_0 to the lowest-utility peer p_{N-1}, where N is the total number of peers. The position of a peer p in such a ranking, denoted $R(p)$, is called *rank*. A tree-based gradient topology (TGT) is defined as a gradient topology such that each peer p (excluding p_0) is connected with a peer ranked $\lfloor\frac{R(p)}{B}\rfloor$, where B is a constant system parameter called *branching factor*, $B > 1$. Peer ranked $\lfloor\frac{R(p)}{B}\rfloor$ is called p's *parent*.

Figure 3.2 shows a sample TGT with 27 peers and branching factor $B = 3$. For clarity, peers have only these connections that are required by the TGT definition, i.e., to their parents. In most realistic

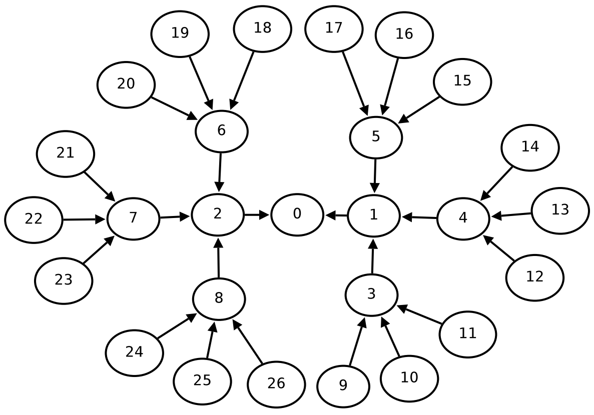


Figure 3.2: Sample tree-based gradient topology.

use cases, peers need to maintain additional links with each other in order to increase the system's fault-tolerance and reduce the probability of the topology partitioning.

3.2.1 Peer Degree

In a tree-based gradient topology, assuming no peer connections other than between a peer and its parent, peers ranked less than $\lfloor \frac{N}{B} \rfloor$ (with the exception of p_0) are connected with $B+1$ other peers: one parent and B children. Peers ranked above $\lfloor \frac{N}{B} \rfloor$ have no children and are connected with their parents only. The average peer degree is $\frac{2(N-1)}{N}$, since each peer (excluding p_0) has one outgoing connection to a parent, and the total number of incoming parent connections is equal to the total number of outgoing parent connections. Given that N is usually large in P2P systems, the average peer degree is close to two.

3.2.2 Topology Diameter

One of the main properties of TGT is that the shortest path between peer p and p_0 has at most $O(\log R(p))$ edges, where p_0 is the highest utility peer in the system. This fact can be shown by a straight-forward induction.

Theorem 3.2 *In a tree-based gradient topology, the shortest path between a peer, p, and the highest utility peer, p_0, where $p \neq p_0$, has at most $1 + \log_B R(p)$ edges.*

Proof Let $\mathrm{dist}(r)$ denote the shortest path length between peer p_r and p_0, i.e., $\mathrm{dist}(r) = \mathrm{dist}(p_r, p_0)$. The proof is by induction on the peer rank. The base case is for peer p_1, which is directly connected to p_0, and hence $\mathrm{dist}(1) = 1$.

Inductive step. Assume $\mathrm{dist}(i) \leq 1 + \log_B i$ for all i such that $i \leq r$. It needs to be shown that $\mathrm{dist}(r+1) \leq 1 + \log_B(r+1)$. Peer p_{r+1} is connected with its parent, ranked $\lfloor \frac{r+1}{B} \rfloor$, and hence

$\mathrm{dist}(r+1) \leq 1+\mathrm{dist}(\lfloor \frac{r+1}{B} \rfloor)$. Using the induction hypothesis, $1+\mathrm{dist}(\lfloor \frac{r+1}{B} \rfloor) \leq 1+1+\log_B \lfloor \frac{r+1}{B} \rfloor \leq 2+\log_B \frac{r+1}{B} = 1+\log_B (r+1)$. □

Theorem 3.2 immediately implies that the diameter of a TGT with N peers is at most $2+2\log_B N$, that is $O(\log N)$.

A single fact that a P2P topology has a short diameter does not necessarily indicate that such a topology is useful. For example, the diameter of purely random topologies grows logarithmically with the number of peers, but determining the shortest paths between peers in such topologies is expensive due to the lack of knowledge about the topology structure. Searching algorithms used in random P2P topologies, such as flooding, random walks, Breadth First Search (BFS), Depth First Search (DFS), and iterative deepening [109, 187, 199, 64], require sending messages to large numbers of peers, potentially all peers in the system.

Gradient topologies, in contrary to random and unstructured P2P topologies [31, 109, 42, 78], contain information about peer utility and enable efficient routing from low-utility peers to high-utility peers using gradient search. It can be shown that in a TGT, a message from peer p is routed by gradient search to peer p_0 through at most $\log_B R(p)$ intermediate peers. Thus, the worst-case cost of gradient search is $O(\log N)$ message transmissions. This fact can be shown by a simple induction on the peer rank, almost identically to Proof 3.2.

3.2.3 Distance to Super-Peer

Super-peers are elected in a TGT using utility thresholds, described in section 3.1.1, exactly in the same way as in GTs. According to Theorem 3.2, the K highest utility peers in a TGT form a TGT sub-overlay with a diameter of $O(\log K)$. Moreover, the topology structure imposes bounds on the maximum distance between a super-peer and a client.

Theorem 3.3 *In a tree-based gradient topology, where the K highest utility peers are super-peers, the shortest path between an ordinary peer, p, and a super-peer has at most $1 + \lfloor \log_B \frac{R(p)}{K} \rfloor$ edges.*

Proof Let $\mathrm{dist}'(r)$ denote the shortest path length from peer p_r to any super-peer. The proof is by induction on the peer rank. Obviously, for all peers p_i where $i < K$, $\mathrm{dist}'(r) = 0$. Furthermore, all peers p_i such that $K \leq i < KB$ are directly connected to super-peers through their parent links, and hence $\mathrm{dist}'(i) \leq 1$ for $i < KB$.

Inductive step. Assume $\mathrm{dist}'(i) \leq 1 + \lfloor \log_B \frac{i}{K} \rfloor$ for all i such that $K < i \leq r$. It needs to be shown that $\mathrm{dist}'(r+1) \leq 1 + \lfloor \log_B \frac{r+1}{K} \rfloor$. Peer p_{r+1} is connected with its parent, ranked $\lfloor \frac{r+1}{B} \rfloor$, and hence $\mathrm{dist}'(r+1) \leq 1 + \mathrm{dist}'(\lfloor \frac{r+1}{B} \rfloor)$. Using the induction hypothesis, $1 + \mathrm{dist}(\lfloor \frac{r+1}{B} \rfloor) \leq 1 + 1 + \lfloor \log_B (\lfloor \frac{r+1}{B} \rfloor / K) \rfloor \leq 2 + \lfloor \log_B \frac{r+1}{BK} \rfloor = 1 + \lfloor \log_B \frac{r+1}{K} \rfloor$. □

According to Theorem 3.3, every peer in a TGT is located within at most $O(\log_B \frac{N}{K})$ overlay hops from a super-peer, where N is the overlay size and K is the number of super-peers in the overlay. Given a super-peer ratio $Q = \frac{K}{N}$, the maximum distance from a client to the closest super-peer is at most $O(\log_B Q^{-1})$ overlay hops. Hence, if the branching factor, B, is set to the reciprocal of the super-peer ratio, Q, then every peer in the system is directly connected to a super-peer.

Theorem 3.4 *In a tree-based gradient topology where $B = Q^{-1}$, the maximum distance between a client and its closest super-peer is one.*

Proof Let N be the total number of peers in the system and K be the number of super-peers. The super-peer ration, Q, is equal to $\frac{K}{N}$. From Theorem 3.3, the maximum distance between a client and its closest super-peer is $1 + \lfloor \log_B \frac{N-1}{K} \rfloor$. This distance is equal to one if $\lfloor \log_B \frac{N-1}{K} \rfloor = 0$, which is equivalent to $0 \leq \log_B \frac{N-1}{K} < 1$, which is equivalent to $B > \frac{N-1}{K}$. This can be achieved by setting B equal to $\frac{N}{K} = Q^{-1}$. $\square$

An analogous reasoning to Proof 3.3 can be used to determine bounds on the performance of gradient search. It can be shown that in a TGT with a super-peer ratio of Q, gradient search routes a message from any peer in the system to a super-peer through at most $\log_B Q^{-1}$ intermediate peers, and the maximum cost of a super-peer discovery in a TGT is $1 + \log_B Q^{-1}$ message transmissions. This cost is equal to one if $B = Q^{-1}$.

3.3 Sample Applications

This section describes the design of two sample applications, a P2P storage system and a P2P name service, that take advantage of the properties of gradient topologies.

3.3.1 Storage System

The storage system is designed to permanently store user-provided data. It supports the following operations: (i) create an empty file, (ii) delete a file, (iii) read from a file, and (iv) write to a file. All operations (i-iv) take a file name as a parameter.

For performance and reliability reasons, the data stored by the system is hosted by super-peers only. Furthermore, the data is partitioned between super-peers using a DHT, such as Chord described in section 2.4.1. Each file name is mapped onto the DHT identifier space using a hash function, such as SHA-1, and assigned to a super-peer.

In order to create or delete a file, a peer p generates a request and routes it using gradient search to the closest super-peer, q. The contacted super-peer, q, forwards the request using the DHT protocol to the super-peer, s, that is responsible for the given file. Super-peer s then creates or deletes the file, as requested by p.

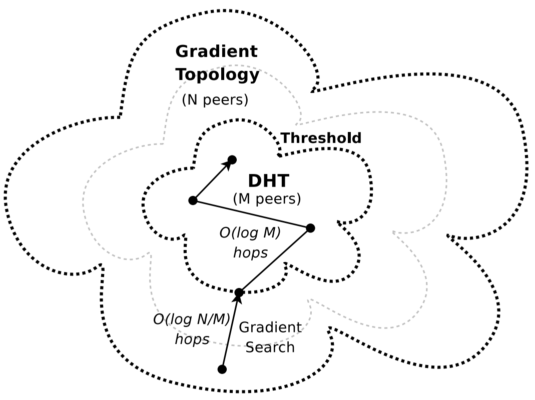


Figure 3.3: Storage system based on a gradient topology.

Similarly, in order to read or write a file, peer p performs a gradient search to discover a super-peer, q, which uses the DHT overlay to contact s, the super-peer responsible for the given file. Super-peer s directly contacts p and transfers the contents of the file. The design can be further extended to allow for file encryption and access verification, for example using public-key cryptography.

In a gradient topology with N peers and M super-peers, super-peer discovery requires at most $O(\log \frac{N}{M})$ message transmissions, as shown in Theorem 3.3. Furthermore, the DHT guarantees that the cost of a DHT lookup is at most $O(\log M)$ message transmissions. Thus, as shown in Figure 3.3, the total cost of locating a file is $O(\log \frac{N}{M}) + O(\log M) = O(\log N)$ message transmissions, as in a traditional DHT. This cost can be further reduced if clients cache super-peer addresses and reuse them when performing sequences of operations.

Super-peers are elected using a load-based utility threshold. Peer capacity is defined as the amount of storage space available at a peer, and the load at a peer is defined as the amount of data stored by the peer. The system maintains a set of super-peers that has a sufficient storage space to accommodate all data uploaded to the system. In order to distribute the data between the super-peers, one of the well-known approaches to load-balancing in a DHT is applied [88, 148, 76].

Furthermore, for improved data availability, each file can be replicated on r super-peers with numerically closest DHT identifiers to the hash of the file's name, as described in [179, 162, 52]. In this case, the total super-peer capacity must be r times higher than the amount of data uploaded to the system.

The choice of the peer utility function depends on the system requirements. In order to minimise the number of super-peers in the system, peer utility is defined as peer capacity. In order to maximise the throughput of read and write operations, peer utility is defined as a function of peer's downstream and upstream bandwidth capacity. According to theorem 3.1, if peer bandwidth follows a Pareto distribution with exponent k, and Q is the ratio of super-peers to the total number of peers, the average super-peer bandwidth is $\sqrt[-k]{Q}$ times higher than the average peer bandwidth. Thus, the expected file transfer rate, assuming no contention, is improved by a factor of $\sqrt[-k]{Q}$ compared with a traditional DHT where no super-peers are elected and all peers host data.

Another approach is to define peer utility as the expected peer session duration, which can be estimated using the history of previous peer sessions and current peer uptime [181]. If peer sessions follow a Pareto distribution with exponent k, than according to theorem 3.1, the average leave rate of super-peers is lower by a factor of $\sqrt[-k]{Q}$ compared with the average leave rate for all peers in the system. This has two advantages. First, the probability of data loss is lower compared with a traditional DHT, since peers hosting data are less likely to fail. Second, file replica maintenance cost is reduced, since peers join and leave the DHT less frequently (assuming the super-peer election threshold does not change) and hence, file replicas need to be transferred or re-created less frequently.

3.3.2 Registry Service

The P2P registry service stores a collection of domain-specific records, and allows peers to add new records, update existing records, delete records, and search for records that satisfy certain criteria. Each record can be updated or deleted only by its *owner*, which is the peer that created the record, but can be read by all peers in the system.

For fault-tolerance and performance reasons, the registry service is replicated between a limited number of high-utility super-peers, determined by an adaptive election threshold. Unlike the storage system described in the previous section, where the data is partitioned between super-peers, each super-peer in the P2P registry service maintains a full replica of the entire registry, i.e., has a copy of all records stored in the system. It is assumed that the average size of a record in the registry is relatively small (order of kilobytes), and hence, a single super-peer is likely to have enough storage space to host a full registry replica.

It is assumed that search operations are significantly more frequent than update operations, and hence, the registry is optimised for handling search. Due to the applied replication scheme, every super-peer can independently handle any search query without communicating with other super-peers. This is important, since complex search, for example based on attributes, keywords, or range queries, is known to be expensive in distributed systems [99, 141, 199, 109].

In order to perform a search on the registry, a peer generates a query and routes it to the closest super-peer using gradient search, as shown in 3.4. The super-peer processes the query and returns the search results directly to the originating peer. It can be shown that in a system with N peers and M super-peers, a query passes through at most $O(\log \frac{N}{M})$ peers before it is delivered to a super-peer. Optionally, if the super-peer is heavily-loaded, it may forward the query to another super-peer which has enough capacity to handle it. Clients may also cache super-peer addresses and contact the super-peers directly in order to reduce the routing overhead.

In order to create, delete, or update a record in the registry, a peer generates an update request and

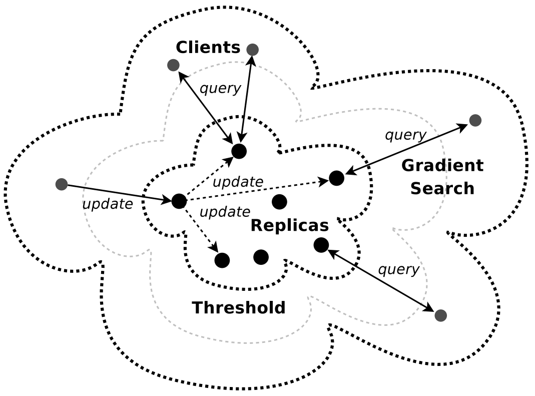


Figure 3.4: Registry service built upon a gradient topology.

routes it to the closest super-peer using gradient search. The update is then gradually disseminated to all super-peers using a probabilistic gossip protocol.

Every record in the registry is associated with a timestamp of the most recent update operation on this record. The timestamps are issued by the records' owners. Super-peers periodically gossip with each other and synchronise their registry replicas, as in [49]. Each super-peer periodically initiates a replica synchronisation with a randomly chosen super-peer neighbour, and exchanges with this neighbour all updates that it has received since the last time the two super-peers gossipped with each other.

Super-peers do not need to maintain a membership list of all replicas in the system. Due to the properties of the gradient topology, all super-peers are located within a connected component, and hence, every super-peer eventually receives every update. Conflicts between concurrent updates are resolved based on the update timestamps. Every record can be updated only by its owner, and it is assumed that the owner is responsible for assigning consistent timestamps for its own update operations.

Super-peers are elected using a load-based utility threshold. Each peer defines its capacity as the maximum number of queries it can handle at one time. The load at a peer is defined as the number of queries the peer is currently processing. The super-peer election threshold is calculated in such a way that the super-peers have sufficient capacity to handle all queries issued in the system. When the load in the system grows, new replicas are automatically created.

In order to reduce the probability of a super-peer departure or failure, peer utility is defined as the expected peer session duration. Hence, super-peers are elected amongst the most stable peers in the system. Furthermore, every peer, once elected a super-peer, never switches back to the role of a client, and maintains a registry replica until it permanently leaves the system.

Chapter 4

Design and Algorithms

This chapter describes a set of algorithms that allow a P2P system to generate and maintain a gradient topology, elect super-peers, and route messages from clients to super-peers. The chapter is organised as follows. The first section gives a high-level overview of all algorithms. The second section describes a number of peer utility metrics and shows how they can be computed by peers. The third section presents neighbour selection algorithms that generate GTs and TGTs. The fourth section describes aggregation algorithms that approximate global system properties. The fifth section covers super-peer election strategies for gradient topologies. The sixth section describes peer ranking algorithms. The seventh section addresses gradient search. Finally, the last section shows a bootstrapping mechanism for peers joining the system. The algorithms are evaluated in the next chapter.

4.1 Overview

Figure 4.1 shows a general overview of the algorithms introduced in this chapter, with dependencies between them indicated by arrows.

In a TGT, the neighbours of each peer are divided into three subsets: random, successors and tree, and each of these subsets is managed by a different neighbour selection algorithm. Random sets are used by an aggregation algorithm that estimates global system properties, such as the system size and peer utility distribution, and allows peers to calculate adaptive utility thresholds, which in turn are the basis for super-peer election. Moreover, the estimation of global system properties, together with successor sets, allows peers to estimate their ranks and generate tree sets. The topology structure created by the tree sets is exploited by gradient search, which routes messages from clients to super-peers. In addition, every peer has a black-box component that calculates peer's current utility. Since almost all algorithms rely on this component, for clarity, it is not drawn in Figure 4.1.

The construction of a GT is similar. The main difference is that GTs do not require tree neighbour

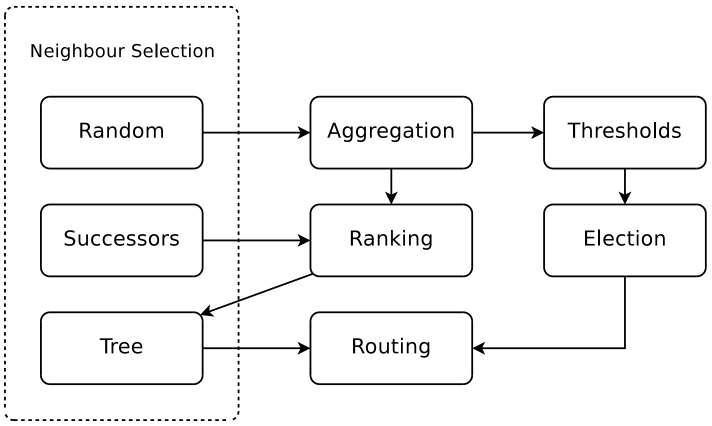


Figure 4.1: Overview on the main algorithms.

sets, and therefore, do not run the peer ranking algorithm.

4.2 Utility Metrics

The utility $U(p)$ of peer p is a number that reflects the appropriateness of peer p to act as a super-peer. The higher the utility, the more suitable a peer is to become a super-peer.

Peer utility is application-specific, and can be defined by the higher-level application in an arbitrary way. However, it can be expected that many P2P applications aim to elect super-peers that have the best possible hardware parameters, such as CPU clock speed, amount of RAM, and storage space. In such applications, peer utility can be defined as a function, such as a weighted sum or product, of these hardware parameter. This approach has the advantage that the computation of peer utility is simple and straight-forward. Each peer can obtain from the operating system, or measure directly, the values of its relevant parameters and independently compute its utility.

More sophisticated utility metrics may involve feedback from neighbouring peers. In particular, in untrusted environments, a decentralised approach to trust or reputation management [86, 138, 61] may be adopted in order to prevent malicious peers from providing fake utility information. However, trust-based approaches to utility computation are beyond the focus of this thesis.

4.2.1 Network Characteristics

Network characteristics, such as bandwidth, latency, and firewall status, are more challenging to estimate due to the decentralised and complex nature of wide-area networks. Moreover, many network properties, including bandwidth and latency, are properties of pairs of peers, i.e., connections between two peers, rather than individual peers.

Nevertheless, a peer can estimate the average latency and bandwidth of all its connections over time and use the average value as a general indication of its network connectivity and overall utility for the system. Furthermore, it has been shown that the bottleneck bandwidth of a connection between a peer and another machine on the Internet is often determined by the upstream bandwidth of the peer's direct link to the Internet [97, 167]. Thus, available bandwidth can be treated as a property of single peers.

The type and behaviour of the peer's firewall or Network Address Translator (NAT) can be determined using the STUN protocol [161, 160].

4.2.2 Peer Stability

For many applications, peer stability is amongst the most important peer characteristics, since in typical P2P systems the session times vary by orders of magnitude between peers, and only a relatively small fraction of peers stay on-line for a long time, as discussed in section 1.2.

One way of measuring the peer stability is to estimate the expected peer session duration. Stutzbach et al [181] show that the durations of consecutive peer sessions are highly correlated and the durations of previous peer sessions are good estimates for the duration of the current peer session. A number of sophisticated models, based on patterns in the history of peer sessions, have been proposed for predicting the behaviour of peers [106, 121].

In some applications, peer availability, defined as the fraction of time a peer is on-line, may be more adequate for expressing peer utility than the expected peer session length. As shown in a number of analyses, peer availability can be estimated based on the history of previous peer sessions [121, 172, 51, 18].

However, the information about previous peer sessions may not always be available, for example when peers are joining the system for the first time. In these cases, the remaining peer session duration, as well as the expected peer availability, can be estimated based on the peer's current uptime. Stutzbach et al [181] show that the peer's uptime is on average a good indicator of the remaining peer session time, although it exhibits high variance.

It can be shown that in systems where peer session times follow the power-law (i.e., Pareto distribution), the expected remaining session time of a peer is proportional to the current peer uptime. Similar properties can be derived for other session time distributions, such as the Weibull and log-normal distributions, which are often used in P2P system modelling.

Formally, if the peer session times in a system follow a Pareto distribution, the probability that a peer session duration, X, is greater than some value x is given by $P(X > x) = (\frac{m}{x})^k$, where m is the minimum session duration and k is a system constant such that $k > 1$. The expected peer session duration is $E(X) = \mu = \frac{k \cdot m}{k-1}$.

For peers with uptime of u, where $u > m$, session durations also follow a Pareto distribution, but with the minimum value of u, i.e., the probability that a peer's session is greater than x is equal to $P(X > x) = (\frac{u}{x})^k$. Hence, the expected session duration for peers with an uptime of u is $\frac{k \cdot u}{k-1}$. From this, the expected remaining session time is derived as $\frac{k \cdot u}{k-1} - u = \frac{u}{k-1}$.

4.2.3 Composite Utility

Multiple peer properties, such as hardware parameters, network characteristics, peer stability and availability, can be combined into one utility function. Given a set of M measurable peer properties, φ_1, φ_2, ..., φ_M, the simplest utility definition for a peer p is the product

$$U(p) = \varphi_1(p) \cdot \varphi_2(p) \cdot \ \ldots \ \cdot \varphi_m(p) \tag{4.1}$$

where all properties have an equal impact on the overall peer utility. In order to assign individual weights α_i for each peer parameter φ_i, peer utility is defined as a weighted sum

$$U(p) = \alpha_1\varphi_1(p) + \alpha_2\varphi_2(p) + \ \ldots \ + \alpha_m\varphi_m(p). \tag{4.2}$$

Naturally, any function of peer properties, depending on application-specific knowledge, can be used to define the peer utility metric.

4.2.4 Capacity and Load

Many P2P systems, such as SG-1, SG-2 and DLM, introduce the notion of peer capacity and load. Peer capacity, $C(p)$, usually refers to the total amount of resources, such as storage space, processing power, and bandwidth, available at peer p, while peer load, $L(p)$, usually represents the mount of resources that are currently being used.

There are two general approaches to defining peer utility based on capacity and load. One approach is to specify peer utility as a function of the peer's *available* capacity, i.e., $C(p) - L(p)$. This, however, has a significant drawback that peer utility changes over time, as it depends on the peer's current load. In particular, a peer decreases its utility when it is elected a super-peer and starts to receive load, and a peer increases its utility when it falls below the super-peer election threshold and stops receiving load. Such cyclic peer behaviour may destabilise the overlay and even prevent peers from generating a gradient topology. Moreover, depending on the application, frequent switches between ordinary peers and super-peers may introduce a significant overhead.

A better approach is to define the peer utility as a function of the *total* peer capacity, $C(p)$, and use the information about system load for the calculation of the super-peer election threshold. This way, peer utility, and hence the system topology, remain stable, while the super-peer set grows and shrinks as the total system load increases and decreases.

4.2.5 Dynamic Utility

If an external process consumes peer resources (i.e., storage space, processing power, bandwidth, etc.), the utility of peers *perceived* by the P2P application may change over time. Such utility changes are

```
1: **GetUtility():**
2: $t_{now} \leftarrow$ current time
3: **if** $t_{now} - t > t_{max}$ **then**
4:     $U \leftarrow$ measure peer utility
5:     $t \leftarrow t_{now}$
6: **end if**
7: **return** $U$
```

Figure 4.2: Lazy utility evaluation.

unpredictable from the P2P application's point of view, and hence, they force peers to occasionally re-compute their utility. In the simplest case, a peer p may calculate its utility every time $U(p)$ is requested by an algorithm running at p. However, this may introduce a significant overhead, especially if the evaluation of $U(p)$ requires measurements to be performed at peer p.

In order to reduce the overhead, a peer may cache the most recently calculated utility value, and re-evaluate its utility only if the cached value is older than a predefined parameter t_{max}. Figure 4.2 shows the pseudo-code of such a lazy utility evaluation strategy. Another approach is to calculate peer utility periodically.

Since peers are not able to measure or predict the utility of their neighbours, each peer p needs to maintain a cache that contains the most recent utility value, $U_p(q)$, for each neighbour q. Every entry $U_p(q)$ in the cache is associated with a timestamp created by q when the utility of q is calculated. Neighbouring peers exchange and merge their caches every time their neighbour selection algorithms exchange messages, preserving the most recent entries in the caches. Clocks do not need to be synchronised between peers since all utility values for a peer q are timestamped by q.

Highly variable peer utility is generally not desired, as it may impede peers' ability to create and maintain a gradient topology, and may cause frequent switches between super-peers and ordinary peers. In order to reduce utility fluctuations, a peer may calculate a moving average of its latest utility samples. Assuming a peer p periodically measures its utility and obtains utility samples

$$S_{p,t},\ \ S_{p,t-1},\ \ S_{p,t-2},\ \ S_{p,t-3},\ \ldots \tag{4.3}$$

the utility of peer p at time t can be defined as

$$U_{p,t}(p) = \frac{1}{T}(S_{p,t} + S_{p,t-1} + S_{p,t-2} + \cdots + S_{p,t-T+1}) \tag{4.4}$$

where T is a system parameter. This strategy, called simple moving average (SMA), requires peers to maintain a FIFO list with T most recent utility samples.

An alternative approach, called exponential moving average (EMA), does not require state to be maintained at peers and can be combined with both periodic and on-demand utility sampling. When a peer p obtains a new utility sample, $S_{p,t}$, it updates its current utility according to the following formula

$$U_{p,t}(p) = \alpha S_{p,t} + (1-\alpha)U_{p,t-1}(p) \tag{4.5}$$

where α is a constant weighing factor, $0 < \alpha < 1$. In an infinite system run, the utility calculated using EMA converges to a sum of all samples with exponentially decreasing weights

$$U_{p,t}(p) = \alpha S_{p,t} + \alpha(1-\alpha)S_{p,t-1} + \alpha(1-\alpha)^2 S_{p,t-2} + \alpha(1-\alpha)^3 S_{p,t-3} + \dots . \tag{4.6}$$

4.2.6 Utility Monotonicity

Assuming peer utility changes over time, two important utility properties are monotonicity and predictability. When peer utility grows or decreases monotonically, peers can cross the super-peer election threshold only once, assuming the threshold is constant. Moreover, if the utility changes are predictable, a peer is able to accurately estimate its own utility and the utility of its neighbours at any given time.

For example, if peer p defines its utility as the expected *session duration*, $Ses(p)$, and estimates it based on the history of its previous sessions, $U(p)$ is constant during one session. When p is elected a super-peer, it is not demoted to a client unless the super-peer election threshold increases above $U(p)$.

If the utility of p is defined as p's current *uptime*, $Up(p)$, the utility increases monotonically with time. Again, when p is elected a super-peer, it is not demoted unless the election threshold changes significantly. Furthermore, the utility function is fully predictable. Any peer q, at any time t, can compute the utility of p, given q has a knowledge of p's *birth time*, i.e., the time t_p when peer p entered the system, since

$$U(p) = t - t_p. \tag{4.7}$$

Clocks do not need to be synchronised between peers, and q can estimate the birth time of p using its own clock. At time t, when q receives the current uptime $Up(p)$ from p, it assumes that $t_p = t - Up(p)$.

If peer utility is defined as the expected *remaining session* time, i.e., $U(p) = Ses(p) - Up(p)$, the utility of a peer decreases monotonically over time. Peer q calculates the utility of p at time t as

$$U(p) = Ses(p) - (t - t_p). \tag{4.8}$$

Peer utility in this case may be negative, since $Ses(p)$ is only the expected session length and may differ from the actual session length. Furthermore, as peer utility decreases monotonically over time, every super-peer gradually drifts away from the gradient topology centre and may eventually be demoted.

4.2.7 Utility Uniqueness

Many algorithms described in this thesis assume that the utility value of each peer is unique, i.e., $U(p) \neq U(q)$ for any peers $p \neq q$. This property may not hold for some utility definition, particularly if peer utility is based on hardware parameters such as CPU clock speed, amount of RAM, etc. If the utility function is significantly coarse-grained, the ranking of peers based on utility and the construction of a gradient topology may become impossible.

Utility Metric	Constant	Monotonic	Predictable
Total Capacity	yes	constant	yes
Available Capacity	no	no	no
Session Length	yes	constant	yes
Remaining Session	no	decreasing	yes
Uptime	no	increasing	yes

Table 4.1: Utility metric properties.

In order to address this problem, each peer can add a relatively small random number to its utility value to break the symmetry with other peers. Thus, the utility of p is defined as

$$U_p(p) = U'_p(p) + \varepsilon_p \tag{4.9}$$

where $U'(p)$ is p's utility measured using one of methods described in sections 4.2.1 up to 4.2.6, and ε is pseudo-random variable initialised when p joins the system.

Table 4.1 summarises the main properties of the utility metrics described in this section.

4.3 Neighbour Selection

In a P2P system, a peer p has a knowledge of and communicates with a limited number of peers, i.e., its ***neighbours***, denoted $\mathcal{N}_p$. The addition and removal of neighbours at peer p is controlled by a neighbour selection algorithm. This section describes a number of neighbour selection algorithms that generate gradient topologies.

4.3.1 Connection Model

There are two general approaches to modelling neighbourhood in P2P systems. In one approach, the neighbourhood relation is asymmetric, i.e., if $q \in \mathcal{N}_p$ for two peers p and q, it is not required that $p \in \mathcal{N}_q$. Thus, the system topology is a directed graph. This model is relatively straight-forward to implement, as it only requires a peer to store contact addresses of each of its neighbour. However, it also has the drawback that peers may store stale addresses of neighbours that have already left the system. This is especially likely in the presence of heavy churn in the system. Such dangling references are disseminated between peers unless an additional mechanism is imposed that eliminates them from the system, such as timestamps used in the peer sampling service [78] and Newscast [80]. Moreover, in the asymmetric neighbourhood model, a peer has no knowledge about other peers that decide to add it to their neighbourhood sets, and hence, cannot control its in-degree in the topology structure.

In the second approach, the neighbourhood relation between peers is symmetric. If $p \in \mathcal{N}_q$, peers p and q are *connected,* and it is required that $q \in \mathcal{N}_p$. It should be noticed, however, that the symmetric model does not require any particular communication protocols. It only assumes that the neighbouring peers are able to contact each other. For example, in systems where messages are small and peers frequently change their neighbours (e.g., in gossip-based protocols), neighbouring peers may simply store the addresses of each other and may communicate over UDP. In applications where large amounts of data are exchanged between neighbouring peers and communication needs to be reliable, neighbouring peers may establish TCP connections. A peer can also create and close TCP connections using some algorithm, for example based on the Least Recently Used (LRU) strategy.

The main advantage of the symmetric connection model is that peers can notify each other when changing their neighbourhoods or leaving the system, which helps to keep the neighbourhood sets up to date. Furthermore, outdated neighbour entries are not propagated between peers in the system, as each peer verifies references received from other peers by contacting each new neighbour directly. Symmetric peer connections also reduce the risk of peer overloading. When a peer receives too many incoming connection requests, it may simply reject them. In the case of neighbours crashing, or leaving without notice, broken connections can be detected either by the operating system (e.g., when using the keep alive protocol for TCP connections) or through periodic polling of neighbours at the application level. The main drawback of the symmetric connection model is that every time two peers set up or close a connection, they need to exchange at least two messages.

In the remaining part of this thesis, it is assumed that peers maintain symmetric neighbour relations. Furthermore, it is assumed that any two peers are directly and mutually reachable. Firewall bypassing, using techniques such as STUN [161, 160], TURN [159], and ICE [158], is not addressed in this thesis.

4.3.2 Neighbour Subsets

The neighbours at each peer p are assigned into *neighbour subsets.* Each of these subsets is managed by a different neighbour selection algorithm, which adds and removes entries in it. A single neighbour q may belong to multiple subsets at peer p.

The neighbour selection algorithms are periodic. As shown in figure 4.3, peer p periodically invokes an update() procedure on each neighbour subset, which performs a cycle (also called time step) of a neighbour selection algorithm. If $\mathcal{N}_p$ is empty, peer p invokes a bootstrap procedure described in section 4.8.

Periodic neighbour selection algorithms generally perform better than reactive algorithms in heavy churn conditions, as they have a fixed invocation frequency and usually a bound communication cost. It has been observed that in systems with reactive neighbour exchange, peers generate bursts of messages in response to local failures, which congest local connections and results in a chain-reaction

```
1: loop
2:   if $N_p$ is empty then
3:     bootstrap
4:   else
5:     for each neighbour subset $S$ do
6:       update($S$)
7:     end for
8:   end if
9:   wait until next cycle
10: end loop
```

Figure 4.3: Neighbour selection framework.

from other peers that send more messages, potentially leading to a major overlay congestion and failure [153].

4.3.3 Connection Maintenance

The neighbourhood $\mathcal{N}_p$ at each peer p has a maximum size of $\mathcal{N}_p^*$. For each neighbour $q \in \mathcal{N}_p$, peer p maintains an estimation $U_p(q)$ of q's utility, as described in section 4.2. Additionally, for each neighbour q, peer p maintains an estimate $R_p(q)$ of q's rank. Peer ranks are estimated using an algorithm described in section 4.6, and are exchanged and timestamped by neighbouring peers in a similar way as utility values.

Moreover, for each neighbour $q \in \mathcal{N}_p$, peer p maintains a variable, $Ref_p(q)$, which counts the number of p's neighbour subsets that q belongs to. These counter variables serve two purposes. First, a neighbour q can be removed from $\mathcal{N}_p$ by a neighbour selection algorithm only if $Ref_p(q) = 0$, i.e., when q does not belong to any neighbour subset. By adhering to this rule, neighbour selection algorithms running at peer p do not interfere with each other when adding and removing neighbours. Second, a neighbour q can be removed from $\mathcal{N}_p$ only if $Ref_q(p) = 0$, that is when the connection between p and q is not used by the neighbour q. This way, peers achieve an agreement on which connections can be closed, and do not remove connections initiated by neighbours. Connection thrashing is harmful, as it increases the system overhead and may prevent peers from generating a desired topology.

When a neighbour selection algorithm at peer p decides to add peer q to a neighbour subset S, it follows the add procedure shown in Figure 4.4. In this procedure, peer p first checks if q belongs to the neighbourhood set $\mathcal{N}_p$ (line 2). If $q \notin \mathcal{N}_p$, peer p attempts to sends a connect request to q and waits for a response (lines 4-5). If none of the two peers reached the maximum number of neighbours, the peers agree to establish a connection and add their addresses to each other's neighbourhood sets (lines 7-8). Finally, peer p increments the $Ref_p(q)$ counter and adds q to S (lines 13-14).

In order to remove peer q from a neighbour subset S, peer p executes the remove procedure shown in Figure 4.4. First, peer p decrements $Ref_p(q)$ by one (line 17). If $Ref_p(q)$ is equal to zero, peer p concludes that q does not belong to any of its neighbour subset and hence is a candidate for removal

```
 1: Add($S$, $q$):
 2: if $q \notin \mathcal{N}_p$ then
 3:     if $|\mathcal{N}_p| < \mathcal{N}_p^*$ then
 4:         send connect request to $q$
 5:         receive response from $q$
 6:         if $|\mathcal{N}_q| < \mathcal{N}_q^*$ then
 7:             $\mathcal{N}_p \leftarrow \mathcal{N}_p \cup \{q\}$
 8:             $\mathcal{N}_q \leftarrow \mathcal{N}_q \cup \{p\}$
 9:         end if
10:     end if
11: end if
12: if $q \in \mathcal{N}_p$ then
13:     $S \leftarrow S \cup \{q\}$
14:     $Ref_p(q) \leftarrow Ref_p(q) + 1$
15: end if

16: Remove($S$, $q$):
17: $Ref_p(q) \leftarrow Ref_p(q) - 1$
18: if $Ref_p(q) = 0$ then
19:     send disconnect request to $q$
20:     receive response from $q$
21:     if $Ref_q(p) = 0$ then
22:         $\mathcal{N}_p \leftarrow \mathcal{N}_p \setminus \{q\}$
23:         $\mathcal{N}_q \leftarrow \mathcal{N}_q \setminus \{p\}$
24:     end if
25: end if

26: Leave():
27: for all neighbour $q$ in $\mathcal{N}_p$ do
28:     send disconnect message to $q$
29:     $\mathcal{N}_q \leftarrow \mathcal{N}_q \setminus \{p\}$
30:     for each neighbour subset $S$ at $q$ do
31:         $S \leftarrow S \setminus \{p\}$
32:     end for
33: end for

34: Verify():
35: for all neighbours $q \in \mathcal{N}_p$ do
36:     request $Ref_q(p)$ from $q$
37:     if $q$ is unresponsive then
38:         $\mathcal{N}_p \leftarrow \mathcal{N}_p \setminus \{q\}$
39:         for each neighbour subset $S$ at $q$ do
40:             $S \leftarrow S \setminus \{p\}$
41:         end for
42:     else
43:         if $p \notin \mathcal{N}_q$ then
44:             if $Ref_p(q) = 0$ then
45:                 $\mathcal{N}_p \leftarrow \mathcal{N}_p \setminus \{q\}$
46:             else
47:                 $\mathcal{N}_q \leftarrow \mathcal{N}_q \cup \{p\}$
48:             end if
49:         else if $Ref_p(q) = Ref_q(p) = 0$ then
50:             $\mathcal{N}_p \leftarrow \mathcal{N}_p \setminus \{q\}$
51:             $\mathcal{N}_q \leftarrow \mathcal{N}_q \setminus \{p\}$
52:         end if
53:     end if
54: end for
```

Figure 4.4: Neighbour connect, disconnect, leave and verify procedures.

(line 18). To that end, peer p sends a disconnect request to peer q (lines 19-20). If $Ref_q(p) = 0$, peer q agrees to close the connection (line 21), and both peers remove each other from their neighbourhood sets (lines 22-23). Otherwise, peer p preserves neighbour q in $\mathcal{N}_p$.

Thus, two peers need to exchange two messages with each other in order to connect or disconnect.

When peer p is leaving the system, it performs the leave procedure shown in Figure 4.4. In this procedure, it sends a disconnect message and unilaterally closes the connection to each neighbour q in $\mathcal{N}_p$, without waiting for a reply (lines 27-28). A neighbour q, when receiving a disconnect message from p, removes p from from its neighbourhood set $\mathcal{N}_q$ (line 29) and from all its neighbour subsets (lines 30-32). The purpose of the leave procedure is to help peers keep their neighbourhood sets up-to-date.

However, in an open P2P system, it cannot be assumed that all peers perform a leave procedure. A fraction of peers may silently crash, become unreachable, or leave without notifying their neighbours. Furthermore, due to message loss, the neighbourhood sets maintained at neighbouring peers may

```
1: Update($S_p$):
2: if $S_p$ is empty then
3:     $r \leftarrow$ random peer in $\mathcal{N}_p$
4:     add($S_p$, $r$)
5: end if
6: if $|S_p| \geq S_p^*$ then
7:     $r \leftarrow$ random peer in $S_p$
8:     remove($S_p$, $r$)
9:     remove($S_r$, $p$)
10: end if
11: if $|S_p| < S_p^*$ then
12:     $r \leftarrow$ random peer in $S_p$
13:     obtain random set $S_r$ from $r$
14:     $S_r' \leftarrow S_r \setminus S_p$
15:     $q \leftarrow$ random peer in $S_r'$
16:     add($S_p$, $q$)
17:     add($S_q$, $p$)
18: end if
```

Figure 4.5: Random set management algorithm.

become inconsistent. In particular, it may happen that $p \in \mathcal{N}_q$, but $q \notin \mathcal{N}_p$ for two peers p and q.

In order to remove stale entries from the neighbourhood set and resolve discrepancies with neighbours, each peer p periodically runs a neighbour verification algorithm shown in Figure 4.4. In this algorithm, peer p attempts to contact each neighbour $q \in \mathcal{N}_p$ (lines 35-36). If peer q is unreachable (line 37), peer p removes q from $\mathcal{N}_p$ (line 38) and all its neighbour subsets (lines 39-41). If q is responsive, but $p \notin \mathcal{N}_q$ (line 43), the discrepancy between peers is resolved based on $Ref_p(q)$ (line 44). If $Ref_p(q) = 0$, none of the peers are interested in maintaining the connection, and hence it is closed (lines 45). If $Ref_p(q) > 0$, peer q adds p to $\mathcal{N}_q$ in order to remove the inconsistency between the two peers (lines 46-47). Finally, in the case where $Ref_p(q) = Ref_q(p) = 0$ (line 49), the peers remove the connection between them (lines 50-51).

Furthermore, a neighbour is removed from $\mathcal{N}_p$ if a communication failure occurs when contacting this neighbour, for example, if no acknowledgement has been received within a certain period, or a certain number of message re-transmissions have failed, depending on the low-level communication protocol.

4.3.4 Random Set

A *random* neighbour subset contains a close-to-uniformly-random sample of all peers in the system. The main purpose of this set is to supply the aggregation algorithm with candidates for gossipping. As they are relatively easy to maintain and bootstrap, they can also be used by peers that have just joined the system for routing messages to higher utility neighbours, and as a simple means of system exploration. Furthermore, random sets practically prevent overlay partitioning, as the probability of a random graph partitioning is extremely low.

The neighbour selection algorithm that maintains a random set S_p at peer p is shown in Figure 4.5. The algorithm aims to keep approximately S_p^* entries in S_p. If the set is empty, it is initialised with a random neighbour from $\mathcal{N}_p$ (lines 2-5). If the size of S_p is equal to or higher than S_p^* (line 6), peer p removes a random peer, r, from S_p (line 7-8), and in order to maintain the symmetry, peer r removes p from its random set S_r (line 9). Finally, if the size of S_p is below S_p^* (line 11), peer p selects a random neighbour, r, from S_p (line 12) and obtains r's random set, S_r (line 13). Peer p then selects a random entry, q, from S_r, such that $q \notin S_p$ (lines 14-15), and both peers p and q add each other to their random sets (lines 16-17).

By removing excessive neighbours (i.e., when $|S_p| > S^*_p$) and adding new neighbours when $|S_p| < S^*_p$, the algorithm strives to maintain S^*_p neighbours in the random set. Moreover, as all entries inserted and removed from the set are selected at random, the algorithm generates pseudo-random samples of peers in the system.

The algorithm initiates at most one neighbour exchange with another peer per time step. Hence, on average, a peer participates in less than two neighbour exchanges per time step. In every exchange, a peer connects to at most one new neighbour. Hence, on average, a peer connects to less than two other peers per time step, and disconnects less than two neighbours per time step. Each of these operations, connecting, disconnecting, and exchanging neighbours, requires sending one message by the peer. Thus, the average cost of the neighbour selection algorithm can be bounded by 6 messages per time step.

A number of other gossip-based algorithms for the generation of pseudo-random topologies are known, including Cyclone [190], Newscast [80], and other approaches described in [78]. The main difference between these algorithms and the algorithm shown in Figure 4.5 is the connection model. The algorithm described in this thesis is based on a symmetric neighbourhood model, while all the other mentioned algorithms assume an asymmetric model.

RanSub [91] is another system that generates uniformly random neighbourhood sets. However, it is not based on randomised gossipping, but instead constructs a global overlay tree. For this reason, RanSub is less robust to churn and random failures compared to gossip-based approaches.

The algorithm presented here belongs to the $\langle$rand, pull, rand$\rangle$ class from the the taxonomy proposed in [78]. However, it does not converge to a star topology, as expected in [78], due to the symmetric neighbourhood model, which allows peers to eliminate excessive connections from other peers.

4.3.5 Preference-Based Set

In order to generate a topology that is more sophisticated than a random graph, peers must be able to select their neighbours according to individual criteria. In the preference-based neighbour selection algorithm, each peer p defines a *preference* function, which allows p to order any set of peers, S, from the most preferred neighbour, $\max(S)$, to the least preferred neighbour, $\min(S)$. The ordering is achieved by comparing pairs of peers using a preference operator $\underset{p}{>}$. If $q \underset{p}{>} r$ for two peers q and r, then p prefers q over r as its neighbour.

Figure 4.6 shows the pseudo-code of the preference-based neighbour selection algorithm. The algorithm is divided into two parts, which correspond to the left-hand side and right-hand side of

```
1:  **Update($S_p$):**
2:  loop
3:    if $\mathcal{N}_p = S_p$ then
4:      exit loop
5:    end if
6:    $m \leftarrow \max(\mathcal{N}_p \setminus S_p)$
7:    if $|S_p| < S_p^*$ then
8:      add($S_p, m$)
9:    else
10:     $n \leftarrow \min(S_p)$
11:     if $m \underset{p}{>} n$ then
12:       remove($S_p, n$)
13:       add($S_p, m$)
14:     else
15:       exit loop
16:     end if
17:   end if
18: end loop
19: $q \leftarrow$ random peer in $S_p$
20: obtain $\mathcal{N}_q$ from $q$
21: if $\mathcal{N}_q \nsubseteq \mathcal{N}_p$ then
22:   $\mathcal{N}'_q \leftarrow \mathcal{N}_q \setminus \mathcal{N}_p$
23:   $m \leftarrow \max(\mathcal{N}'_q)$
24:   if $|S_p| < S_p^*$ then
25:     add($S_p, m$)
26:   else
27:     $n \leftarrow \min(S_p)$
28:     if $m \underset{p}{>} n$ then
29:       remove($S_p, n$)
30:       add($S_p, m$)
31:     end if
32:   end if
33: end if
```

Figure 4.6: Preference-based neighbour selection algorithm.

Figure 4.6. In the first part, peer p updates its set S_p based on entries available in $\mathcal{N}_p$. This part of the algorithm does not require any communication with other peers, as it operates on peers that are already connected to p and belong to $\mathcal{N}_p$. These steps are necessary, as the neighbourhood $\mathcal{N}_p$ may have changed between the algorithm cycles due to churn, actions taken by other peers, and actions taken by other neighbour selection algorithms running at peer p. Furthermore, the preference function of peer p may have changed since the last algorithm cycle, for example due to the peer utility change.

If $S_p = \mathcal{N}_p$, nothing can be improved in S_p and the set update is finished (lines 3-5). Otherwise, peer p selects the most preferred neighbour, m, in $\mathcal{N}_p$ that does not belong to S_p (line 6). If the size of S_p is below S_p^*, peer p adds m to S_p (lines 7-8). If the size of S_p has already reached S_p^*, peer p selects the least preferred neighbour, n, in S_p (line 10) and compares it with m (line 11). If $m \underset{p}{>} n$, m and n are swapped in S_p (lines 12-13). If $m \underset{p}{\ngtr} n$, S_p cannot be improved with entries in $\mathcal{N}_p$ and the algorithm continues in line 19.

In the second part of the algorithm (lines 19-31), peer p attempts to obtain a new neighbour through gossipping with one of its current neighbours, as in the T-Man topology generator framework [77]. First, it chooses a gossip partner, q, that belongs to $\mathcal{N}_p$ (line 19). In the simplest case, q is selected randomly from S_p. Alternatively, q may be selected from S_p using a round-robin strategy. In principle, the choice of the gossip partner, q, enables a trade-off between the exploration of random connections in $\mathcal{N}_p$ and greedy exploitation of the knowledge about neighbours in $\mathcal{N}_p$. In some systems, the selection of $q = \max(\mathcal{N}_p)$ results in the fastest topology convergence to an optimal configuration, but randomised approaches are more robust in the general case.

When q has been chosen, peer p sends a gossip request to q and obtains the set of q's neighbours,

$\mathcal{N}_q$ (line 20). From this set, peer p selects the most preferred neighbour, m, that does not belong to $\mathcal{N}_p$ (lines 22-23). If such a peer exists, and the size of S_p is below S_p^*, peer p adds m to S_p (lines 24-25). If S_p has reached the maximum size, peer p selects the least desired neighbour, n, in S_p (lines 26-27) and compares it with m. If $m \underset{p}{>} n$, peer p removes n from S_p and replaces it with m.

In the presence of churn and communication failures, if either a request or response message is lost, a neighbour exchange fails. Thus, churn and message loss reduce the convergence speed of the system topology to the desired configuration, but do not introduce any side-effects such as a bias towards a particular non-desired topology.

The performance of the algorithm can be further improved by introducing "age bias" [79]. With this technique, a peer p does not initiate gossip exchange with low-uptime neighbours, because such neighbours have not had enough time to optimise their neighbourhood sets according to the preference function, and therefore are not likely to provide good neighbours for p.

As in the random set, on average, a peer gossips with two other peers per time step, connects at most two new neighbours, and disconnects at most two neighbours. The number of neighbours changed in the set per time step depends on the stability of the topology. A peer generally swap its neighbours more frequently upon joining the system, and less frequently when it has already found its optimal position in the topology. Neighbours are also more often swapped when the utility of peers changes dynamically. In all cases, the average number of messages sent by a peer per time step is between two and six.

4.3.5.1 Successor Set

A simple gradient topology can be constructed using utility *successor* sets. In these sets, peers aim to connect with neighbours that have higher but similar utility. Formally, a peer p prefers q over r for its successor set ($q \underset{p}{>} r$) if and only if

$$U_p(q) > U_p(p) \text{ and } U_p(r) < U_p(p) \tag{4.10}$$

or

$$|U_p(q) - U_p(p)| < |U_p(r) - U_p(p)| \tag{4.11}$$

for $U_p(q), U_p(r) > U_p(p)$ and $U_p(q), U_p(r) < U_p(p)$.

The advantage of the above preference function is that it does not require any knowledge of peer ranks and can be easily evaluated by every peer in the system. Gradient topologies generated using this preference function have been studied in [164, 163]. In principle, they do not guarantee the $O(\log N)$ gradient search cost and provide inferior support for message routing compared with TGTs.

4.3.5.2 Predecessor Set

Similar to successor sets, utility *predecessor* sets are generated by a preference function where $q \underset{p}{>} r$ if and only if

$$U_p(q) < U_p(p) \ \ and \ \ U_p(r) > U_p(p) \tag{4.12}$$

or

$$|U_p(q) - U_p(p)| < |U_p(r) - U_p(p)| \tag{4.13}$$

for $U_p(q), U_p(r) > U_p(p)$ and $U_p(q), U_p(r) < U_p(p)$.

Predecessor sets, when used together with successor sets, improve the topology convergence to a configuration where every peer is connected to the most similar peers in terms of utility [164, 163]. Again, such gradient topologies are outperformed by TGTs.

4.3.5.3 Tree Set

In order to generate a tree-based gradient topology, as defined in section 3.1, peers maintain *tree* sets. In these sets, each peer p aims to connect with neighbours that have ranks as close as possible to $\lfloor \frac{R_p(p)}{B} \rfloor$, where B is the topology branching factor and $R_p(p)$ is p's current estimation of its own rank.

Tree sets are generated by a preference function where $q \underset{p}{>} r$ for peers p, q and r if and only if

$$\left| R_p(q) - \frac{R_p(p)}{B} \right| < \left| R_p(r) - \frac{R_p(p)}{B} \right| . \tag{4.14}$$

In the simplest case, the size of the set is defined as $S^*_p = 1$, which produces a TGT where each peer has one parent. In order to improve the fault tolerance of the overlay, S^*_p is increased, so that each peer has more than one parent. However, this significantly increases the number of connections at high-utility peers, since in a TGT with N peers, where $S^*_p = M$ for each peer p, a peer with utility above $\frac{N}{B}$ has on average $(B + 1) \cdot M$ neighbours.

4.4 Aggregation

The aggregation algorithm, described in this section, allows peers to compute and periodically update approximations of global system properties, such as the minimum, maximum, and mean of peer utility, system size, and others. The approximations of these properties are used in turn for the calculation of peer ranks and super-peer election thresholds. The aggregation algorithm requires that a peer is able to obtain a close-to-random sample of all peers in the system. Such samples can be generated based on random neighbour subsets, described in 4.3.4, as well as the Newscast [80] and Cyclon [190] protocols.

```
1: loop
2:     $q \leftarrow$ random peer
3:     send $\mathcal{T}_p$ to $q$
4:     receive $\mathcal{T}_q$ from $q$
5:     $\mathcal{T}_p \leftarrow$ merge($\mathcal{T}_p, \mathcal{T}_q$)
6:     wait until next cycle
7: end loop
```

(a) Active thread

```
1: loop
2:     receive $\mathcal{T}_q$ from any peer $q$
3:     send $\mathcal{T}_p$ to $q$
4:     $\mathcal{T}_p \leftarrow$ merge($\mathcal{T}_p, \mathcal{T}_q$)
5: end loop
```

(b) Passive thread

Figure 4.7: Aggregation algorithm skeleton.

4.4.1 Framework

The aggregation algorithm is based on periodic gossipping. Each peer p maintains estimates (also called *aggregates*) $X_{p,0}$, $X_{p,1}$, $\ldots X_{p,n-1}$ of n global system properties. The true values of these properties are denoted X_0, X_1, $\ldots X_{n-1}$. Additionally, each peer stores a set $\mathcal{T}_p$ that contains the currently executing aggregation instances and auxiliary data. The aggregation algorithm can be seen as a meta-algorithm, since the selection of the system properties X_0, X_1, $\ldots X_{n-1}$ can be tailored to application-specific requirements.

Each peer runs an active and a passive thread. The active thread periodically initiates a gossip exchange and the passive thread responds to all gossip requests received from neighbours. Thus, on average, a peer sends and receives two aggregation messages per time step, i.e., protocol cycle. When initiating a gossip exchange, peer p randomly selects a neighbour q from its random neighbourhood set, and sends $\mathcal{T}_p$ to q. Peer q responds immediately by sending $\mathcal{T}_q$ to p. Upon receiving their sets, both peers merge them using a merge() operation described later. The general structure of the algorithm, shown in figure 4.7, is based on Jelasity's push-pull epidemic aggregation [81, 125].

The aggregation algorithm can be intuitively explained using the concept of aggregation *instances*. An aggregation instance is a computation that generates new approximations of the system properties X_0, X_1, $\ldots X_{n-1}$ at all peers in the overlay. Aggregation instances may overlap in time and each instance is associated with a unique identifier *id*. Potentially any peer can start a new aggregation instance by generating a new *id* and creating a new entry in $\mathcal{T}_p$. As the new entry is propagated throughout the system, other peers join the instance by creating corresponding entries with the same *id*. Thus, each entry stored by a peer corresponds to one aggregation instance that this peer is participating in. Every instance has a finite time-to-live, and when an instance ends, all entries corresponding to this instance are removed and all peers obtain new approximations of X_0, X_1, $\ldots X_{n-1}$.

Formally, each entry, T_p, in $\mathcal{T}_p$ of peer p is a tuple consisting of $n+2$ values,

$$(id,\ ttl,\ x_0,\ x_1,\ \ldots\ x_{n-1}) \tag{4.15}$$

where *id* is the unique aggregation instance identifier, *ttl* is the *time-to-live* for the instance, and x_0, x_1, $\dots x_{n-1}$ are variables that contain the current estimations of X_0, X_1, $\dots X_{n-1}$.

A peer p starts a new aggregation instance by creating a local tuple

$$(id,\ \ TTL,\ \ \mathrm{init}_0(p),\ \ \mathrm{init}_1(p),\ \ \dots\ \mathrm{init}_{n-1}(p)) \tag{4.16}$$

where *id* is chosen randomly, TTL is a system constant, and $\mathrm{init}_0(p)$, $\mathrm{init}_1(p)$, $\dots \mathrm{init}_{n-1}(p)$ are initial estimations of system properties X_0, X_1, $\dots X_{n-1}$ by p, as explained later.

As the initial tuple is disseminated by gossipping, peers join the new aggregation instance. It can be shown that in push-pull epidemic protocols, the dissemination speed is super-exponential, and with a very high probability, every peer in the system joins an aggregation instance in just a few time steps [80, 81].

The tuple merge procedure, $\mathrm{merge}(\mathcal{T}_p, \mathcal{T}_q)$, consists of the following steps. First, for each individual tuple $T_q = (id,\ ttl,\ x_0,\ x_1,\ \dots x_{n-1}) \in \mathcal{T}_q$ received by p from q, peer p checks if its local set $\mathcal{T}_p$ contains a tuple that is identified by *id*. If such a tuple is not found, and $ttl \geq \frac{TTL}{2}$, peer p creates a new tuple

$$(id,\ \ ttl,\ \ \mathrm{join}_0(p),\ \ \mathrm{join}_1(p),\ \ \dots\ \mathrm{join}_{n-1}(p)) \tag{4.17}$$

and adds it to $\mathcal{T}_p$. Values $\mathrm{join}_0(p)$, $\mathrm{join}_1(p)$, $\dots \mathrm{join}_{n-1}(p)$ depend on the approximated system properties X_0, X_1, $\dots X_{n-1}$ and are covered later.

By creating a local tuple, peer p joins a new aggregation instance *id* and introduces its own input to the approximation of X_0, X_1, $\dots X_n$. However, if $ttl < \frac{TTL}{2}$, peer p should not join the aggregation, as there is not enough time before the end of the aggregation instance to disseminate the information about p and to calculate accurate aggregates. This usually happens if p has just joined the P2P system and received an aggregation message that belongs to an already running aggregation instance. In this case, the update operation is aborted by p.

In the next step, for each tuple $T_q = (id,\ ttl_q,\ x_0,\ x_1,\ \dots, x_n) \in \mathcal{T}_q$ received from q, peer p replaces its own tuple, $T_p = (id,\ ttl_p,\ y_0,\ y_1,\ \dots,\ y_n)$, with

$$(id,\ \ \frac{ttl_p + ttl_q - 1}{2},\ \ \mathrm{merge}(x_0, y_0),\ \ \mathrm{merge}(x_1, y_1),\ \ \dots\ \mathrm{merge}(x_n, y_n)) \tag{4.18}$$

where again, functions $\mathrm{merge}_0()$, $\mathrm{merge}_1()$, $\dots \mathrm{merge}_{n-1}()$ are specific to the approximated system properties. By merging its local tuples with the tuples received from q, peer p contributes to the calculation of the aggregation result.

Finally, for each tuple $T_p = (id,\ ttl,\ x_0,\ x_1,\ \dots x_n)$ in $\mathcal{T}_p$ such that $ttl \leq 0$, peer p removes T_p from $\mathcal{T}_p$ and updates its current estimates of the system properties by setting $X_{p,i} = \mathrm{update}(x_i)$.

4.4.2 Aggregates

The definitions of $\mathrm{init}_i()$, $\mathrm{join}_i()$, $\mathrm{merge}_i()$ and $\mathrm{update}_i()$, for $0 \leq i < n$, depend on the approximated system properties X_0, X_1, $\dots X_n$. For example, in order to approximate the average peer utility in

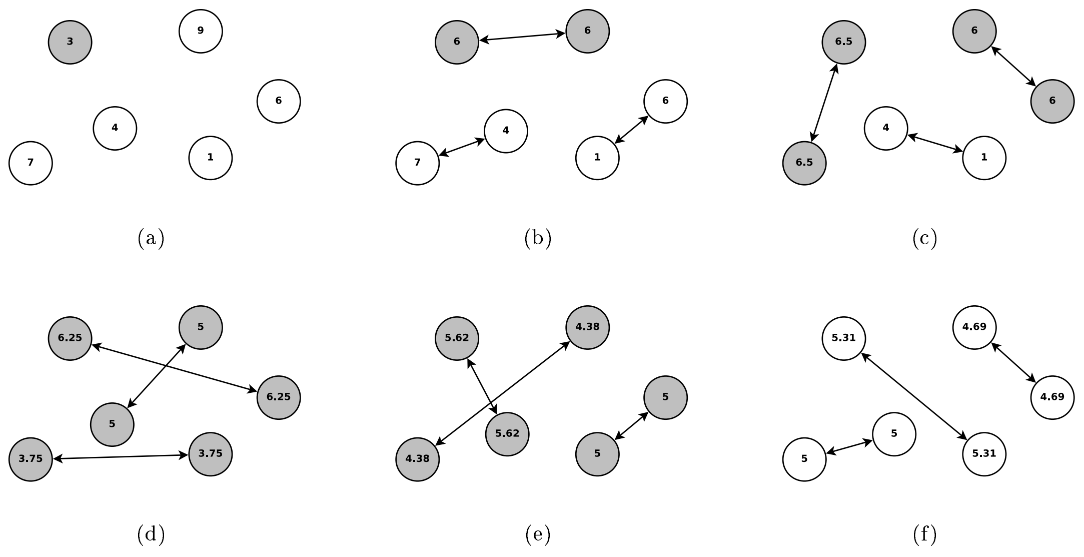


Figure 4.8: Approximation of the average through aggregation.

the system, X_i, $\text{init}_i(p)$ and $\text{join}_i(p)$ are defined as $U(p)$ for every peer p, $\text{merge}_i(x, y)$ is defined as $\frac{x+y}{2}$, and $\text{update}_i(x)$ is simply x. It can be shown that the values $X_{p,i}$ generated by the aggregation algorithm at each peer p approximate the the true utility average X_i with the average error and variance decreasing exponentially over time [82, 125, 81].

Figure 4.8 shows graphically a simple scenario where 6 peers estimate the average of their utility. The initial configuration is shown in picture (a). The peer in the top left corner initiates a new aggregation instance. As peers exchange tuples with each other, they gradually join the aggregation instance and update their estimations of the average (b-e). The variance between peers' estimates decreases over time. After 5 time steps, peers terminate the aggregation instance (f).

The estimation of the total number of peers in the system is based on the calculation of the average. The peer that starts the aggregation instance creates a tuple containing $init_i(p) = 1$. The remaining peers join the aggregation instance by adding tuples with $join_i(p) = 0$. Tuples are merged as in the calculation of the average. The values obtained at the end of the aggregation approximate the reciprocal of the system size, and hence, $\text{update}_i(x) = \frac{1}{x}$. Moreover, given the number of peers in the system, N, and the average peer utility, Avg, the sum of all peers' utility values is calculated as $Avg \cdot N$.

Figure 4.9 shows a sample aggregation execution where 6 peers estimate the system size N. Initially, one peer holds a value of one and the remaining peers are initialised with zeros (a). Over a few random gossip exchanges, peers average out their values and obtain close approximations of $\frac{1}{N}$ (b-e). After six time steps, four peers estimate the system size as $(\frac{5}{32})^{-1} = 6.4$, and two peers estimate the system size as $(\frac{3}{16})^{-1} = 5.33$ (f).

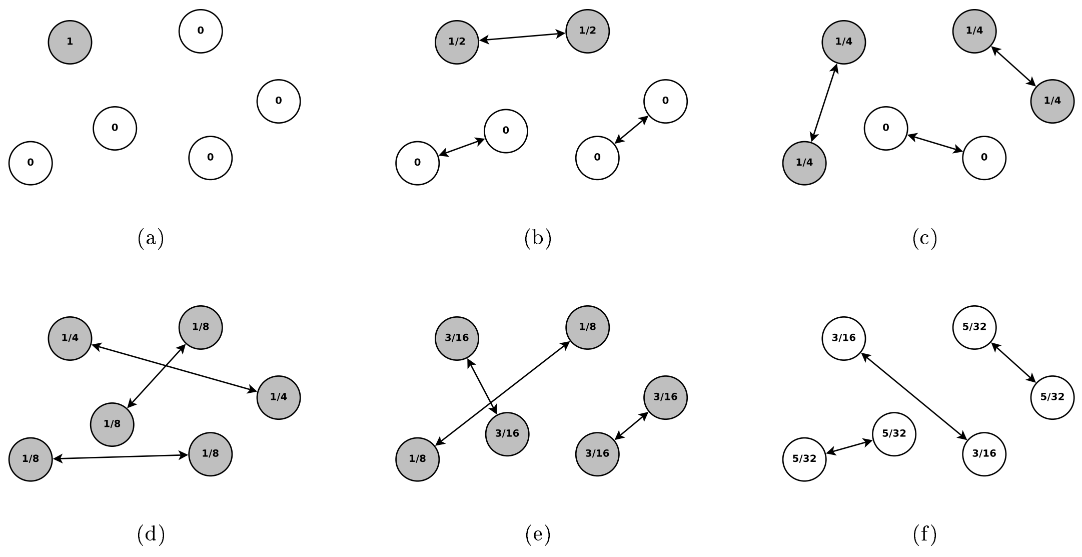


Figure 4.9: Approximation of the system size through aggregation.

Despite an extreme skew in the initial value distribution (1 for the peer that starts the instance and 0 for all other peers), the algorithm efficiently approximates the system size. This can be intuitively explained in the following way. At the beginning of an instance, very few peers hold non-zero values. When a non-zero holding peer performs a random gossip, with a very high probability, it passes half of its value to a zero-holding peer. Hence, the value held by the peer decreases at an exponential rate. At the same time, the number of peers participating in the instance grows exponentially, as in a push-pull epidemic. When only few non-participating peers are left, they join the instance within just few time steps.

It can be shown that the gossip exchange operations do not change the sums of values held by peers. Thus, the sum of all values is always equal to one, as in the initial configuration. Moreover, each time two peers perform an exchange, they can only reduce (or leave unchanged) the global variance. Intuitively, extreme values (either high or low compared to the mean $\frac{1}{N}$) have higher probability of being averaged out in a random exchange, since the expected peer value is equal to the mean $\frac{1}{N}$. Over time, the global variance converges to zero and the values held by peers converge to the mean $\frac{1}{N}$. A theoretical proof of this behaviour, as well as an experimental evaluation, are described in full detail in [82, 125, 81].

In order to estimate the minimum and maximum peer utility in the system, the $\text{init}_i()$ and $\text{join}_i()$ functions are defined as $U(p)$ for each peer p, $\text{merge}_i(x, y)$ is defined as $\min(x, y)$ and $\max(x, y)$, respectively, and $\text{update}_i(x)$ is defined as x. This way, the true system minimum (or maximum, respectively) is broadcast between peers in the system using a push-pull epidemic and the convergence speed is exponential [82].

Property	**init(p)**	**join(p)**	**merge(x, y)**	**update(x)**
Maximum	$U(p)$	$U(p)$	$\max(x, y)$	$Max_p \leftarrow x$
Minimum	$U(p)$	$U(p)$	$\min(x, y)$	$Min_p \leftarrow x$
Mean	$U(p)$	$U(p)$	$\frac{x+y}{2}$	$Avg_p \leftarrow x$
Mean Square	$U(p)^2$	$U(p)^2$	$\frac{x+y}{2}$	$S_p \leftarrow x$
Variance	—	—	—	$S_p - Avg_p^2$
Count	1	0	$\frac{x+y}{2}$	$N_p \leftarrow \frac{1}{x}$
Sum	—	—	—	$Avg_p \cdot N_p$
Fraction	$\begin{cases} 1 & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\begin{cases} 1 & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\frac{x+y}{2}$	$F_p \leftarrow x$
Conditional Count	—	—	—	$N_p \cdot F_p$
Conditional Sum	$\begin{cases} U(p) & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\begin{cases} U(p) & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\frac{x+y}{2}$	$x \cdot N_p$
Conditional Average	$\begin{cases} U(p) & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\begin{cases} U(p) & \text{if cond}(p) \\ 0 & \text{if not cond}(p) \end{cases}$	$\frac{x+y}{2}$	$x \cdot F_p$

Table 4.2: Aggregation settings.

The aggregation algorithm can also be used to estimate the fraction of peers in the system that satisfy a certain condition cond. To that end, $\text{init}_i()$ and $\text{join}_i()$ are set to one for peers that satisfy the condition cond and zero for peers that do not satisfy cond. Merge and update are defined as in the average utility calculation.

Given the fraction Q of peers that satisfy cond, and the system size N, the number of peers that satisfy cond can be calculated as $Q \cdot N$. Similar approaches allow the approximation of the average utility of peers that satisfy a given condition, and the sum of utility values for peers that satisfy a condition. Table 4.2 lists the definitions of $\text{init}_i()$, $\text{join}_i()$, $\text{merge}_i()$ and $\text{update}_i()$ for the approximation of a number of different system properties.

4.4.3 Histograms

In order to construct a tree-based gradient topology and elect super-peers using thresholds as described in section 3.1.1, peers need to have a knowledge of the current number of peers in the system, N, the total load in the system, L, and most importantly, the distribution of of peer utility, D, and capacity, D^c.

It is straight-forward to approximate the system size and load using the aggregation algorithm described in the previous section. However, D and D^c are functions rather than scalars and cannot be directly approximated through aggregation. Instead, they are interpolated using ***histograms***, which divide the peer utility spectrum into fixed intervals and approximate D and D^c in a finite number of points.

A cumulative peer utility histogram, H, is defined as an array consisting of b elements, called ***bins***, where the i'th element, $H(i)$, is equal to the number of peers in the system with utility above $min + i \cdot \lambda$

$$H(i) = \Big|\{p : U(p) \geq min + i \cdot \lambda\}\Big| \tag{4.19}$$

where λ is a parameter called ***bin width***, defined as $\frac{max-min}{b-1}$, and min and max are two variables that determine the ***range*** of the histogram. The number of bins, b, is also called histogram ***resolution***, $b > 1$. A utility histogram, H, is a discrete version of the utility distribution function, D, with b points positioned between min and max, since $H(i) = D(min + i \cdot \lambda)$ for each $i \in \mathbb{N}$ such that $0 \leq i < b$.

Similarly, a cumulative peer capacity histogram, H^c, is defined as a an array consisting of b elements, where the i'th element, $H^c(i)$, is equal to the total capacity of peers with utility above $min + i \cdot \lambda$

$$H^c(i) = \sum_{U(p) \geq i \cdot \lambda} C(p) \tag{4.20}$$

with the definitions of min, max, and λ as above for utility histograms. A capacity histogram is a discrete version of the capacity distribution function, as $H^c(i) = D^c(min + i \cdot \lambda)$ for $i \in \mathbb{N}$ such that $0 \leq i < b$.

Utility and capacity histograms can be generated using aggregation. According to the classification in Table 4.2, each bin $H(i)$ in a utility histogram is conditional count, and each bin $H^c(i)$ in a capacity histogram is a conditional sum, with the condition cond in both histograms defined as $U(p) \geq min{+}i{\cdot}\lambda$.

4.4.4 Histogram Interpolation

A utility histogram produced by the aggregation algorithm approximates the utility distribution function, D, in a limited number of points. In order to obtain the value of $D(u)$ for any utility u, a peer interpolates its utility histogram, H, and generates an approximation of D, denoted D_H.

When using linear interpolation, $D_H(u)$ is calculated as follows. For $u < min$, the interpolant is outside of the histogram range, and $D_H(u)$ is defined as $H(0)$. Similarly, for $u > min$, $D_H(u)$ is

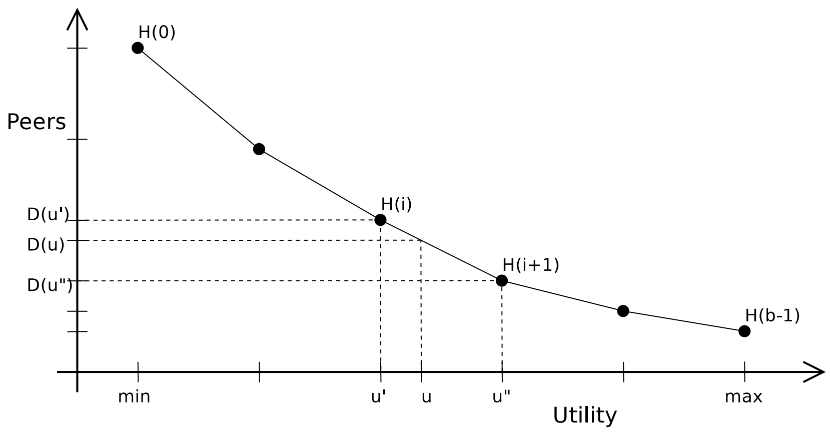


Figure 4.10: Histogram interpolation.

set to $H(b-1)$. For $min < u < max$, value $D_H(u)$ is approximated based on two histogram bins, $H(i)$ and $H(i+1)$, where $i = \lfloor \frac{u-min}{\lambda} \rfloor$. The interpolant u must lie between $u' = min + i\lambda$ and $u'' = min + (i+1)\lambda$, as shown in Figure 4.10, where the values of $D(u')$ and $D(u'')$ are known and are equal to $H(i)$ and $H(i+1)$. Using linear interpolation,

$$D_H(u) = D(u'') + (D(u') - D(u''))\frac{u'' - u}{u'' - u'} = H(i+1) + (H(i) - H(i+1))\frac{min + \lambda(i+1) - u}{\lambda}.$$

Naturally, any other interpolation methods can be used by peers to approximate $D(u)$, such as polynomial and spline interpolation.

Interpolation techniques can also be used by peers to estimate utility thresholds, t, such that $D(t) = K$ for some variable K. If the inverse function for D exists, $t = D^{-1}(K)$. A peer calculates t_H, which approximates t, based on a utility histogram H.

For $K > H(0)$, t_H is set to min, and similarly, for $K < H(b-1)$, t_H is set to max. For $H(0) \geq K \geq H(b-1)$, there must be histogram bins $H(i)$ and $H(i+1)$ such that $H(i) \geq K > H(i+1)$. The two histogram bins are correspond to the values of D for $u' = min + i\lambda$ and $u'' = min + \lambda(i+1)$. Using a linear interpolation,

$$t_H = u' + (u'' - u')\frac{D(u') - K}{D(u') - D(u'')} = min + i\lambda + \lambda\frac{H(i) - K}{H(i) - H(i+1)}.$$

Similar approach can be applied by peers to approximate the values of the capacity distribution function, D^c, based on capacity histograms, H^c.

4.4.5 Algorithm

This section describes a full version of the aggregation algorithm that approximates all system properties needed for the super-peer election. Each tuple T_p in $\mathcal{T}_p$ at peer p consists of 10 values

$$(id,\ ttl,\ w,\ l,\ min,\ max,\ h,\ h^c,\ hn,\ hx) \tag{4.21}$$

where id is the unique instance identifier, ttl is the time-to-live value, w is called the *weight* of the tuple and is used to estimate the system size, l is the current estimation of the system load, min and max are the current estimations of the minimum and maximum peer utility, h and h^c are two b-bin arrays used in the estimation of H and H^c, and hn and hx are the histogram range parameters used in this aggregation instance.

A peer joining the system obtains the current estimations of the system properties from one of its initial neighbours. At each time step, peer p starts a new aggregation instance with probability P_s by creating a tuple

$$(id,\ TTL,\ 1,\ L(p),\ U(p),\ U(p),\ I_p,\ I_p^c,\ Min_p,\ Max_p) \tag{4.22}$$

where Min_p and Max_p are the current estimations at peer p of the minimum and maximum peer utility in the system, I_p is an initial utility histogram created by p such that

$$I_p(i) = \begin{cases} 0 & \text{if } U(p) < hn + i \cdot \lambda \\ 1 & \text{if } U(p) \geq hn + i \cdot \lambda \end{cases} \tag{4.23}$$

and I_p^c is an initial capacity histogram where

$$I_p^c(i) = \begin{cases} 0 & \text{if } U(p) < hn + i \cdot \lambda \\ C(p) & \text{if } U(p) \geq hn + i \cdot \lambda \end{cases} \tag{4.24}$$

for $0 \leq i < b$. The bin width λ is defined as $\frac{hn-hx}{b-1}$, and b is a constant system parameter determining the histogram resolution, $b > 1$. Probability P_s is calculated as $\frac{1}{N_p \cdot F}$, where N_p is the current estimation of N at peer p, and F is a system constant that regulates the frequency of peers' starting aggregation instances. In a stable state, with a steady number of peers in the system, a new aggregation instance is created on average with frequency $\frac{1}{F}$.

A peer joins an aggregation instance by creating a tuple

$$(id,\ ttl,\ 0,\ L(p),\ U(p),\ U(p),\ I_p,\ I_p^c,\ hn,\ hx) \tag{4.25}$$

where ttl, hn and hx are obtained from the tuple received from the neighbour that invited p to join this aggregation instance.

When a tuple $(id,\ ttl_p,\ w_p,\ l_p,\ min_p,\ max_p,\ h_p,\ h_p^c,\ hn_p,\ hx_p)$ is merged with tuple $(id,\ ttl_q,\ w_q,\ l_q,\ min_q,\ max_q,\ l_q,\ h_q,\ h_q^c,\ hn_q,\ hx_q)$, it is replaced with

$$(id,\ \frac{ttl_p + ttl_q}{2} - 1,\ \frac{w_p + w_q}{2},\ \frac{l_p + l_q}{2},\ min_{pq},\ max_{pq},\ \frac{h_p + h_q}{2},\ \frac{h_p^c + h_q^c}{2},\ hn_p,\ hx_p) \tag{4.26}$$

where $min_{pq} = \min(min_p, min_q)$, $max_{pq} = \max(max_p, max_q)$, and all arithmetics on histograms h_p and h_q are performed pair-wise on each bin.

Finally, when an aggregation instance ends, a peer p updates its estimates of the system properties by setting $N_p = \frac{1}{w_p}$, $L_p = l_p$, $Min_p = min_p$, $Max_p = max_p$, and for each $i \in \mathbb{N}$ such that $0 \leq i < b$

$$H_p(i) = \frac{h_p(i)}{w_p} \tag{4.27}$$

and

$$H_p^c(i) = \frac{h_p^c(i)}{w_p}. \tag{4.28}$$

4.4.6 Handling Churn

In a stable population of peers, with no message loss, the aggregation algorithm has the following invariant. For each aggregation instance *id*, the weights of all tuples in the system associated with *id* sum up to 1. Moreover, the average of l, minimum of min, and maximum of max fields in all tuples associated with an aggregation instance are equal to the average load, minimum utility, and maximum utility, respectively, amongst all peers that participate in the instance. It can be shown that the aggregates converge over time to the true values of system properties [82, 125, 81].

In the presence of churn, the results produced by the aggregation algorithm may diverge from the true system properties, since the system is changing while the aggregation is running. However, the approximation error depends on the type of the aggregated system property. When estimating an average, such as the average peer utility, the expected value introduced by a joining peer, as well as the expected value removed from the system by a leaving peer, is equal to the overall system average. Hence, the expected value produced by aggregation is equal to the true average, and there is no bias towards higher or lower values. Churn increases the variance of the aggregated results, but does not change their expected value.

This property does not hold when estimating peer counts and sums, both conditional and unconditional, and histograms. This can be shown by the following analysis. A single run of an aggregation instance can be divided into two halves. In the first half, when the instance time-to-live is above $\frac{TTL}{2}$, peers can join the instance as well as leave the instance by departing from the P2P system. In the second half, peers can only leave and are not allowed to join.

Leaving peers always have non-negative weights, which they remove from the system. At the same time, joining peers initialise their weights to zero and hence do not increase the instance weight. These two facts cause the phenomenon of the aggregation *weight loss*. In the presence of churn, the longer an aggregation instance runs, the lower the weight of this instance.

Since the system size is approximated by the reciprocal of the tuple weight, peers tend to overestimate N due to the aggregation weight loss. Similarly, sums and histograms generated by the aggregation algorithm have a bias towards higher values.

In order to improve the accuracy of aggregation results in systems with churn, peers departing from the system, if they do not crash, perform a leave procedure, where they send all currently stored tuples with time-to-live values above $\frac{TTL}{2}$ to randomly chosen neighbours. The receiving neighbours add the weight of each received tuple to their corresponding tuples, joining aggregation instances when necessary. This way, peers prevent the aggregation weight loss and help to preserve the weight invariant.

Peers do not perform the leave procedure for tuples with time-to-live values below $\frac{TTL}{2}$ for two reasons. First, extra weight passed between peers shortly before the end of an aggregation instance may distort the aggregation results. Second, in the second half of an aggregation instance, no peers are allowed to join, and the total instance weight must be reduced when peers are leaving the system in order to obtain correct weight at each peer at the end of the aggregation instance.

4.4.7 Message Loss

In the presence of communication failures, similar to churn, aggregation may produce inaccurate estimations of system properties. When an aggregation request fails, a peer simply does not perform an gossip exchange with a selected neighbour and does not update its tuples. Hence, request loss decreases the convergence speed of aggregates towards the true system properties, but does not produce any bias in the generated results.

When a response message is lost, the requestee updates its tuples, as if an exchange took place, while the requester dismisses the exchange assuming a communication failure. Thus, the algorithm's invariants, such as mass conservation, are violated and the results produced by aggregation become inaccurate. Moreover, a systematic bias may appear, similar to the weight loss effect in the presence of churn.

However, in many P2P systems, it can be expected that response messages are significantly less likely to fail compared with request messages. A request message is lost when a peer attempts to send it to a neighbour that no longer exists in the system. In dynamic systems, peers are always likely to have such outdated entries in their neighbourhood sets due to churn, communication latency, and failures. At the same time, response messages are only sent to peers that have just issued an aggregation request, and hence are likely to be on-line. Furthermore, with some probability, request messages are blocked by firewalls and NATs installed between peers. However, a response message is less likely to be blocked by a firewall, since it is only sent when two peers have already transmitted a request, and hence there is no firewall between the peers, or the firewall permits a communication between them.

Since push-pull aggregation algorithms can handle request message loss, it is argued that they are more applicable to P2P systems than push-based algorithms, such as those proposed by Kempe et al. [89] and Sacha et al. [165]. Moreover, it is believed that push-pull aggregation algorithms to outperform push-based approaches since push-pull epidemics spread faster than push-only epidemics [80].

4.4.8 Increasing Fan-Out

In the aggregation algorithm described in section 4.4.5, an aggregation instance is started with an average frequency of $\frac{1}{F}$. Since a peer performs on average two gossip exchanges per time step and the time-to-live of a tuple is decreased by $\frac{1}{2}$ at each gossip exchange, an aggregation instance lasts approximately TTL time steps and a peer participates on average in less than $\frac{TTL}{F}$ aggregation instances. Hence, a peer sends and receives on average two messages per time step and the message size is proportional to $\frac{TTL}{F}$.

When the frequency of aggregation is increased, the aggregates maintained by peers are more up-to-date, but the cost of the algorithm increases proportionally to $\frac{1}{F}$. Similarly, the cost grows when TTL is increased. A third approach is increase the algorithm *fan-out*, i.e., the number of neighbours contacted per time step. In this version of the algorithm, the active thread at peer p performs one gossip exchange if p participates in no aggregation instances (i.e., $\mathcal{T}_p = \emptyset$), and performs G gossip exchanges with randomly selected neighbours if p runs at least one aggregation instance (i.e., $\mathcal{T}_p > \emptyset$), where G is a system constant. The passive thread remains unchanged.

An aggregation instance is started with an average frequency of $\frac{1}{F}$, as before, but it lasts only $\frac{TTL}{G}$ time steps, since peers perform on average $2G$ gossip exchanges when they participate in aggregation instances. Thus, on average, a peer participates in approximately $\frac{TTL}{F \cdot G}$ instances. The average number of messages generated by a peer per time step, assuming $\frac{TTL}{F \cdot G} < 1$, is then

$$2\left(\frac{TTL}{F \cdot G}G + \left(1 - \frac{TTL}{F \cdot G}\right)\right) = 2 + 2\frac{TTL}{F} \cdot \frac{G-1}{G} < 2 + 2\frac{TTL}{F} \tag{4.29}$$

since a peer initiates G gossip exchanges with probability $\frac{TTL}{F \cdot G}$ and one exchange with probability $1 - \frac{TTL}{F \cdot G}$, and for each exchange peers generate two messages. Thus, neither the average number of generated messages nor the average message size grow with increasing G. When $\frac{TTL}{F \cdot G} \geq 1$, a peer exchanges approximately $2G$ messages per time step.

As G is increased, aggregation instances become shorter ($\frac{TTL}{G}$ time steps), and hence, fewer peers join and leave the system when an instance is executing. This way, the algorithm reduces the impact of churn on the aggregation results. In particular, it reduces the instance weight loss. Furthermore, as the instance is shorter, aggregates are calculated more quickly and are hence more up-to-date. The only cost associated with increasing fan-out is that peers need to send bursts of messages per time step, which may cause a network congestion and a higher message loss rate.

4.5 Super-Peer Election

The super-peer election algorithm, executed locally by each peer in the system, classifies each peer as either a super-peer or an ordinary peer. The algorithm has the property that it elects super-peers with globally highest utility, and it maintains a highest-utility super-peer set as the system evolves.

```
loop
    $t \leftarrow$ calculate threshold
    if $U(p) > t$ and $p$ is client then
        become super-peer
    end if
    if $U(p) < t$ and $p$ is super-peer then
        become ordinary peer
    end if
    wait until next cycle
end loop
```

(a) Single threshold

```
loop
    if super-peer then
        $t \leftarrow$ calculate lower threshold
        if $U(p) < t$ then
            become ordinary peer
        end if
    else
        $t \leftarrow$ calculate upper threshold
        if $U(p) > t$ then
            become super-peer
        end if
    end if
    wait until next cycle
end loop
```

(b) Two thresholds

Figure 4.11: Super-peer election algorithms.

The algorithm also limits the number of switches between ordinary peers and super-peers in order to reduce the associated overhead.

4.5.1 Single-Threshold Election

Given the approximations of the system size, N, system load, L, peer utility distribution D, peer capacity distribution D^c, and other aggregates generated by the algorithm described in section 4.4.5, every peer can calculate super-peer election thresholds, as defined in section 3.1.1. In the simplest case, every peer p periodically calculates the super-peer threshold, t, and compares it with its own utility $U(p)$, as shown in Figure 4.11(a). All peers with utility above the threshold become super-peers, while peers below the threshold become clients.

However, in a dynamic system, the super-peer election threshold constantly changes over time due to peer arrivals and departures, system load fluctuations, and statistical error produced by the aggregation algorithm. Moreover, the utility of individual peers may change, as discussed in section 4.2. As a consequence, peers may frequently cross the super-peer threshold and switch their roles between being super-peers and clients, increasing the system overhead, for example due to the required data migration and synchronisation between super-peers.

4.5.2 Two-Threshold Election

In order to prevent peers from frequently switching their roles between super-peers and clients, the system uses two separate thresholds for the super-peer election, an *upper threshold*, t_u, and a *lower threshold*, t_l, where $t_u > t_l$. An ordinary peer becomes a super-peer when its utility exceeds t_u, while

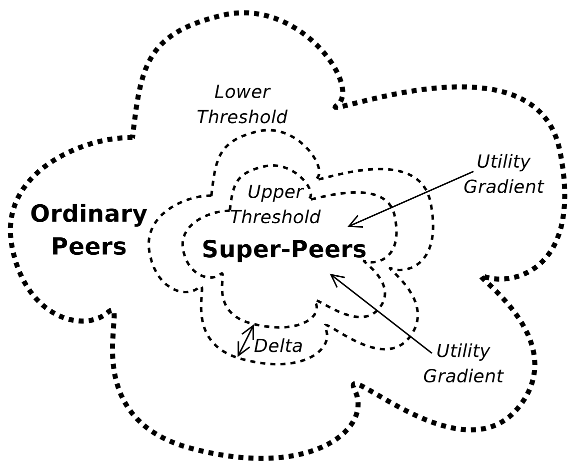


Figure 4.12: Super-peer election using two utility thresholds.

a peer stops to be super-peer when its utility falls below t_l, as shown in figure 4.11(b). This way, peers with utility above t_u are always super-peers, peers with utility below t_l are always clients, while peers between the two thresholds can be either super-peers or clients, depending on their previous utility values and previous election thresholds. Thus, the system exhibits the property of hysteresis. The minimum utility change required for a peer to switch its status, assuming constant election thresholds, is $\triangle = t_u - t_l$. A gradient topology with super-peers elected using two utility thresholds is shown in figure 4.12.

The upper and lower thresholds determine the minimum and maximum number of super-peers in the system. Thus, they restrict the number of super-peers in the system to an interval. For example, in order to maintain between K_1 and K_2 super-peers, where $K_1 \leq K_2$, thresholds t_u and t_l are calculated based on the utility distribution, D, so that $D(t_u) = K_1$ and $D(t_l) = K_2$. Likewise, in order to adapt the number of super-peers to the system load, L, while maintaining the average super-peer utilisation between W_1 and W_2, where $W_1 \leq W_2$, t_u and t_l must satisfy $D^c(t_l) \cdot W_1 = L$ and $D^c(t_u) \cdot W_2 = L$. Similar formulas can be derived for other threshold defined in section 3.1.1.

4.5.3 Election without Demotion

As the gap between the upper and lower threshold increases, the restriction on the number of super-peers in the system is relaxed, but the switches between super-peers and clients become less probable. In order to eliminate super-peer demotions completely, peers may use a variant of the super-peer election algorithm, shown in figure 4.13, super-peers are elected for their lifetimes, i.e, until they leave the system.

In order to control the number of super-peers in this algorithm, two changes are required in the threshold calculation and histogram generation. First, the histograms H and H^c are calculated in

```
1: loop
2:   if ordinary peer then
3:     $t \leftarrow$ calculate threshold
4:     if $U(p) > t$ then
5:       become super-peer
6:     end if
7:   end if
8:   wait until next cycle
9: end loop
```

Figure 4.13: Super-peer election with no super-peer demotion.

such a way that they contain only peers that can be promoted to super-peers, i.e., clients. For that purpose, in the aggregation algorithm, when peers start or join an aggregation instance, they define their initial utility histogram as

$$I_p(i) = \begin{cases} 0 & \text{if } U(p) < i \cdot \lambda \text{ or } p \text{ is super-peer} \\ 1 & \text{if } U(p) \geq i \cdot \lambda \text{ and } p \text{ is client} \end{cases} \tag{4.30}$$

and for the capacity histogram as

$$I_p^c(i) = \begin{cases} 0 & \text{if } U(p) < i \cdot \lambda \text{ or } p \text{ is super-peer} \\ C(p) & \text{if } U(p) \geq i \cdot \lambda \text{ and } p \text{ is client} \end{cases}. \tag{4.31}$$

This way, $H(i)$ contains the number of *clients* with utility above $min + \lambda i$, and $H^c(i)$ contains the total capacity of *clients* with utility above $min + \lambda i$. Through interpolation, histogram H can be used to estimate the *client* utility distribution function, $\hat{D}$, where

$$\hat{D}(u) = \Big|\{p : p \text{ is client and } U(p) \geq i \cdot \lambda\}\Big|. \tag{4.32}$$

Similarly, H^c can be used to estimate the client capacity distribution, $\hat{D^c}$, where

$$\hat{D^c}(u) = \sum_{\substack{U(p)>u \\ p \text{ is client}}} C(p). \tag{4.33}$$

In order to maintain K super-peers using a top-K threshold, a peer checks at each time step how many new super-peers need to be created in the current state of the system. Assuming the peer can estimate the current number of super-peers, K', using aggregation, the number of required new super-peers is $K" = K - K'$. If $K" \leq 0$, the threshold is set to infinity, as no super-peers need to be created. If $K" > 0$, the super-peer election threshold, t, is derived from equation $\hat{D}(t) = K"$. This way, in a system with dynamic peer population, fluctuating peer utility, load, and election thresholds, super-peers are never demoted to clients, but the system creates no more than K super-peers at any given time.

Similarly, in order to maintain a super-peer set with a total capacity of C, for example equal to the total system load, L, peers estimate the current super-peer capacity in the system, C', using aggregation, and calculate a super-peer election threshold according to formula $\hat{D^c}(t) = C - C'$.

4.5.4 Client-Based Election

The calculation of the clients threshold, as defined by formula 3.6 in section 3.1.1, requires the knowledge of both D and D^c. This section describes an approach that allows the estimation of the clients threshold using D only.

Each peer estimates the current system size, N, and the current super-peer ratio, Q. The current number of clients, and hence the required super-peer capacity to handle these clients, is given by $N(1-Q)$. Peers calculate the super-peer election threshold as a utility value, t, such that $D^c(t) = N_p(1-Q_p)$.

Although super-peer sets elected this way do not necessarily have the exact capacity required to handle the clients in the system, they converge to the optimum size when the election is repeated. When the algorithm elects too many super-peers, the number of clients decreases, and fewer super-peers are elected in the following iteration. Similarly, when too few super-peers are elected, the number of clients increases, and the algorithm elects more super-peers in the following run.

4.6 Rank Estimation

This section describes three algorithms that allow peers to estimate their ranks. The first method is based on utility histograms, the second method is based on utility successor sets, and the last method is a combination of the first two. Peer ranks are required for the construction of tree-based gradient topologies.

4.6.1 Histogram-Based Method

The simplest way to calculate peer ranks is to use utility histograms, assuming they are generated by aggregation. The rank of a peer p is equal to $R_p(p) = D_H(U_p(p))$, where D_H is a utility distribution function approximated using histogram H_p at peer p, and $U_p(p)$ is p's estimation of its own utility.

The main drawback of this rank estimation method is that it generates a significant error for the highest-utility peers. This is because the highest utility peers belong to the last bins in the histogram (i.e., close to $b-1$), which contain very few samples, and the interpolation of these few samples is likely to produce a significant approximation error.

4.6.2 Successor-Based Method

Another approach to the peer rank estimation is to use the information about neighbours in successor sets. In order to calculate $R_p(p)$, a peer p first creates a subset, S'_p, of its successor set, S_p, which contains peers with utility higher than $U_p(p)$

$$S'_p = \{q \in S_p : \; U_p(q) > U_p(p)\}. \tag{4.34}$$

In a stable state of the system, where the successor sets are fully optimised according to the preference function defined in section 4.3.5.1, S'_p should contain M peers with ranks equal to

$$R(p) - 1, \quad R(p) - 2, \; \dots \; R(p) - M \tag{4.35}$$

where M is equal to $\min(R(p),\ S^*_p)$. The sum of the ranks of all peers in S'_p is then equal to

$$R(p) \cdot M - \frac{M(M+1)}{2}. \tag{4.36}$$

This information can be used by peer p to estimate its rank $R_p(p)$. If S'_p is empty, peer p assumes that it has the highest utility in the system and sets its rank to zero. Otherwise, peer p calculates the sum of the ranks of its neighbours in S'_p and sets its rank to

$$R_p(p) = \frac{\sum_{q \in S'_p} R_p(q)}{|S'_p|} + \frac{|S'_p| + 1}{2} \tag{4.37}$$

where the rank estimation formula 4.37 is derived directly from 4.36.

In principle, any neighbour in S'_p can be used for the estimation of $R(p)$, given formula 4.35. However, by summing and averaging the ranks of all neighbours in S'_p, peer p uses all available information for the estimation of its rank, and reduces the estimation error, since the individual errors introduced by each $R_p(q)$ variable may cancel out.

In order to initialise its rank when joining the system, peer p selects then the highest utility neighbour, m, in its initial neighbour set, $\mathcal{N}_p$, such that $U_p(m) < U_p(p)$, and the lowest utility neighbour, $n \in \mathcal{N}_p$, such that $U_p(n) > U_p(p)$. The rank of peer p is calculated as the average between $R_p(m)$ and $R_p(n)$

$$R_p(p) = \frac{R_p(m) + R_p(n)}{2}. \tag{4.38}$$

The successor-based rank estimation method requires that peers maintain globally optimal successor sets. If a peer does not connect to a higher-utility neighbour that would belong to its successor set in an optimal topology configuration, the peer is likely to underestimate its rank, i.e., calculate its rank $R_p(p)$ such that $R_p(p) < R(p)$. Moreover, as peers calculate their ranks based on the rank estimates of their higher-utility neighbours, rank estimation errors are propagated, and gradually magnified, from the highest-utility peers to the lowest-utility peers.

Thus, in large and dynamic systems, due to churn and communication failures, peers tend to underestimate their ranks, and the rank estimation error increases together with peer rank. Due to that, the successor-based rank estimation method is only applicable to small P2P systems.

4.6.3 Mixed Method

In order to combine the advantages of both the histogram-based and successor-based rank estimation methods, a mixed approach is proposed, where the highest-utility peers use successor sets to estimate their ranks, and all remaining peers use histograms for the rank estimation.

```
 1: GetRank():
 2: if $p$ has no successor set then
 3:     $R_p(p) \leftarrow$ histogram-based rank
 4:     if $R_p(p) < R_s$ then
 5:         create successor and predecessor sets
 6:     end if
 7: else
 8:     $R_p(p) \leftarrow$ successor-based rank
 9:     if $R_p(p) \geq R_s$ then
10:         $r \leftarrow$ histogram-based rank
11:         if $r \geq R_s$ then
12:             remove successor and predecessor sets
13:         end if
14:     end if
15: end if
16: return $R_p(p)$
```

Figure 4.14: Mixed rank estimation method.

The pseudo-code of the mixed method at peer p is shown in Figure 4.14. If peer p has no successor set (line 2), it approximates its rank using a utility histogram (line 3), as described in section 4.6.1. Furthermore, if the estimated rank of p is lower than a threshold value R_s (line 4), peer p creates successor and predecessor sets and joins the highest utility peers (line 5).

If peer p estimates its rank when it already has a successor set (line 8), it applies the successor-based method described in section 4.6.2 (line 8). If the estimated rank $R_p(p)$ is equal to or higher than R_s (line 9), peer p generates another rank estimation, r, using its current histogram (line 10), and removes its successor and predecessor sets if $r \geq R_s$ (line 11-12).

The extra check between $R_p(p)$ and r in line 11 of the algorithm is necessary to prevent peer p from cyclically adding and removing its successor and predecessor sets. Such a situation could have happened, if the rank estimation obtained using the histogram-based method was below R_s, while the rank estimated using the successor set at peer p was above R_s. In the algorithm shown in figure 4.14, a peer removes the successor and predecessor sets only if both estimation methods produce ranks below R_s. Moreover, in case of an inconsistency between the ranks obtained using the two methods, the algorithm assumes that the successor-based rank estimation method is more accurate.

Further improvements to the algorithm may be introduced in order to allow a gradual transition between rank estimates obtained using the two methods. For example, the rank of peer p can be calculated as $R_p(p) = \gamma r_1 + (1 - \gamma) r_2$, where $0 < \gamma < 1$ is a system parameter, and r_1 and r_2 are two rank estimates produces by the two methods.

4.7 Gradient Search

Gradient search is a multi-hop message routing algorithm, that can deliver a message from potentially any peer in the system to a high utility peer, i.e., a peer with utility above a given utility threshold. In gradient search, a peer p greedily forwards each message that it currently holds to its highest-utility neighbour, i.e., to a neighbour q whose utility is equal to

$$\max_{q \in \mathcal{N}_p} U_p(q). \tag{4.39}$$

Thus, messages are forwarded along the utility gradient, as in hill climbing and similar techniques.

If gradient search is performed over a TGT, peers are most likely to forward messages to their parents. As shown in theorem 3.3, the worst-case cost for routing a message from a peer p to a super-peer in a TGT is $O(\log_B \frac{R(p)}{K})$ overlay hops, where K is the total number of super-peers in the overlay. Message paths can be even shorter than $O(\log_B \frac{R(p)}{K})$ hops, if peers use their random sets for routing, which potentially contain neighbours with higher utility than that of parents.

By exploiting the information contained in the topology, gradient search achieves a significantly better performance than general-purpose search techniques for unstructured P2P networks, such as flooding or random walking [164]. Gradient search also reduces message loss rate by preferentially forwarding messages to high utility peers, assuming peer utility is based on a notion of peer stability [164].

4.7.1 Local Maxima

Local maxima should not occur in an idealised gradient topology, however, every P2P system is under constant churn and a gradient topology may undergo local perturbations from the idealised structure.

There are a number of different approaches to prevent message looping in the presence of a local maxima in a gradient topology. One approach is to assign a unique identifier to every generated message, and to maintain at each peer a cache of recently forwarded messages. For each message, the cache contains the neighbours to which the message was forwarded. When a peer receives a message that it has already routed, it forwards the message to a neighbour that is not yet present in the cache for this message. If no such neighbour can be chosen, the message is forwarded to a random neighbour in order to escape the local deviation in the topology structure.

Another approach is to append a list of visited peers to each search message, and to impose a constraint that forbids forwarding messages to previously visited peers. Again, if a peer is not able to forward a message to a non-visited neighbour, it routes the message randomly. Additionally, a time-to-live value is added to each message, which is decremented by one at each overlay hop. Messages with negative time-to-live values are discarded by peers in order to prevent infinite message circulation in the overlay.

4.8 Bootstrap

Bootstrap is a process in which a peer obtains an initial configuration in order to join the system. In P2P systems, this primarily involves obtaining addresses of initial neighbours. Once a peer connects to at least one neighbour, it can receive from this neighbour the addresses of other peers in the system as well as other initialisation data, such as the current values of aggregates.

However, initial neighbour discovery is challenging in wide-area networks, such as the Internet, since a broadcast facility is not widely available and peers cannot simply broadcast a join request to all addresses in the network. In particular, the IP multicast protocol has not been commonly adopted by Internet service providers due to design and deployment difficulties [50]. Most existing P2P systems rely on centralised bootstrap servers that maintain lists of peer addresses.

This section describes a bootstrap procedure that consists of two stages. In the first stage, a peer attempts to obtain initial neighbour addresses from a local cache saved during the previous session, for example on a local disk. This can be very effective; Stutzbach et al [181] analyse statistical properties of peer session times in a number of deployed P2P systems and show that if a peer caches the addresses of several high-uptime neighbours, there is a high probability that some of these high-uptime neighbours will be on-line during the peer's subsequent session. Furthermore, such a bootstrap strategy is fully decentralised, as it does not require any fixed infrastructure, and it scales with the system size.

However, if all addresses in the cache are unavailable or the cache is empty, for example if the peer is joining the system for the first time, the peer needs to have an alternative bootstrap mechanism. In the second stage, peers obtain initial neighbour addresses from a bootstrap node. The IP addresses of the bootstrap nodes are either hard-coded in the application, or preferably, are obtained by resolving well known domain names. This latter approach allows greater flexibility, as bootstrap nodes can be added or removed over the course of the system's lifetime. Moreover, the domain name may resolve to a number of bootstrap node addresses, for example selected using a round-robin strategy, in order to balance the load between bootstrap nodes.

4.8.1 Bootstrap Nodes

Each bootstrap node is independent and maintains its own cache containing peer addresses. The cache size and the update strategy are critical in a P2P system, as the bootstrap process may have a strong impact on the system topology, particularly in the case of high churn rates. If the cache is too small, subsequently joining peers have similar initial neighbours, and as a consequence, the system topology may become highly clustered or even disconnected if the peers in the cache become overloaded. On the other hand, a large cache is more difficult to keep up to date and may contain addresses of peers that have already left the system.

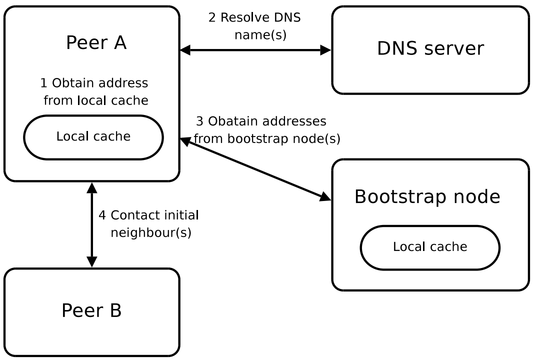

Peer A
1 Obtain address from local cache
Local cache
2 Resolve DNS name(s)
DNS server
3 Obatain addresses from bootstrap node(s)
Bootstrap node
Local cache
4 Contact initial neighbour(s)
Peer B

Figure 4.15: Bootstrap process of Peer A.

A simple cache update strategy is to add the addresses of currently bootstrapped peers to the cache and to remove them in the FIFO order. However, this strategy has the drawback that it generates topologies where joining peers are highly connected with each other. In such topologies, joining peers are less likely to discover their optimal neighbours, since most peers they communicate with are also joining the system and have not yet optimised their configurations.

A better approach is to continuously "crawl" the P2P network and "harvest" available peer addresses. In this case, the bootstrap node periodically selects a random peer from the cache, obtains the peer's current neighbours, preferably from the peer's random neighbour subset, adds these neighbours to the cache, and removes the oldest entries in the cache. This has the advantage that the addresses stored in the cache are close to a random sample from all peers in the system.

Figure 4.15 shows the bootstrap process of Peer A. In step (1), Peer A obtains a list of initial neighbour addresses from its local cache. If any of the initial neighbours are alive, the peer goes to step (4). Otherwise, in step (3), the peer resolves a well-known domain name and obtains an IP address of a bootstrap node. From the bootstrap node, in step (3), Peer A receives a list of addresses of initial neighbours. Finally, in step (4), Peer A contacts one of the initial neighbours (Peer B) and obtains additional information about the system, such as current values of the aggregates.

4.9 Summary

This chapter describes the design of the gradient topology and its main component. The chapter shows a number of utility functions and neighbour selection algorithms that generate TGTs, aggregation and ranking algorithms that estimate global system properties, super-peer election strategies, gradient search variations, and an approach to bootstrap peers joining the system. The next chapter evaluates these algorithms, shows their performance characteristics and trade-offs, and compares them to a number of state-of-the-art systems.

Chapter 5

Evaluation

The description of the gradient topology evaluation consists of six sections. The first section contains a functional comparison between gradient topologies and a number of state-of-the-art super-peer election systems. It shows that gradient topologies offer more flexible and powerful election mechanisms than the other considered systems. The second section describes a custom-built P2P simulator, which is used later to run performance experiments. The third section compares the performance of gradient topologies and selected state-of-the-art systems, and shows that the election algorithms used in the gradient topologies generate higher-quality super-peer sets, according to a number of different metrics, at a similar cost, compared to the other systems. The fourth section describes a detailed evaluation and analysis of gradient topologies and their specific features that are not supported in the other systems. The purpose of this section is to verify the theoretical properties of gradient topologies introduced earlier in chapter 3, and to validate the design of the algorithms introduced in chapter 4. The fifth section validates the custom-built simulator. Finally, the last section demonstrates the practical viability of gradient topologies by applying the experimental results to a sample application scenario.

5.1 Functional Comparison

The functional comparison is intended to examine the ability of systems to generate, control and adapt super-peer sets in the overlay according to the higher-level application requirements. For this reason, the comparison is concerned with the state-of-the-art systems that are classified as adaptive and DHT-based in chapter 2, sections 2.5 and 2.4. Group-based super-peer election systems, described in section 2.3, are not considered in the comparison, since most of these systems depend on application-specific peer properties, such as physical location, position in a virtual space, and semantic properties, and introduce additional application-specific constraints in the super-peer election which cannot be directly

	Fixed number	Fixed ratio	Fixed capacity	Clients-based	Load-based	Latency-based	Multiple sets	Dynamic capacity
SOLE	✓	–	–	–	–	–	✓	–
Hierarchical DHTs	✓	–	–	–	–	–	✓	?
HONet	✓	–	–	–	–	✓	?	–
SPChord	–	✓	–	?	?	✓	–	?
Structured Superpeers	–	–	–	?	?	–	–	?
DLM	–	✓	–	?	?	–	–	?
SG-1	–	–	–	✓	–	–	–	–
SG-2	–	–	–	✓	–	✓	?	?
Gradient Topology	✓	✓	✓	✓	✓	–	✓	✓

✓supported – not supported ? hypothetical

Table 5.1: Functional system comparison.

compared with general-purpose super-peer election algorithms addressed in this thesis. The simplest approaches to super-peer election, described in section 2.2, are also excluded from the comparison.

5.1.1 Feature Set

The systems are compared based on eight features. Six of these features correspond to super-peer election criteria identified in the super-peer systems reviewed in chapter 2. These criteria can be described as a fixed number of super-peers, fixed ratio of super-peers to clients, fixed super-peer capacity, super-peer capacity based on the client set, super-peer capacity based on load, and latency-based super-peer constraints. Additionally, two features are added, which are believed to be highly relevant to many P2P applications. These are the support for peers that dynamically change their properties, and election of multiple super-peer sets.

5.1.2 Analysis

SOLE [105], covered in section 2.4.3, allows the election of fixed numbers of super-peers, determined by global super-peer identifiers. It supports the election of multiple super-peer sets, as each application running on top of SOLE can define its own set of super-peer identifiers. However, SOLE does not introduce any notion of peer capability and elects super-peers solely based on their positions in the DHT.

Hierarchical DHTs [59], described in section 2.4.4, also elect fixed numbers of super-peers, determined by peer groups. Each group in the system elects its highest-capability peer as the group's super-peer. In cases where peer capabilities change over time, the system can swap super-peers with higher-capability clients, but does not provide any mechanisms to reduce the associated overhead. Hierarchical DHTs can be generalised to multi-level hierarchies with super-peers elected at each level in the system structure, but the design of such an approach is not described in [59].

HONet [184], introduced in section 2.4.5, elects super-peers based on peer coordinates in a virtual space. A super-peer is created each time a peer is located beyond a fixed distance, T, from the existing super-peers. Thus, the maximum number of super-peers elected in a HONet system is fixed and is determined by the size of the virtual space and the super-peer coverage, T. Hypothetically, each application built on top of a HONet system could elect its own super-peer set with a custom value for T, using the virtual peer space and the DHT overlay, but this approach is not considered in [184].

In SPChord [101], described in section 2.4.6, a new super-peer is created when the size of a cluster exceeds a fixed threshold, c, and a super-peer is removed when the size of a cluster falls below $\frac{c}{2}$. Thus, SPChord maintains a ratio of clients to super-peers between c and $\frac{c}{2}$. Peers are positioned in a virtual coordinate system, and each client is associated with the closest super-peer, while super-peers are elected based on their uptimes. The super-peer election criteria can be potentially changed, but the system does not provide any mechanisms to stabilise the topology and reduce the number of swappings between super-peers and clients if peer properties fluctuate over time. Parameter c can be theoretically defined for each peer individually, reflecting peer capacity, in which case SPChord would generate super-peer sets with total capacity equal to or higher than the number of clients in the system. Moreover, it may also be possible to split and merge super-peer clusters depending on the load on super-peers, but all these approaches are not addressed in [101].

Structured Superpeers [123], covered in section 2.4.7, is a system similar in many respects to SPChord. It splits and merges clusters, defined as arcs in an outer DHT ring, based on the number of clients and load associated with each super-peer. However, due to the lack of detailed system description, it is difficult to determine exactly how the super-peers are elected and what are the properties of the election mechanism.

DLM [196], described in section 2.5.3, is an election algorithm that maintain a fixed ratio of super-peers to clients in a P2P system. Super-peers are elected based on their capacity and uptime, but similar to hierarchical SPChord and Structured Superpeers, other super-peer election criteria could potentially be applied in DLM. The DLM algorithm can also handle peers with dynamically changing capacity, but [196] does not address the problem of super-peer stability and switches between super-peers and clients.

SG-1 [124], addressed in section 2.5.1, generates a P2P topology where the total capacity of super-peers is approximately equal to the number of clients in the system, and the capacity of super-peers is maximised. The algorithm assumes that peer capacity is constant, and continuously attempts to transfer clients from lower-capacity super-peers to higher-capacity super-peers, reducing any redundant super-peers. In order to elect two super-peer sets, each peer would have to participate in two *superpeer* and two *underloaded* overlays, and would have to maintain two *clients* sets. Hence, the system overhead would almost double.

SG-2 [83], discussed in section 2.5.2, extends the SG-1 algorithm and adds latency constraints to the generated P2P topology. Super-peers in SG-2 cover the system space and periodically broadcast advertisement messages in their neighbourhood, while clients are transferred from lower-capacity super-peers to higher-capacity super-peers. Hypothetically, independent sets of super-peers can be created in SG-2 if each set of super-peers generates and broadcasts it own type of super-peer messages, but such an approach is not followed in [83]. Moreover, since super-peer election is localised in SG-2, the algorithm should work with peers dynamically changing their capacity.

Gradient topologies, together with the aggregation and election algorithms described in this thesis, allow the election of super-peer sets with fixed sizes, proportional sizes to the number of peers, fixed capacity, capacity based on the number of clients, and capacity based on the system load. Double super-peer election thresholds, and other election techniques described in section 4.5, allow peers to minimise the number of switches between super-peers and clients as the utility of peers changes. Gradient topologies also support the election of multiple super-peer sets, since peers can calculate any number of super-peer election thresholds, and the topology structure guarantees that each super-peer set is connected and can be discovered using gradient search.

5.1.3 Summary

Table 5.1 summarises the results of the comparison. It can be concluded that gradient topologies, together with aggregation-based election techniques, offer more flexible and powerful super-peer election mechanisms compared with the existing P2P systems, and thus extend the current state-of-the-art knowledge on super-peers, and more generally, heterogeneity exploitation in P2P systems.

5.2 Peer-to-Peer Simulator

Performance evaluation is especially important when designing a novel P2P topology, such as the gradient topology, since P2P systems usually exhibit complex, dynamic behaviour that is difficult to predict a priori. Theoretical system analysis is difficult, and often infeasible in practice, due to the system complexity. At the same time, a full implementation and deployment of a P2P system on a realistic scale requires extremely large amounts of resources, such as machines and users, that are prohibitive in most circumstances. Consequently, the approach followed in this thesis is simulation experiments.

However, designing P2P simulations is also challenging. The designer has to decide upon numerous system assumptions and parameters, where the appropriate choices or parameter values are non-trivial to determine. Furthermore, dependencies between different elements of a complex system are often non-linear, and a relatively small change of one parameter may result in a dramatic change in the system behaviour. Moreover, due to the large size and complexity, full-scale P2P systems are not amenable to visualisation techniques, as a display millions of peers, connections, and messages is not human-readable. P2P simulations must continuously collect and aggregate statistical information about the system in order to detect topology partitions, identify bottlenecks, measure global system properties, etc. Such frequent and extensive measurements are often computationally expensive, which adds further challenges to analysing P2P systems.

5.2.1 Existing Simulators

A number of P2P simulators have been used for evaluating P2P systems, as summarised in [131, 130]. The choice of the P2P simulator is an important part of the evaluation strategy, since many existing simulators differ in the system models and assumptions, and introduce different constraints on the performed experiments. Typically, P2P systems are evaluated in Discrete Event Simulators (DES), where the operation of a system is represented as a chronological sequence of instant events that atomically switch the system state [157]. The state is usually stored on a single machine and all the processing is often performed in one thread. However, simulators vary in the extent to which they model the low-level network underlying the P2P system.

At one extreme are simulators such as NS-2 [25, 13], which model the entire TCP/IP stack at every network node. Such approaches are expensive, and simulators based on NS-2 can usually support at most a few thousand simultaneous nodes [74, 131, 130]. Moreover, it is argued that rigid network modelling at the packet level is not necessary when evaluating P2P applications, since most interesting features of P2P systems can be observed at the application protocol level [120, 107].

At the other extreme are P2P simulators where the network layer is extremely simplified and messages are passed between peers through direct method calls, with no delay. This approach is followed in PeerSim, a P2P simulator which has been used for evaluating algorithms such as SG-1 [124], SG-2 [83], Newscast [80], and push-pull aggregation [82]. Apart from the networking model, PeerSim also simplifies the control flow. Instead of ordering and processing discrete events using a priority queue, as do most other P2P simulators, PeerSim uses a simple cycle model, where every peer in the system executes a *step* procedure at each cycle of a global simulation loop. Due to these simplifications, it is believed that PeerSim can simulate up to one million simultaneous nodes [131, 124].

A number of other P2P simulators exist and have a different balance between performance and accurate low-level network modelling. Many simulators, such as p-sim [120], PlanetSim [107], and the

Simulator	Network model	Flow control	Scalability	Code	GUI
He et al. [74]	Full network stack (NS2, GTNets, GT-ITM latency)	Event queue	2000 nodes	C++	no
OverSim [16]	Full network stack (OMNeT++), Euclidean latency	Event queue	10^5 nodes	C++	yes
Narses [63]	Flow-based, GT-ITM latency	Event queue	600 nodes	Java	no
GPS [201]	Flow-based, GT-ITM latency	Event queue	1054 nodes	Java	yes
p-sim [120]	Latency (GT-ITM)	Event queue	2048 nodes	C	no
P2PSim[1]	Latency (GT-ITM, G2 graph, Euclidean, end-to-end time graph, random)	Event queue	3000 nodes	C++	no
PlanetSim [107]	Latency (GT-ITM, Brite)	Cycle-based	10^5 nodes	Java	yes
PeerSim[2]	Latency (end-to-end time graph, random)	Cycle-based, Event queue	10^6 (cycle-based), 10^5 (event-based)	Java	no

[1]http://pdos.csail.mit.edu/p2psim/
[2]http://peersim.sourceforge.net/

Table 5.2: Peer-to-peer system simulators.

simulator proposed by He et al. [74], model network communication latency using Internet topology models and generators, such as the Georgia Tech Internetwork Topology Model (GT-ITM) [203, 29, 202], Boston University Representative Internet Topology Generator (BRITE) [119], and Global Network Positioning (GNP) [136]. Furthermore, simulators such as Narses [63] and the General Peer-to-Peer Simulator (GPS) [201] model network bandwidth constraints and simulate data flows based on connection latencies and throughputs. Table 5.2 shows a summary of a few well-known sequential P2P simulators.

Simulators that execute on multiple machines (e.g., a computer cluster) can be considered a separate category. This category contains in particular Parallel Discrete Event Simulators (PDES) [58, 122], which are based on the same system model as DES, but process system events in parallel threads in order to shorten the overall execution time. In principle, PDES produce the same (or close to) simulation results as sequential DES.

Another approach is taken in simulators such as ModelNet [189], which provide interfaces to the evaluated application at the operating system level, simulating an Internet-like environment. ModelNet allows thus a very accurate system performance evaluation, but also requires a significant implementation cost, which is often comparable to a complete system development.

Finally, WiDS [104] is a toolkit that allows a protocol simulation in both the DES or PDES mode and a real-network deployment. Once a distributed protocol is developed using WiDS interfaces, it can be simulated within a single address space on a single machine, simulated on a cluster of machines, or deployed and run in the target environment.

5.2.2 Simulation Requirements

Gradient topologies and the algorithms described in this thesis have been specifically designed to achieve a high scalability and resilience in the presence of churn and communication failures. Therefore, the evaluation of gradient topologies must be performed in a setup that involves a large-scale, dynamic and heterogeneous peer population and unreliable communication.

At the same time, it can be argued that a detailed model of the low-level network is not necessary to perform experiments on gradient topologies. First of all, gradient topologies and super-peer election algorithms discussed in this thesis are based on peer characteristics, such as peer utility, and do not exploit the properties of peer connections. Low-level network concerns play a minor role in the construction of gradient topologies. Furthermore, since nearly all algorithms described in this thesis are periodic (with the period length of a few seconds) and generate relatively small messages (approximately a kilobyte or less per message), they are not likely to congest network connections. Higher-level applications running over gradient topologies, which may use network resources more extensively, are not evaluated in this thesis. Thus, it can be expected the system behaviour can be adequately analysed without considering peer connections' bandwidth, cross traffic and in-network queueing. A simplified network model also enables running experiments on larger-scale, and hence more realistic, P2P systems.

Moreover, the evaluated algorithms are not sensitive to message latency, as they generally require that messages are delivered and responses are returned within each algorithm cycle. Since the cycle lengths are on the order to seconds, while a typical message round-trip time (RTT) on the Internet is approximately 100-200 milliseconds, message latency can be assumed insignificant in the evaluated protocols.

Table 5.3 summarises a number of measurements on the average RTT between machines connected to the Internet. The results vary due to differences in the applied measurement methodologies. Some values in the table are approximated based on graphs, and hence may contain minor estimation errors. In the cases where multiple results are available in one publication, the average RTT is put into the table. The average RTT over all measurements is 169 milliseconds.

Most measurements indicate that the distributions of message RTT are skewed, and relatively small fractions of messages are significantly delayed, even by many seconds. However, peers can deal with such extremely delayed (or lost) messages by estimating the average RTT to their neighbours. When a message is not responded to within a certain timeout, such as three times the RTT, it is assumed to have been lost. This way, peers can decide whether their message have been delivered or lost in a relatively short time (below a second), which is significantly shorter than the simulation time step.

Measurement	Year	RTT (ms)	System
Chen [35]	2005	157	Skype
Wang [191]	2005[2]	81	Internet
Kutzner [95]	2004	410[1]	Overnet
Shakkottai [174]	2002-03	180	Internet
Lakshminarayanan [97]	2002	80[1]	Internet
Fraleigh [57]	2001-02	130	Internet
Saroiu [167]	2001	100[1]	Gnutella
Acharya [3]	1996	100[1]	Internet
Fasbender [55]	1994-95	331	Internet
Bolot [22]	92	200	Internet
Sanghi [166]	92	85	Internet

[1]median value [2]publication date

Table 5.3: Average round-trip times between machines on the Internet.

5.2.3 PlanetLab

An alternative approach to evaluating the performance of a P2P system is to implement it and deploy it on PlanetLab [38, 178], a wide-area test bed for distributed systems. PlanetLab consists of a collection of machines (914 hosts at 473 sites at the moment of writing), physically distributed across the Globe, which are available for running scientific experiments.

However, there are at least three relevant reasons for which gradient topologies and the compared systems have not been evaluated on PlanetLab. First, gradient topologies have been specifically designed for heterogeneous P2P environments, while in PlanetLab, most nodes have similar performance characteristics, since the PlanetLab Consortium specifies a minimum hardware specification for the participating nodes. At the time of writing, the minimum requirement for a node was 3.2 GHz CPU clock speed, 4 GB RAM, and 320 GB disk space. Moreover, most nodes in PlanetLab are hosted by large research institutions and are connected with each other through fast and reliable Internet links, such as university and corporate networks. In order to perform experiments on gradient topologies, node capabilities in PlanetLab would have to be artificially limited, as in a traditional P2P simulator, in order to model a heterogeneous peer population.

Second, most machines in PlanetLab are relatively stable and have high uptimes, since they are used by the PlanetLab community for running long-term experiments. In order to perform experiments with realistic churn conditions, with peers joining and leaving the system according to some distribution, peer life cycle would have to be explicitly modelled, as in a P2P simulator.

Finally, PlanetLab consists of a few hundred machines only, while many existing P2P systems have already reached the scale of millions of nodes [35, 102, 181]. In order to evaluate a P2P system on a realistic scale, each physical node in PlanetLab would have to emulate hundreds of virtual peers.

Measurement	Year	System	Median Session	Distribution
Stutzbach [181]	2003-05	Gnutella	30 min	Log-normal, Weibull
Stutzbach [181]	2003-05	Kademlia	20 min	Log-normal, Weibull
Stutzbach [181]	2003-05	Bittorrent	2-5 min	Weibull, Log-normal
Kutzner [95]	2004	Overnet	6 h	Power-law, Diurnal
Pouwelse [144]	2003-04	Bittorrent	17% $>$ 60 min	Power-law (close to)
Bustamante [27]	2003	Gnutella	2 h	Pareto
Bhagwan [18]	2003	Overnet	60 min	Diurnal
Gummadi [68]	2002	Kazaa	2.4 min	Heavy-tailed
Ripeanu [155]	2000-01	Gnutella	40% $<$ 4 h	—
Chu [36]	2001-02	Gnutella	31% $<$ 10 min	Log-quadratic, Diurnal
Sen [173]	2001	FastTrack	15 min	Power-law, Diurnal
Saroiu [167]	2001	Napster	60 min	Diurnal
Saroiu [167]	2001	Gnutella	60 min	Diurnal

Table 5.4: Median peer session times in P2P systems.

5.2.4 Simulator Overview

At the time when the experiments described in this thesis were implemented, none of the available P2P simulators were suitable for running experiments on gradient topologies, and as a consequence, a custom P2P simulator was developed. The simulator is based on a cycle model, similar to PeerSim, as most of the evaluated algorithms are periodic. The main structure in the simulator is a global loop which controls the flow of time. At each cycle of this loop, every peer in the system executes one cycle of its neighbour selection, aggregation, super-peer election and routing algorithms. The order of peers at each time step is chosen randomly.

5.2.5 Churn Model

The simulator models an open, heterogeneous and dynamic population of peers, with continuous peer arrivals and departures. The churn rate (i.e., the average fraction of peers leaving the system per time unit) in the experiments is carefully chosen to reflect the conditions that could be expected in a real P2P system. Table 5.4 lists a number of independent measurements on peer session times in various P2P systems. Notation $x\% < y$ is used to indicate that x percent of peers in a system have a session time below y. Some values in the table are approximated based on graphs in the original publications. Overall, most studies show that median peer session durations in existing P2P systems are between a few minutes and and a few hours. In order to be consistent with these real-world measurements, the mean peer session time, μ, in most experiments described in this thesis is assumed to be 30 minutes, which corresponds to a churn rate of $\frac{1}{\mu} = 3.3\%$ peers per minute. Assuming a time step of 5 seconds in the simulation, this is equivalent to mean session time of 360 time steps and a churn rate of 0.28%

peers per time step.

While almost all published reports agree that peer session distributions are highly-skewed, there is no general consensus whether these distributions are heavy-tailed and which mathematical model best fits the empirically observed peer session times. Sen at al. [173], Kutzner et al. [95], Bustamante [27], and Pouwelse et al. [144] report power-law (i.e., Pareto) or close to power-law peer session distributions, and Gummadi et al. [68] conclude that session distributions are heavy-tailed (see Table 5.4 for comparison). However, Chu et al. [36] suggest a log-quadratic peer session time distribution, while Stutzbach and Rejaie [181] propose log-normal and Weibull distributions. Saroiu et al. [167], Sen et al. [173], Chu et al. [36], and Bhagwan et al. [18] also observe diurnal patterns in peer session times.

Moreover, Stutzbach and Rejaie estimate that the best power-law fit for the peer session times in a number of BitTorrent overlays has a shape parameter between 2.1 and 2.7 (hence the distribution is not heavy-tailed), and the best Weibull fit has a shape parameter between 0.34 and 0.59. Similarly, Nurmi et al. [137] find that peer session times are best fitted with Weibull distributions with a shape parameter between 0.5 and 0.6.

In this thesis, three session models are used. In the Pareto model, a peer p is assigned a session longer than x with probability

$$P(Ses(p) > x) = (\frac{m}{x})^k. \tag{5.1}$$

The shape parameter, k, is set to 2 (border case between heavy-tailed and non-heavy-tailed distributions) and the minimum session duration, m, is calculated in such a way that the mean session is equal to μ

$$m = \mu\frac{k-1}{k}. \tag{5.2}$$

In the Weibull session model, peer p has a session longer than x with probability

$$P(Ses(p) > x) = 1 - e^{-(\frac{x}{\lambda})^l} \tag{5.3}$$

where l is a shape parameter, $l = 0.5$, and λ is a scale parameter, such that the mean session duration is equal to μ

$$\lambda = \frac{\mu}{\Gamma(1 + \frac{1}{l})} \tag{5.4}$$

and Γ is the Gamma function. In the third session model, used for a theoretical system analysis, peer session times are infinite, and peers can join the system but never leave or fail.

In all session models, joining peers are bootstrapped by a centralised server. The bootstrap server obtains peer addresses through "crawling" the P2P overlay and maintaining a FIFO buffer with 1,000 entries, as described in section 4.8. Additionally, the bootstrap server is used for starting aggregation instances.

Measurement	Year	Message Loss
Wang [191]	2005[1]	0.12%
Heckmann [75]	2004[1]	3.75%
Zhang [205]	1999-01	0.73%
Yajnik [198]	1997-98	1.98%
Boyce [24]	1997-98	7.12%
Borella [23]	1997	1.50%
Moon [128]	1997	6.87%
Paxon [142]	1994-95	3.95%
Fasbender [55]	1994-95	13.03%
Sanghi [166]	1992	4.57%
Bolot [22]	1992	13.5%

[1]publication date

Table 5.5: Average message loss rate on the Internet.

5.2.6 Network Model

The network layer in the simulator is extremely simple. For the reasons explained in section 5.2.2, connection bandwidth and latency are not modelled, and it is assumed that messages are transferred between peers instantly. However, the simulator supports three message loss models.

In the simplest model, each transmitted message has a fixed loss probability P_{loss}. Since gradient topologies do not exploit peer network proximity, and most messages in the evaluated protocols traverse wide-area networks, the message failure probability can be adequately modelled by the overall packet loss rate on the Internet.

Table 5.5 summarises a number of measurements on the message loss rates between hosts on the Internet. In cases where multiple experiments are reported in one publication, the average result is calculated. Heckmann et al. [75] is the most relevant measurement in the context of P2P systems, since it has been performed in e-Donkey, a P2P file-sharing application. The results vary between 0.12% and 13.5%, depending on the measurement methodology, and the average loss rate over all measurements is 5.19%. In order to be consistent with these measurements, message loss probability in the experiments is set to $P_{loss} = 0.05$.

In the second model, the probability of a message loss is proportional to the reciprocal of the recipients session time. A message sent from peer p to peer q fails with probability

$$P(p,q) = \frac{P_{prop}}{Ses(q)} \tag{5.5}$$

where $Ses(q)$ is the total session duration of q (known to the simulator only), and P_{prop} is a system constant scaled in such a way that the average message loss between all peers in the system is equal to P_{loss}. This model is based on observations that peer neighbourhood tables often contain stale entries, due to churn, and large numbers of messages are lost because peers attempt to forward messages using neighbours that no longer exist in the system [30, 113]. Hence, messages are less likely to fail if they are forwarded to more stable neighbours.

The third message loss model is specific to request-response protocols, such as aggregation and neighbour selection. The probability of a request loss is calculated using the fixed-loss of the proportional-loss model, but responses are never lost. The rationale behind this approach is the following. First, it can be assumed that the recipient of a response message, i.e., the requester, is on-line, since the average request-response exchange time is only 100-200 milliseconds, and the probability of the requester leaves during the exchange is negligible. Second, it is estimated that large fractions of peers in P2P systems are connected to the Internet through firewalls or NATs, which significantly contribute to the overall message loss [102, 181]. Typical firewalls allow peers to generate outgoing traffic, but reject traffic initiated by peers outside of the firewalled zone. In the request-response protocols, if a peer has already received a request and generates a response message, it can be assumed that the peer is either not firewalled at all or has a firewall that allows communication with the requester.

5.2.7 Connection Model

It is assumed that all peers are mutually reachable and any pair of peers can potentially establish a connection. In all algorithms, except for Newscast, the neighbourhood model is symmetric, as discussed in section 4.3.1. Each peer p is assigned a capacity value, $C(p)$, and the maximum number of connections peer p can establish at a time is limited to $150 + C(p)$. Capacity values follow a Pareto distribution with the shape parameter $k_c = 2$ (again, border case between heavy-tailed and non-heavy-tailed distributions) and a mean of $\mu_c = 10$. This way, an average peer can have up to 160 neighbours. However, as shown later, peers rarely approach this limit and in most cases have approximately 50 neighbours. The highest utility peers in the simulated systems maintain approximately a few hundred neighbours. For a comparison, in e-Donkey, a P2P file-sharing system, peers have on average about 30-50 connections, and servers (i.e., super-peers) have on average 700 connections [75]. In Gnutella, an ultrapeer can accept up to 32 connections from leaves and up to 30 connections from other ultrapeers [103].

The neighbour verification algorithm and the leave procedure, described in section 4.3.3, are not executed by peers. The reason is twofold. First, it is very hard for the experiment designer to determine the fraction of peers that perform the leave procedure when departing from the system, without a detailed knowledge of the system deployment environment, application scenario and user community. In particular, no study on peer crashes (as opposed to intentional departures) in P2P systems is known to the authors of this thesis. Second, a simplified model of peer connections significantly improves the performance and scalability of experiments, allowing larger networks to be simulated.

For these reasons, the simulator assumes that peers always have consistent views of their con-

Parameter	Symbol	Value
Number of peers in the system	N	100,000
Mean peer session time	μ	360 steps
Session times Pareto shape	k	2.0
Session times Weibull shape	l	0.5
Message loss probability	P_{loss}	0.05
Peer capacity mean	μ_c	10
Peer capacity Pareto shape	k_c	2.0
Maximum number of neighbours	$\mathcal{N}^*$	150+$C(p)$
Aggregation instance interval	F	50
Aggregation instance duration	TTL	60
Aggregation fan-out	G	4
Histogram resolution	b	200
Gradient topology branching factor	B	10
Gradient search time-to-live		30
Utility successors		5
Utility predecessors		3
Random set size		10
Newscast view size		20
SG-1 newscast set size		30
Chord successors		3
Chord predecessors		3

Table 5.6: System parameters.

nections. However, outdated neighbour entries, and the resulting message loss, are simulated in the proportional message loss model described above in section 5.2.6.

In order to establish a connection, peers exchange two messages. If the exchange fails, the connection is unsuccessful. Similarly, peers exchange two messages when they disconnect. Connection closing always succeeds, as it can be assumed that the neighbour verification algorithm eventually resolves any discrepancies between neighbouring peers.

5.2.8 Scalability

The simulator has been implemented in Java using RePast [43], a multi-agent simulation toolkit, and Colt[1], a high performance scientific and technical computing library. For performance and scalability reasons, the simulator uses regular arrays instead of Java collections, primitive data types (such as int, double, etc.) instead of object types (such as Integer, Double, etc.) for storing numbers, and the Colt's pseudo-random number generator instead of the default generator in Java. A number of further optimisations have been performed using a performance profiler, the Java Runtime Analysis

[1]`http://acs.lbl.gov/~hoschek/colt/`

Toolkit[2] (JRat), which allow the simulator to support up to 100,000 gradient topology nodes using 2GB of memory. The simulator can also generate system visualisations, which are especially useful for preliminary system analysis and debugging at early stages of development.

Table 5.6 summarises the main parameters used in the experiments. The parameters at the top of the table, i.e., churn rate, session time distribution, message loss rate, and maximum number of connections per peer, define the simulated systems' environment, and are initialised based on measurements on existing P2P systems, as explained in sections 5.2.5, 5.2.6, and 5.2.7. The remaining parameters are configurable algorithm properties, and their impact on the system performance is discussed further in this chapter. In particular, section 5.4.6 covers the aggregation settings (F, TTL, G, and b), section 5.4.1 discusses the gradient topology branching factor and neighbourhood set sizes, and section 5.4.7 evaluates the impact of the branching factor and message time-to-live on routing performance. The parameters for the state-of-the-art systems (Newscast, SG-1, and Chord) are set based on the original code and publications describing these systems.

5.3 Performance Comparison

This section describes a performance comparison between aggregation-based super-peer election algorithms, used in gradient topologies, and selected state-of-the-art systems, SG-1, SPChord, Hierarchical DHT (abbreviated to H-DHT), and SOLE, conducted using the P2P simulator described in the previous section.

5.3.1 Compared Systems

For SG-1, the original code has been obtained from the PeerSim's website[3] and ported to the P2P simulator used in this study. Moreover, an extended version of SG-1, labelled SG-1-fix, based on the algorithm proposed in section 2.5.1.3, has been implemented and compared with the other systems.

SOLE, H-DHT, and SPChord have been implemented on top of a custom-built Chord implementation. In the H-DHT, peers are randomly assigned to static groups, and each group elects one super-peer only. In SPChord, peer identifiers are chosen randomly and do not correspond to peer locations in a virtual coordinate space. Two version of SPChord are considered, with and without the adjustment algorithm, where the latter is labelled SPChord*.

The systems are compared with a tree-based gradient topology (labelled TGT), generated using tree sets, described in section 4.3.5.3, and Newscast sets, covered in section 2.5.1.1. Ranks are estimated using the mixed method (section 4.6.3), and super-peers are elected using top-K and clients thresholds (section 3.1.1), depending on the experiment.

[2]`http://jrat.sourceforge.net/`
[3]`http://peersim.sourceforge.net/`

Additionally, a variant of TGTs is considered, labelled TGT*, where the random neighbourhood sets, described in section 4.3.4, are used instead of Newscast sets. TGTs are also compared with simpler gradient topologies, described in [163, 164], generated using the utility successor, predecessor, and Newscast neighbourhood sets. These topologies, labelled GT, do not have the tree sets, and hence do not guarantee logarithmic routing performance.

5.3.2 Comparison Criteria

The evaluated systems are compared with respect to the quality of the super-peer sets they generate and the overall cost of the super-peer topology maintenance. The comparison is divided into three sections. In the first section, the goal of each system is to maximise the super-peer capacity, and the systems are compared based on the numbers of elected super-peers, average super-peer capacity, and fraction of optimal super-peers (defined by peer capacity values). In the second section, super-peers are elected based on their stability, and the systems are compared with respect to the average super-peer session duration, super-peer leave rate, and the number of switches between super-peers and clients. Finally, the last section evaluates the super-peer set election and maintenance cost, measured as the average number of neighbours maintained by peers and the number of messages generated by peers. The exact definitions of the comparison metrics are formally introduced in the sections below.

The systems are compared by running a series of simulation experiments. In each experiment, an initial network consisting of a single peer is gradually expanded by adding 0.5% peers at each time step until the system size grows to $N = 100,000$ peers. At subsequent time steps, the system is still under continuous churn, as peers leave according to their session times and are replaced by new peers, but the rates of peer arrivals and departures are equal and the system size remains constant. The system is run for $T = 1000$ additional time steps, during which measurements are performed and various statistical data are collected. The obtained samples are averaged at the end of each experiment.

The experiment is repeated for each of the evaluated systems, and the results are summarised using graphs. The error bars on the graphs indicate the standard deviation in the measurements. In each experiment, all system parameters are initialised with values shown in Table 5.6, unless the experiment description explicitly states that a different setup is used.

5.3.3 Capacity Maximisation

In the first set of experiments, described in this section, super-peers are elected based on their capacity, and the goal of the election algorithm is to maximise the super-peer capacity.

In SOLE, H-DHT, and TGT with a top-K threshold, the desired number of super-peers is given as a system parameter $K = 1000$. Given the total system size of $N = 100,000$ peers, the optimum super-peer ratio is $\frac{1}{100}$, which roughly corresponds to typical super-peer ratios observed in existing

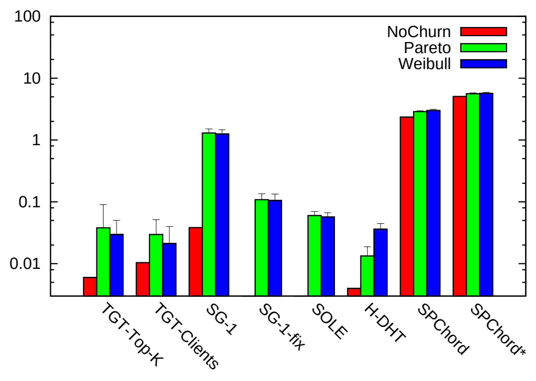


Figure 5.1: Relative error in the number of elected super-peers.

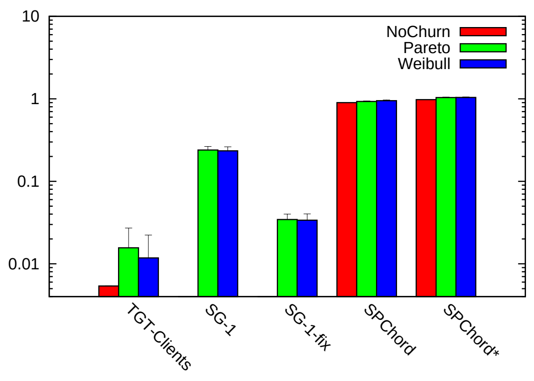


Figure 5.2: Relative error in the super-peer set capacity.

P2P systems, such as 30,000 super-peers for 3 million peers in KaZaA according to Liang et al. [102], approximately 1 super-peer per 30-65 clients in KaZaA according to Xiao et al. [196], and 1 super-peer per 32 clients in Gnutella version 0.6 [103].

In SG-1, SPChord, and TGT with a clients threshold, super-peers are elected in such a way that the super-peer set has a capacity equal to or higher than the total number of clients in the system. The capacity values are assigned in such a way, that the optimum super-peer ratio is approximately equal to $\frac{1}{100}$. More precisely, given that peer capacity values follow a Pareto distribution with shape parameter of $k = 2$ and mean $\mu_c = 10$, from Theorem 3.1, the average capacity of the 1000 highest-capacity peers in a 100,000 peer system is 100,000. The exact optimum size for the super-peer set changes with time and depends on the current peer capacity values.

5.3.3.1 Super-Peer Election Error

Given the optimum number of super-peers, M_t^*, in a simulated system at time t, and the current number of super-peers at time t, denoted M_t, the average relative error in the number of super-peers over all time steps is defined as

$$Err_s = \frac{1}{T} \sum_{t=1}^{T} \frac{|M_t - M_t^*|}{M_t^*} \tag{5.6}$$

where T is the measurement duration. Similarly, the average relative error in super-peer capacity is defined as

$$Err_c = \frac{1}{T} \sum_{t=1}^{T} \frac{|C_t - C_t^*|}{C_t^*} \tag{5.7}$$

where C_t is the total super-peer capacity at time t, and C_t^* is the optimum super-peer capacity at time t.

Figure 5.1 shows the value of Err_s for each of the compared systems. Clearly, SPChord performs worse than the other systems (note the logarithmic scale), both in conditions with churn and without

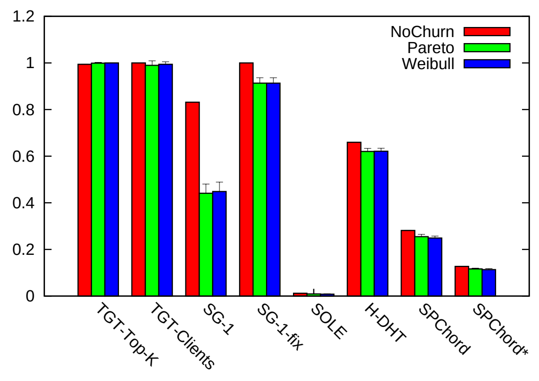


Figure 5.3: Fraction of optimal super-peers.

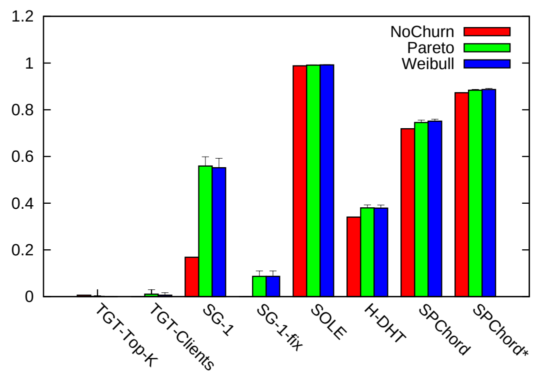


Figure 5.4: Fraction of suboptimal super-peers.

churn. This is caused by the DHT constraints imposed on super-peers in SPChord, which prevent the system from reducing the size of the super-peer set to the required minimum. The topology adjustment algorithm significantly reduces the number of redundant super-peers in SPChord, reducing Err_s by a half, but is not sufficient to outperform the other systems.

Similarly, SG-1 elects too many super-peers in the presence of churn, since every joining peer, and every client that becomes disconnected from a leaving super-peer, becomes a super-peer. This problem is greatly reduced in the extended version of the protocol (SG-1-fix), where peers always attempt to discover super-peers using the *underloaded* sets and connect as clients before deciding to become super-peers. In the absence of churn, both SG-1 and SG-1-fix manage to elect a nearly optimum number of super-peers. All the remaining systems perform relatively well, and the super-peer election error is below a few percent in experiments with churn.

Figure 5.2 shows the value of Err_c for SG-1, SG-1-fix, SPChord, and TGT with a clients threshold. Again, SPChord and SG-1 generate super-peer sets with a significantly higher capacity than the system optimum. However, in both systems, the relative error in super-peer capacity (Err_c) is about 5-6 times lower that the error in the number of super-peers (Err_s), since the low-capacity super-peers that are elected in both systems significantly increase Err_s, but have a relatively low impact on Err_c. In the gradient topology, the average super-peer capacity error is below 2%.

5.3.3.2 Super-Peer Optimality

At each time step t, an optimal super-peer set, S_t^*, is defined based on the current peer capacity values in the system. In SOLE, H-DHT, and TGT with a top-K threshold, this set contains the K highest-capacity peers. In SG-1, SPChord, and TGT with a clients threshold, the optimum set contains the minimum number of highest capacity-peer that have a total capacity equal to or higher than the number of clients.

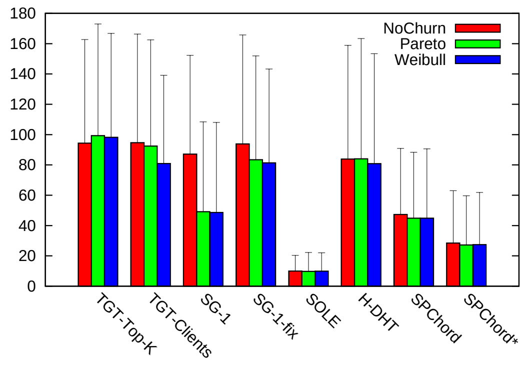


Figure 5.5: Average super-peer capacity.

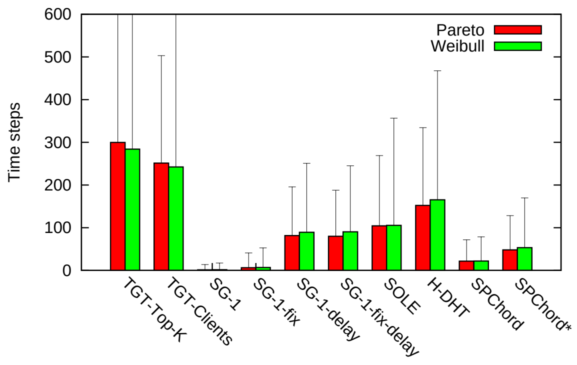


Figure 5.6: Average super-peer session duration.

Given the current super-peer set, S_t, at time t, the average fraction of optimal super-peers is defined as

$$Opt = \frac{1}{T}\sum_{t=1}^{T}\frac{|S_t \cap S_t^*|}{|S_t|}. \tag{5.8}$$

Similarly, the average fraction of suboptimal super-peers in the elected sets is defined as

$$Sub = \frac{1}{T}\sum_{t=1}^{T}\frac{|S_t \setminus S_t^*|}{|S_t|}. \tag{5.9}$$

Figures 5.3 and 5.4 show the values of *Opt* and *Sub* in each of the compared systems. As expected, the gradient topology performs very well, since in threshold-based elections, by definition, super-peer sets contain the highest-capacity peers only.

In SG-1, even in the absence churn, optimum super-peer sets are not achieved. This is caused by the lack of swapping between super-peers and clients in the SG-1 protocol when super-peers are fully utilised, as discussed in section 2.5.1.3. If churn is present, SG-1 elects a large number of suboptimal super-peers, since all clients that are not associated with super-peers are automatically promoted to super-peers, decreasing the value of *Opt* and increasing *Sub*. The results are significantly improved in the extended version of the SG-1 protocol, which maintains close-to-optimum super-peer sets in experiments with churn and fully-optimised sets in the absence of churn.

SOLE generates mainly suboptimal super-peer sets, as it elects super-peers based on peer DHT identifiers and does not take into account peer capacity. Similarly, in SPChord, the generated super-peer sets are largely suboptimal, since the election is based not only on peer capacity values but also on peer positions in the DHT overlay, as previously discussed. The topology adjustment algorithm in SPChord clearly improves the quality of super-peers.

In H-DHT, the fraction of optimum super-peers does not change significantly between the experiments and is close to 60% in both experiments with churn and without churn. This stems from the fact that each super-peer in H-DHT is elected independently in its peer group, and peers located in

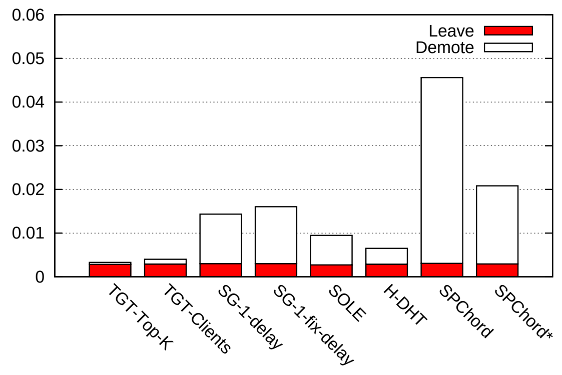


Figure 5.7: Super-peer leave and demote rates.

different groups are not compared. This simplifies the election process, but does not produce optimum super-peer sets at the global system level.

Figure 5.5 shows the average super-peer capacity in the compared systems. The standard error in the results is high, due to the skewed distribution of peer capacity. In particular, Pareto distributions with a shape of $k = 2$ have an unbounded variance. In order to reduce the standard error and increase the experiment repeatability, the maximum capacity in all experiments is limited to 500, i.e., 50 times the mean.

The results shown in Figure 5.5 are consistent with the super-peer optimality shown in Figure 5.3. Systems with higher fractions of optimum super-peers have also a higher average super-peer capacity.

5.3.3.3 Super-Peer Sessions

In order to get more insight into the election algorithms, this section analyses the super-peer session lengths in the evaluated systems. A super-peer session starts when a peer is promoted to a super-peer, and ends either when a super-peer leaves the system or is demoted to a client. Thus, the average super-peer leave rate is defined as

$$L = \frac{1}{T} \sum_{t=1}^{T} \frac{L_t}{M_t} \tag{5.10}$$

where L_t is the number of super-peers that leave the system at time t, and M_t is the total number of super-peers at time t. Similarly, the average super-peer demote rate is defined as

$$Dem = \frac{1}{T} \sum_{t=1}^{T} \frac{Dem_t}{M_t} \tag{5.11}$$

where Dem_t is the number of super-peers demoted to clients at time step t.

Figure 5.6 shows the average durations of super-peer sessions in the compared systems. Similar to Figure 5.5, the standard error is high due to the skewed peer session distribution. In SG-1, super-peer sessions are extremely short, since all peers join the system as super-peers, and are in most cases demoted to clients within just few time steps. In SG-1-fix, super-peer session are longer, since peers can join the system as clients, but the average super-peer session is still short compared with the other systems. This results from two facts. First, joining peers, as well as clients that are disconnected from super-peers, are often unable to discover new super-peers, since the *underloaded* neighbour sets often contain outdated peer entries, i.e., peers that do not have any free capacity, are not super-peer anymore, or have already left the system. Moreover, super-peer sessions are frequently terminated when super-peers are swapped and replaced with higher-capacity clients.

The SG-1 paper [124] suggests that super-peer stability can be improved by allowing peers to change their status in an initial period, when joining the system or searching for a super-peer, and assuming that a super-peer is created only if the peer's status does not change for a fixed number of algorithm rounds, t_s. This approach is evaluated and compared with the other systems in Figure 5.6. It is labelled SG-1-delay for the original SG-1 algorithm and SG-1-fix-delay for the extended SG-1 protocol, and the t_s parameter is set to 10. As expected, the average super-peer session duration is significantly increased in the new measurement method.

Overall, SOLE, H-DHT, SPChord and SG-1 with delayed super-peer election have comparable super-peer session durations, but are all outperformed by gradient topologies. This is further analysed in Figure 5.7, which shows the average super-peer leave rate and demote rate in the evaluated systems. The super-peer leave rate is nearly identical in all systems, and is equal to the overall system churn rate, since super-peers are elected exclusively based on their capacity in the considered experiments. However, the frequency of swappings between super-peers and clients (i.e., super-peer demote rate) varies considerably between the systems. SG-1 and SG-1-fix are not shown in Figure 5.7 for clarity reasons, since the super-peer demote rates in these systems are more than an order of magnitude higher than the corresponding rates in the other systems. The gradient topologies show a significant advantage over the other systems as they have the lowest rate of switches between super-peers and clients. The figure also shows the cost of the topology adjustment algorithm in SPChord. The average rate of super-peer swapping in SPChord is twice higher compared to SPChord*. As a consequence, super-peers in SPChord have twice shorter sessions compared to SPChord*.

5.3.4 Stability Maximisation

This section describes a series of experiments that evaluate the ability of the compared systems to maximise the stability of super-peers. To that end, each system is configured in such a way that super-peers are elected based on their session characteristics rather than their capacity, as in the previous section.

For each system, three super-peer election criteria are considered: uptime, expected session duration, and expected remaining session duration. The calculation of a peer's uptime is straight-forward. For the estimation of the peer's session, three models are used. In the first model, a peer simply

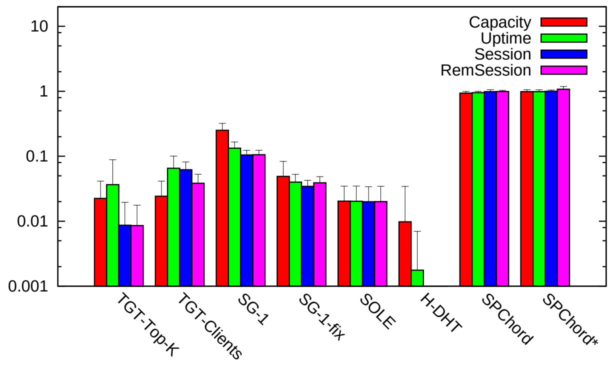


Figure 5.8: Super-peer election error.

obtains the exact session duration from the simulator, which corresponds to a situation where a peer can predict its session length with a 100% certainty. In the second model, labelled “Error1” on the graphs, a peer p calculates its expected session duration as

$$Ses'(p) = (1 + \epsilon_p)Ses(p) \tag{5.12}$$

where $Ses(p)$ is the true session length of p, known by the simulator only, and ϵ_p is a parameter randomly initialised by p at startup which follows a uniform distribution in the range from -1 to $+1$. Thus, $Ses'(p)$ is assigned randomly between 0 and $2Ses(p)$. In the third model, denoted “Error2”, peer p calculates its expected session duration using formula

$$Ses'(p) = \frac{1}{2}Ses(p) + \frac{1}{2}Ses(x) \tag{5.13}$$

where $Ses(x)$ is a variable initialised by p which follows the same distribution as peer sessions in the system, i.e., Pareto or Weibull, depending on the experiment. The remaining peer session is calculated as the estimated session duration, Ses', minus the current peer uptime.

In order to obtain accurate measurement results, due to the high variance of peer session times (in Pareto distributions with a shape parameter of 2, the variance and standard deviation are infinite), each experiment is run for 10,000 time steps. However, due to the increased computational cost, the number of peers in each experiment is reduced to 10,000. As previously, in SOLE, H-DHT, and TGT with a top-K threshold, the desired super-peer ratio is $\frac{1}{100}$ (i.e., the optimum number of super-peers is $K = 100$), and in SG-1, SPChord, and TGT with a clients threshold, the super-peer set is created in such a way that its total capacity is equal to or higher than the total number of clients in the system. Moreover, in the gradient topology, the election strategy described in section 4.5.3 is followed, where super-peers are never demoted to clients.

Figure 5.8 shows the relative error in the number of elected super-peers (i.e., Err_s, as defined above) in SOLE, H-DHT, and TGT, and the relative error in the super-peer capacity (Err_c) in SG-1, SPChord, and TGT. The experiments show that the choice of the super-peer election criteria has a

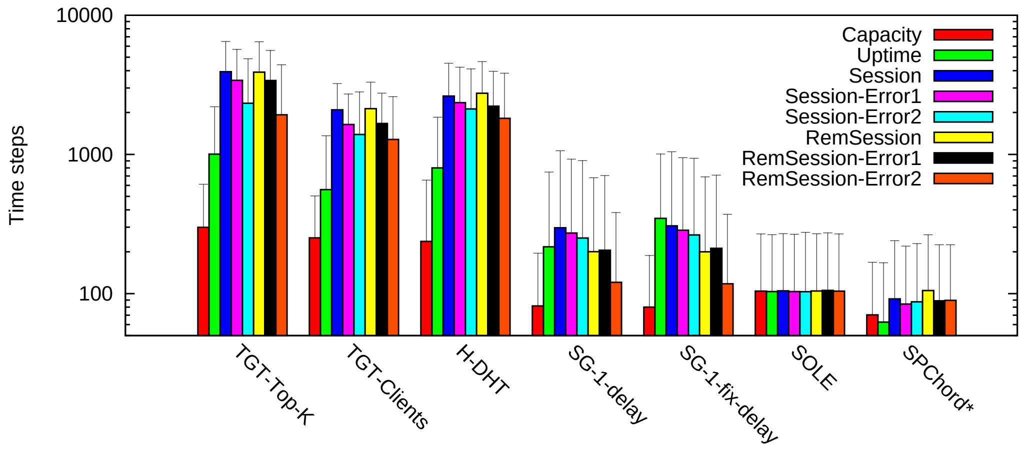


Figure 5.9: Average super-peer session duration.

minor impact on the election error for most algorithms, and the results are generally consistent with the previous measurements in section 5.3.3.1. In H-DHT, TGT, and SG-1, a better performance is observed with the uptime-based and session-based super-peer election criteria, compared with the capacity-based election, since with the former criteria the super-peer sets are more stable and hence easier to manage.

Figure 5.9 shows the average super-peer session duration in the compared systems. Peer sessions are modelled using a Pareto distribution, but similar results are obtained with Weibull distributions. In all considered scenarios, TGT and H-DHT clearly outperform the other systems, achieving several times longer average super-peer sessions. In SOLE, SPChord, and SG-1 (all versions), the choice of super-peer election criteria has little impact on the super-peer stability, since in these systems the super-peer sessions durations are mainly determined by the swappings between super-peers and clients. In all examined scenarios, the topology adjustment algorithm in SPChord only reduces the average super-peer session duration. Moreover, in SOLE, and to a large extent in SPChord, super-peers are elected based on their position in the DHT overlay rather than their estimated stability, resulting in short super-peer sessions.

In the gradient topology, top-K threshold appears to produce longer super-peer sessions than the clients threshold. This results from the fact that the latter threshold elects more super-peers. When peer utility is defined as capacity, the optimum super-peer ratio is approximately $\frac{1}{100}$, both in systems with top-K and clients thresholds. As the utility function is changed from capacity to uptime, session, or remaining session, the clients threshold requires a selection of more super-peers in order to achieve a sufficient capacity to handle all clients in the system.

As expected, in all systems, the stability of super-peers is lowest when the election is based on peer capacity. Uptime-based super-peer election comes second, in terms of super-peer stability, and

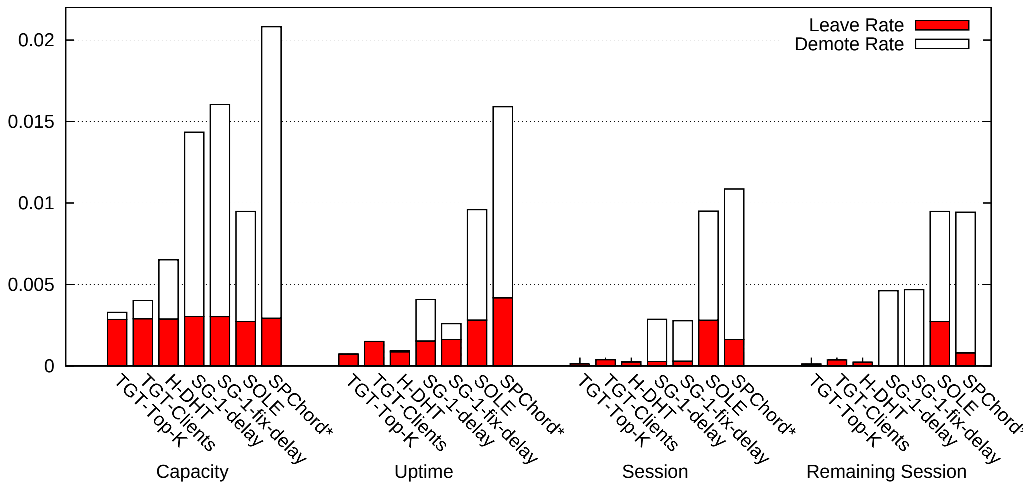


Figure 5.10: Super-peer leave and demote rates.

the best results are obtained with session and remaining session based election, with little difference between the last two methods.

The session and remaining session based election methods outperform the uptime-based and capacity-based methods also in the experiments with peer session estimation error. Furthermore, super-peer sessions are longer in experiments with the first error model (Error1) compared with the second model (Error2). This can be explained in the following way. In the first model, if a peer p has a short session $Ses(p)$, then according to Formula 5.12, it must also have a low session estimation $Ses'(p)$, and hence is not likely to become a super-peer. In the second model, all peers in the system, including those with the shortest sessions, have a non-zero chance of becoming super-peers, since $Ses'(p)$ is unbounded in Figure 5.13.

Figure 5.10 shows the super-peer leave rates and demote rates in the same set of experiments. As previously, in the capacity-based super-peer election, the super-peer leave rate is approximately equal for all systems. However, in experiments with the other super-peer election criteria (i.e., uptime, session, and remaining session), the super-peer leave rate is significantly lower, due to the selection of more stable super-peers.

In SOLE, as previously, the super-peer leave and demote rates are exactly the same in all experiments, since the election mechanism is based on peer DHT identifiers. Similarly, in SPChord, only a small reduction in the super-peer leave and demote rates are observed in the performed experiments, as the system elects large numbers of suboptimal super-peers (i.e., low stability in this case).

In contrast, in TGT, H-DHT, and SG-1, the super-peer leave rate is greatly reduced in experiments where super-peers are elected based on their expected session or remaining session, compared with the capacity and uptime experiments. However, SG-1 suffers from a relatively high frequency of

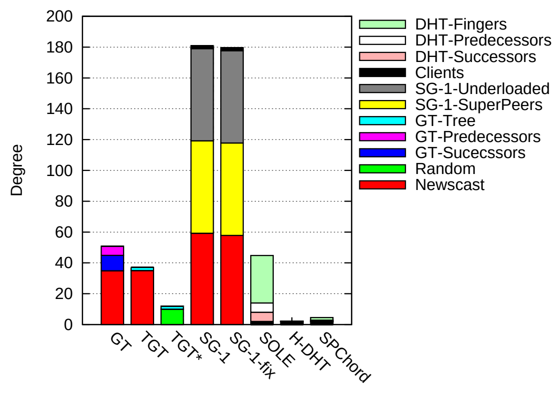


Figure 5.11: Average peer degree.

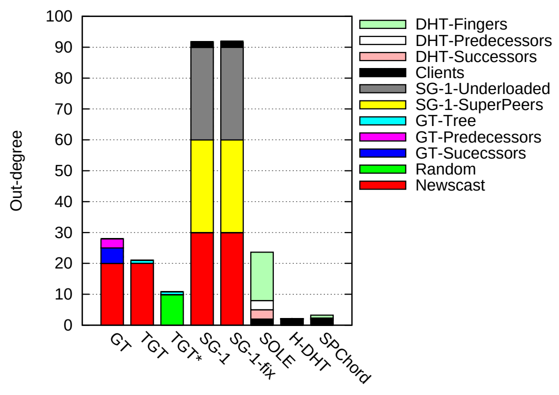


Figure 5.12: Average peer out-degree.

super-peer demotions, which overall causes that the average super-peer sessions in SG-1 are short. In particular, in SG-1 where super-peers are elected based on their remaining sessions, super-peer leave rate is almost zero, since nearly all super-peers are swapped with clients before they leave the system (i.e., when their remaining session times are low). TGT and H-DHT generally outperform all other systems, as they manage to minimise both the super-peer leave rates and demote rates.

5.3.5 Cost

The following section compares the cost of super-peer election in the evaluated systems. This cost is measured as the peer degrees (i.e., the number of neighbours maintained by peers) and the number of messages generated by peers per time step.

5.3.5.1 Peer Degrees

The system topology, T, is an undirected graph (V, E), where V is the set of peers in the system, and E is the set of edges determined by the peer neighbourhood sets. In most algorithms described in this thesis, the neighbourhood model is symmetric, however, some algorithms, such as Newscast, are based on an asymmetric neighbourhood model. In order to introduce a common notation and evaluation metrics for all systems, $(p, q) \in E$ if either $q \in \mathcal{N}_p$ or $p \in \mathcal{N}_q$.

Moreover, for each type of neighbour subsets, S, a sub-topology $T_S = (V, E_S)$ is defined, such that $(p, q) \in E_S$ if $q \in S_p$ or $p \in S_q$, where S_p and S_q are neighbourhood subsets maintained by peers p and q. The degree of a peer p in a sub-topology T_S is defined as the number of edges that connect to peer p in T_S. The out-degree of a peer p in a sub-topology T_S is simply defined as the size of S_p.

Figure 5.11 shows the average peer degree in the system overlay and generated sub-overlays in each of the evaluated systems. SG-1 has a clearly highest (over four times) average peer degree compared with the other systems, which is a consequence of the fact that every peer in SG-1 participates in three Newscast instances, i.e., *connected*, *superpeers*, and *underloaded*. Conversely, in H-DHT and SPChord,

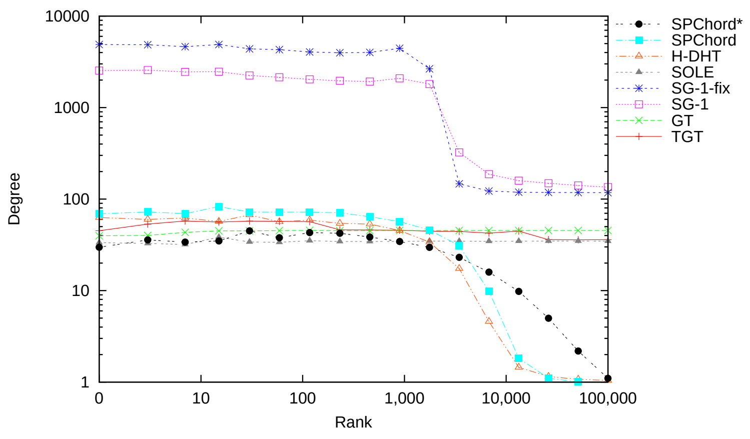


Figure 5.13: Average peer degree versus peer rank.

the average peer degree is many times lower compared with the other systems, since a large majority of peers in these two systems, i.e., all clients, maintain only one connection to a super-peer. This reduces the average number of connections per peer to about two in H-DHT and four in SPChord (almost identical result is obtained for SPChord*). In SOLE, where all peers participate in a global DHT overlay, the overall number of connections is significantly higher (about 40), and is mainly determined by the peer finger tables.

In TGT, an average peer participates in one Newscast overlay, and maintains two connections in the tree set (one outgoing and one incoming). In a Newscast overlay, a peer has on average 20 outgoing connections (30 in the original SG-1 code), and hence the average degree in Newscast is approximately 40 (60 for SG-1). In the presence of churn, the average degree gradually decreases, due to the outdated entries in the Newscast neighbourhood sets.

If the Newscast sets in TGT are replaced with the random sets described in section 4.3.4, the average peer degree is reduced from 37 to 12. This configuration is labelled TGT* in Figure 5.11. The random sets are generally more robust to topology partitioning, and hence require fewer neighbours. Furthermore, if a gradient topology is generated using the successor and predecessor sets instead of the tree sets (which is labelled GT in Figure 5.11), the average peer degree grows by about 10.

Figure 5.12 shows the average peer out-degree. Since the average peer out-degree must be equal to the average peer in-degree in the system, the out-degree distribution is almost identical (scaled by 0.5) to the general peer degree distribution.

Figure 5.13 shows the relationship between peer degree and peer rank. The plot represents a histogram with logarithmically growing bins. Each point (r_i, d_i) in the plot is generated by calculating the average peer degree, d_i, over all peers that belong to the i'th histogram bin, i.e., peers ranked from r_i to r_{i-1}. The histogram is generated periodically during the experiment, and the values in each histogram bin are averaged at the end of the simulation run.

The average peer degree does not exceed approximately 50 in most systems, with SG-1 being the only exception. In SG-1, each client maintains up to 60 connections to super-peers through the *superpeers* and *underloaded* sets, and since the super-peer ratio is approximately $\frac{1}{100}$, the average number of connections per super-peer is approximately 6,000. As the actual super-peer ratio is higher than $\frac{1}{100}$, especially in the original SG-1 algorithm, where a large number of sub-optimal super-peers is elected, the average super-peer degree is close to 2000 is SG-1 and 5000 in SG-1-fix.

In SOLE, the average peer degree does not change with peer rank, since all peers, clients and super-peers, participate in a global DHT. This is different in H-DHT and SPChord, where only the super-peers maintain the DHT, and low-utility peers have a degree close to one, as they are connected to super-peers only. Furthermore, the average degree of low-utility peers in SPChord* is higher compared to SPChord and H-DHT, since SPChord* is more likely to elect low-utility (and hence sub-optimal) super-peers.

In TGT, the average degree of low-utility peers is 36, as these peers participate in one Newscast overlay and have only one tree connection. The average degree increases to 46 for peers ranked 10,000 and less, since each of these peers has 10 incoming tree connections. Moreover, the 100 highest-utility peers have on average 56 connections, since they maintain additional utility successor and predecessor sets for the rank estimation. In GT, all peers have approximately 45 neighbours, as they all participate in a Newscast overlay and have utility successor and predecessor sets.

5.3.5.2 Messages

One of the most commonly used metrics for measuring the cost of a distributed algorithm, or a system overhead, is the number of messages generated by a node per time step. This approach is also taken in this thesis. Figure 5.14 shows the average number of messages generated by each algorithm running at peers in the evaluated systems per time step. Similar to in-degrees and out-degrees, the average number of messages received by peers is equal to the average number of messages sent by peers. Given that each time step in the simulation represents 5 seconds of the real time, Figure 5.14 must be scaled by a factor of 0.2 to obtain the numbers of messages generated by peers per second.

The algorithms considered in the comparison, and the corresponding message types, are neighbour selection (specific to each system), connection handling (i.e., connect and disconnect messages in the symmetric neighbourhood model), and super-peer election, which is again specific to each system, and corresponds to aggregation in the gradient topology, *clients* set maintenance in SG-1, and super-peer discovery using a DHT and client transfer in SOLE, H-DHT, and SPChord.

Similar to peer degree analysed in the previous section, the average number of messages generated by a peer is lowest in H-DHT and SPChord, since the clients in these systems, which constitute a large fraction of all peers, communicate with their super-peer only and generate very few messages. The cost of the DHT maintenance is H-DHT and SPChord is nearly negligible, since the participation in the DHT overlays is restricted to super-peer only. Interestingly, the overall message costs in SPChord

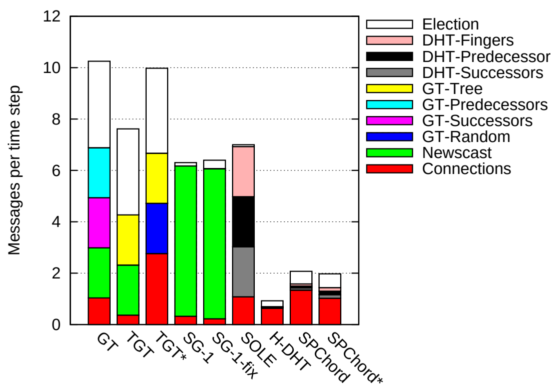


Figure 5.14: Average number of messages generated by a peer per time step.

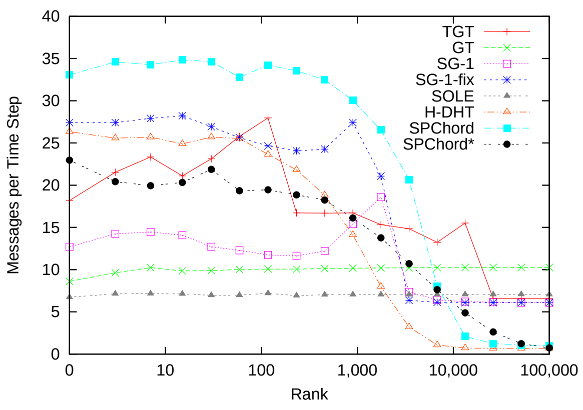


Figure 5.15: Messages generated by a peer per time step as a function of peer rank.

and SPChord* are almost equal. SPChord generates extra messages for the topology adjustment, but elects fewer super-peers and thus reduces the DHT maintenance cost.

SOLE incurs a much higher message cost compared to SPChord and H-DHT, since all peers in SOLE run the DHT overlay. In SG-1, the overall message cost is dominated by the three instances of Newscast run by all peers in the system. Each of these instances requires the exchange of 2 messages per peer per time step.

In TGT, peers generate on average two messages for the maintenance of Newscast sets, two messages for the tree sets, and 3-4 messages for aggregation. In total, 7-8 messages per time step, which is comparable to SG-1 and SOLE, but significantly higher than the message cost in H-DHT and SPChord. If the Newscast sets are replaced by the random sets in a TGT, which is labelled TGT* in Figure 5.14, the overall message cost is significantly increased, since the symmetric connection model used in the random sets requires that peers exchange a pair of messages each time they set up or close a connection. As the connections in the random set are constantly shuffled, the maintenance cost of random sets is significantly higher compared with Newscast (by approximately 2 messages per time step), and for that reason, in the remaining experiments in this thesis, peers use Newscast sets only in the gradient topology.

A GT overlay is more expensive to create and maintain than a TGT, since each peer in a GT participates in three neighbourhood sets, i.e., Newscast, successor, and predecessor, and generates on average two messages per time step for each set. In TGT, only the 100 top-ranked peers construct the successor and predecessor sets, and the cost of their maintenance is negligible compared to the total number of messages generated in the system.

Figure 5.15 shows the relationship between peer rank and the number of generated messages in each of the compared systems. The plots represent rank-based histograms, generated in a similar way as in Figure 5.13. A more detailed analysis of messages generated in each of the evaluated systems is shown in Figure 5.16.

In SG-1, each peer incurs a constant cost of 6 messages per time step, required for the maintenance of three Newscast overlays. Additionally, super-peer gossip and transfer clients between each other. Super-peers with ranks close to 1,000 are most likely to pass their clients to higher-utility super-peers, and hence have the highest connection and client transfer cost.

In SG-1-fix, in addition to the message exchanges required by the original SG-1 protocol, super-peers periodically contact neighbours in their *superpeer* set and potentially swaps them with higher-utility clients. Moreover, clients that join the system or become disconnected from their super-peers, attempt to discover new super-peers using the *underloaded* overlay. Super-peers gossip more often with each other in SG-1-fix compared with SG-1, since the *underloaded* sets in SG-1 contain a high proportion of low-utility and temporary super-peers, and hence, permanent super-peers, ranked below 1,000, are less likely to discover gossip partners.

In SOLE, all peers participate in a global DHT and exchange two messages per time step for the maintenance of their successor, predecessor, and finger sets. The connection maintenance cost is approximately one message per time step.

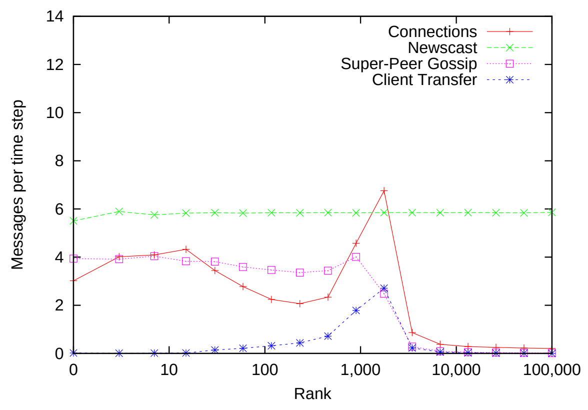
In H-DHT, in contrast to SOLE, only super-peers participate in the DHT and run the successor, predecessor, and finger sets. Additionally, super-peers broadcast client lists to all members in their peer groups at every 10 time steps. Given the super-peer ratio of approximately $\frac{1}{100}$, the average cost of the client list broadcast is approximately 10 messages per super-peer per time step. Due to continuous connections and disconnections of clients, super-peers generate approximately 7 messages per time step for the management of their connections with neighbours.

In SPChord, similar to H-DHT, super-peers maintain the DHT structures, and periodically broadcast client lists to all group members. Additionally, super-peers periodically contact their neighbours in the DHT and occasionally merge or split their clusters or swap positions with higher-uptime clients. Due to the frequent topology modifications, the overall connection handling cost is relatively high.

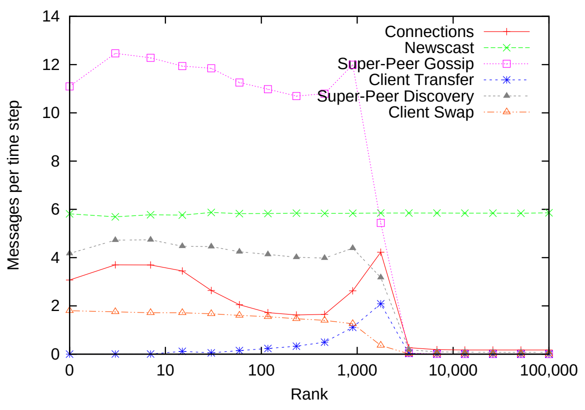
In TGT, every peer participates in a Newscast overlay, which requires an exchange of two messages per time step, and runs the aggregation algorithm, which generates approximately 4 messages per time step. Each peer ranked below 10,000 generates 11 messages per time step for the maintenance of the tree sets, as it receives on average 10 neighbour exchange requests per time step from its children and initiates on average one message exchange with the parent node. Additionally, the 100 top-rank peers maintain utility successor and predecessors sets, for which they exchange 4 messages per time step. Furthermore, peers with ranks close to 100 suffer a high connection handling cost, since they frequently join and leave the successor and predecessor sub-overlays.

In GT, all peers participate in the successor, predecessor and Newscast overlays, and run the aggregation algorithm, and hence, generate approximately 10 messages per time step.

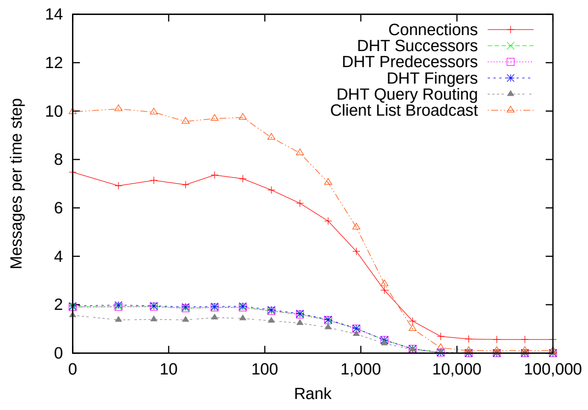
Overall, the average message cost at super-peers varies between 7 and 30 messages per peer per



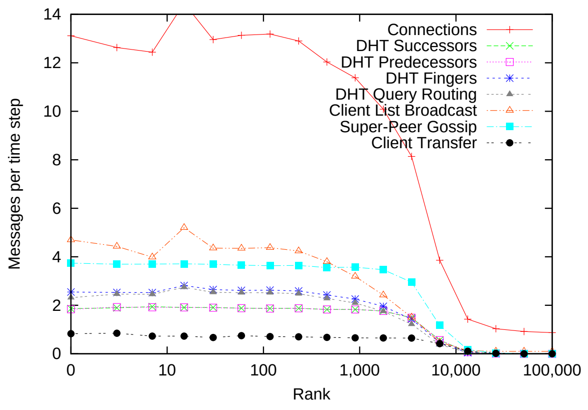
(a) SG-1



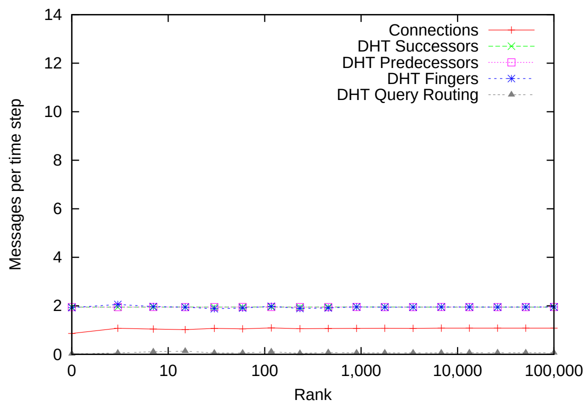
(b) SG-1-fix



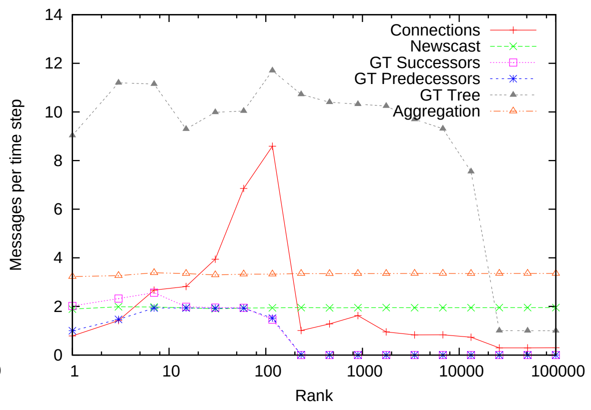
(c) H-DHT



(d) SPChord



(e) SOLE



(f) TGT

Figure 5.16: Messages generated by a peer per time step in the compared systems.

time step, and is comparable in all systems. Moreover, in all systems, super-peers generate more messages per time step than clients, with the exception of SOLE and GT, where the message cost is equal for all peers. H-DHT and SPChord achieve the lowest message cost per peer, since clients in these systems maintain only single connections to their super-peers and rarely generate messages, reducing the average message cost to approximately one message per peer in H-DHT and two messages per peer in SPChord.

5.3.6 Summary

The evaluated systems have been compared in a series of experiments based on three general criteria: super-peer set quality, stability, and cost. In the first set of experiments, described in section 5.3.3, the systems are compared with respect to the average size, capacity, and optimality of elected super-peer sets. The experiments show that only the TGTs (both with top-K and clients thresholds) and systems with simple super-peer criteria (i.e., SOLE and H-DHT, which elect fixed numbers of super-peers) manage to control the number of super-peers in the overlay, while SG-1 and SPChord elect significantly too many super-peers. Moreover, TGTs outperforms all other systems in terms of super-peer optimality, and generate super-peer sets with the highest average capacity.

The second set of experiments, in section 5.3.4, evaluates the systems' ability to maximise the average super-peer session duration. In these experiments, SOLE and SPChord achieve poor results, since they elect low-stability super-peers and frequently swap super-peers with clients. In SG-1 and SG-1-fix, super-peers rarely depart from the system, but the average super-peer session is short due to frequent switches between super-peers and clients. TGT and H-DHT greatly outperform all other systems, as they manage to minimise both the super-peer leave rates and demote rates.

In the last set of experiments, in section 5.3.5, the systems are compared with respect to the maintenance cost, measured as the number of established connections and generated messages. In summary, TGTs are cheaper than SG-1, comparable to SOLE (and hence to Chord), and more expensive than H-DHT and SPChord, in terms of peer connections. Moreover, TGTs are comparable to SG-1 and SOLE in terms of generated messages, and more expensive than H-DHT and SPChord. The cost at super-peers is comparable in all systems, and can be estimated as being between 13 and 27 messages per time step and approximately 50 connections, with the exception of SOLE, where all peers, including super-peers, generate only 7 messages per time step.

Overall, it can be concluded that TGTs elect better-quality and higher-stability super-peer sets, and have similar maintenance cost, compared with the state-of-the-art systems.

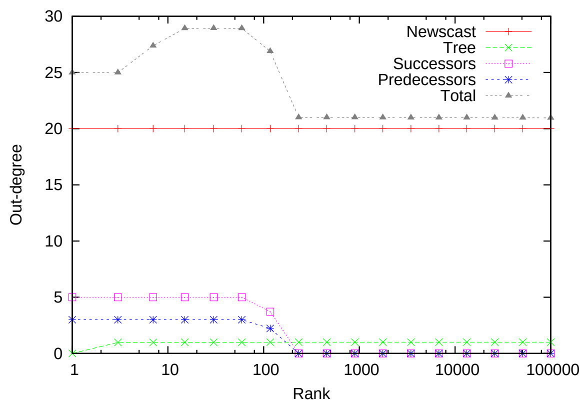


Figure 5.17: Peer out-degree versus peer rank in a TGT.

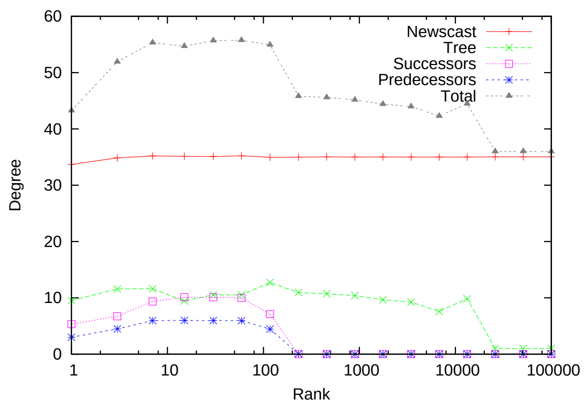


Figure 5.18: Peer degree versus peer rank in a TGT.

5.4 Gradient Topology Analysis

This section provides an in-depth analysis of the gradient topology and the algorithms introduced in chapter 4, based on a series of experiments conducted in the P2P simulator. The main purpose of this section is to verify that the proposed algorithms produce topologies and super-peer sets that have the properties theoretically derived in chapter 3.

The section is organised as follows. The first experiments evaluate the neighbour selection algorithms through an analysis of the generated topologies, where the studied properties include peer degrees, average path lengths, distances to high-utility peers, and parent ranks. The second set of experiments evaluates the accuracy of the aggregation and rank estimation algorithms in a variety of system configurations. The third set of experiments focuses on the performance of gradient search, both in GTs and TGTs, and measures the average number of message hops and failure rates in gradient search. Finally, the last set of experiments examine the behaviour of the super-peer election algorithms in systems where the utility of peers and the total system load are dynamically changed at each time step.

5.4.1 Degree

One of the most basic topology properties is the average peer degree. In the gradient topology, the degree of a peer p is defined as the size of $\mathcal{N}_p$, and the out-degree of peer p is defined as the number of neighbours $q \in \mathcal{N}_p$ such that $Ref_p(q) > 0$. Analogously, peer degrees and out-degrees are defined in sub-overlays generated by individual neighbour selection algorithms running at peers.

Figure 5.17 shows the average peer out-degree in a TGT and its sub-topologies as a function of peer rank. As in the previous experiments, each point plotted in the figure represents one bin of a rank-based histogram. As expected, the figure confirms that every peer has one neighbour in the tree set (i.e., parent), and 20 neighbours in the Newscast set. Additionally, each peer ranked below 100

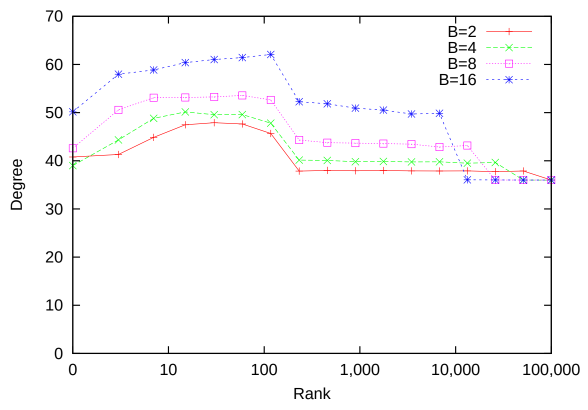


Figure 5.19: Peer degrees in TGTs with different branching factors.

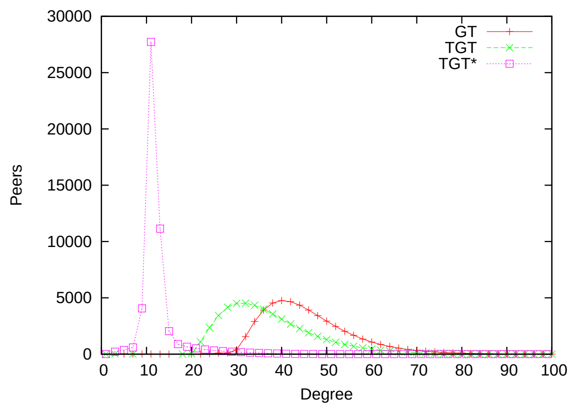


Figure 5.20: Peer degree distribution in three variants of gradient topologies.

has 5 neighbours in its successor set and 3 neighbour in the predecessor set.

Figure 5.18 shows the average peer degree in the same experiment. For most peers, the degree is simply twice as high as the out-degree. However, in the tree sub-topology, peers ranked above approximately 10,000 have no incoming connections (both their degree and out-degree are equal to one), while peers ranked below 10,000 have only one outgoing connection to a parent and 10 incoming connections from lower-rank peers.

Figure 5.19 shows the average peer degree in four different TGTs with branching factors of 2, 4, 8 and 16. In all systems, the degree of high-ranked (i.e., low-utility) peers is approximately 35-40, mainly due to Newscast sets, while for the high-utility peers, the degrees is increased by B incoming tree connections, as expected in a TGT.

Figure 5.20 shows the average peer degree distribution in a GT, and two TGTs with Newscast and random sets (the latter is labelled TGT*). Each plotted point represents the total number of peers in the system that have a given degree. The obtained degree distributions resemble normal distributions, where majority of peers have respectively around 12, 32, and 42 neighbours in TGT*, TGT, and GT. The variance in the degree distribution in TGT* is much smaller compared with GT and TGT, since the random set management algorithm actively removes neighbours when the set size grows above 10 and adds neighbours when the set size is below 10.

Similar peer degree distributions are obtained in TGTs with different utility metrics and branching factors, as shown in Figures 5.21 and 5.22. In all configurations, the fraction of peers that have more than 80 neighbours is marginal. Hence, it can be concluded that the neighbour selection algorithm balances the load between peers and the probability that a peer becomes overloaded by incoming connections is very low.

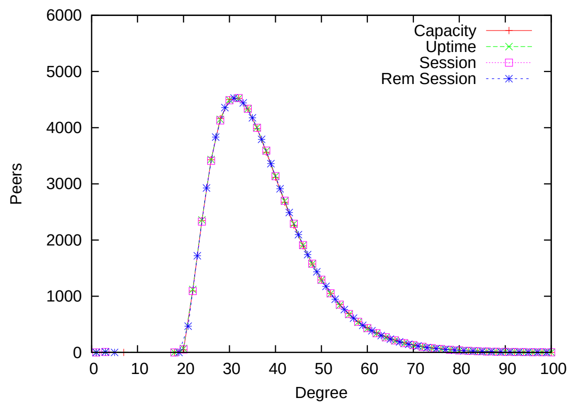


Figure 5.21: Peer degree distribution in TGTs with different utility metrics.

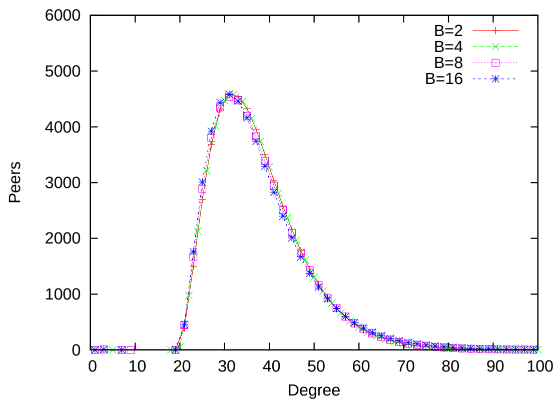


Figure 5.22: Peer degree distribution in TGTs with different branching factors.

5.4.2 Parent Ranks

In an idealised TGT, as defined in chapter 3, a peer p is connected with a parent peer ranked $\lfloor \frac{R(p)}{B} \rfloor$. However, in a dynamic system, it is extremely unlikely that all peers strictly adhere to this rule, and minor deviations are likely to occur in the system topology. In order to measure the extent to which the system topology diverges from the ideal TGT structure, the parent error is defined for each peer p as

$$Err_p(p) = \frac{\left| \lfloor \frac{R(p)}{B} \rfloor - R(par(p)) \right|}{\lfloor \frac{R(p)}{B} \rfloor} \tag{5.14}$$

where $par(p)$ is the current parent peer of p. For peers that do not have any parent, Err_p is defined as one. Figure 5.23 shows the average parent error as a function of peer rank in topologies generated using four different utility metrics. The divergence between $R(par(p))$ and $\lfloor \frac{R(p)}{B} \rfloor$ is partly caused by the peer's inability to discover and stay connected with the desired neighbours in a dynamically changing system, and partly by the inaccuracy of the peer's estimation of its own and its neighbours' ranks. In all performed experiments, Err_p is below 20%. As shown later, this level of Err_p does not have a significant impact on the topology properties.

5.4.3 Path Lengths

In order to get more insight into the structure of the generated topologies, a number of experiments are performed that measure the average path lengths between high utility peers. The following definitions and metrics are used. The system topology, T, is defined as an undirected graph with vertices V and edges E determined by the peer neighbour sets, as previously. $D(p, q)$ is defined as the shortest path length between peers p and q in the system topology T, and analogously, $D_S(p, q)$ is defined as the shortest path length between p and q in a sub-topology T_S. The average path length in the topology

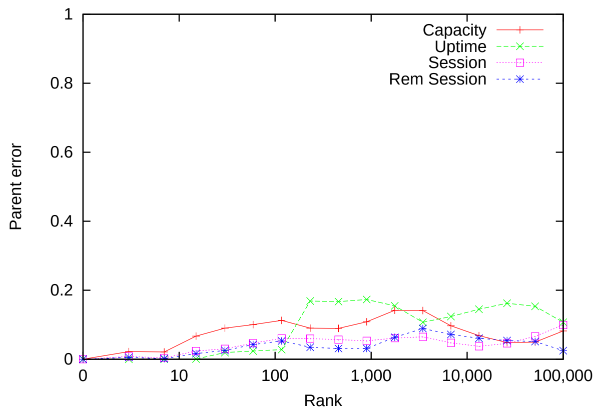


Figure 5.23: Parent error versus peer rank.

T, denoted $Apl(V)$, is the average value of $D(p,q)$ over all possible pairs of peers (p,q)

$$Apl(V) \quad = \quad \frac{\sum_{p,q\in V} D(p,q)}{|V|^2}. \tag{5.15}$$

The average path length $Apl(V)$ can be calculated using the Dijkstra shortest path algorithm at an $O(|V|^2 d)$ cost, where d is the average peer degree in V. However, in the experiments described in thesis, with $|V| = 100,000$ and $d \approx 50$, this would require performing over $100,000,000,000$ basic operations.

This cost can be reduced by selecting a random subset V' from V and approximating $Apl(V)$ with

$$Apl^{'}(V) = \frac{\sum_{p\in V'}\sum_{q\in V} D(p,q)}{|V'|\cdot|V|}. \tag{5.16}$$

Such approximation requires running the Dijkstra algorithm for $|V'|$ peers, and hence, incurs the computational cost of $O(|V'||V|d)$ operations. In practice, $|V'| = 100$ generates accurate results.

In the unlikely case where two peers p and q are not connected in the system topology, the distance $D(p,q)$ is not defined and the (p,q) pair is omitted in the calculation of Apl'. The number of such pairs is extremely low in the reported experiments and such pairs only occur when a peer becomes isolated and needs to be re-bootstrapped. With the exception of isolated peers, topology partitions were never observed in any of the experiments described in this thesis.

In order to investigate the correlation between peer utility and the path lengths in the topology, V_r is defined as a subset of peers in the system, $V_r \subset V$, that contains r highest utility peers. Formally,

$$V_r = \{p \in V \mid R(p) \geq r\}. \tag{5.17}$$

The average path length between the r highest utility peers is then given by $Apl(V_r)$. Similarly, $Apl_S(V_r)$ is defined as the average path length between peers in V_r in a sub-topology T_S.

Figure 5.24 shows the average path lengths between peers in V_r sets in a Newscast overlay and in four TGT sub-overlays generated by tree sets with branching factors of 2, 4, 8 and 16. As expected, the

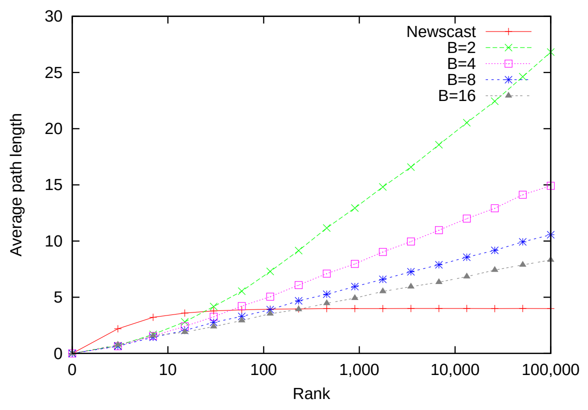


Figure 5.24: Average path lengths in TGT sub-topologies.

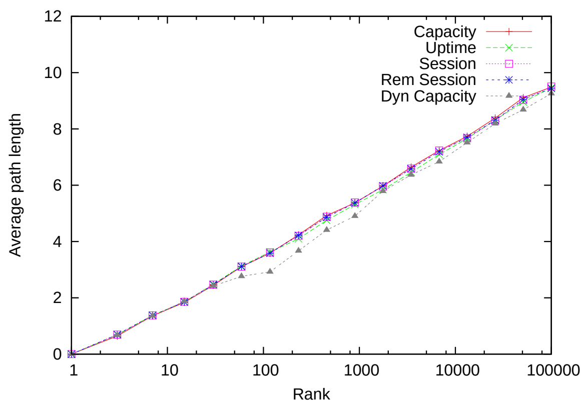


Figure 5.25: Average path lengths in tree sub-overlays in TGTs with different utility metrics.

$Apl'(V_r)$ function for Newscast is almost flat with r, indicating a low correlation between peer utility and Newscast topologies (which are generated randomly). In the tree-based overlays, the average path length in V_r grows linearly with $\log_B(r)$. Therefore, the experiment confirms that the generated topologies have a gradient structure, where the highest utility peers are strongly clustered, and lower utility peers are located at gradually increasing distances from them.

Figure 5.25 shows the average path length in five tree-based sub-overlays created in TGTs with different utility functions. Apart from the previously used utility metrics, based on peer capacity, uptime, expected session, and remaining session, a fifth system is considered (labelled "DynCapacity") where the capacity of peers changes over time. In this system, each peer p is assigned a constant maximum capacity, $C^*(p)$, and its current capacity value, $C(p)$, is calculated at each time step according to formula

$$C(p) = C^*(p) \cdot (1 - \varepsilon) \tag{5.18}$$

where ε is randomly chosen between 0 and ε_{max}. The ε parameter models the interference of external, unpredictable applications, which consume resources at peer p. In Figure 5.25, ε_{max} is set to 0.1, so that the capacity of a peer changes by up 10% at each time step. In all experiments, the average path length between peers grows linearly with $\log_B(r)$, indicating a high robustness of the neighbour selection algorithm to the utility dynamism.

5.4.4 Distance to Core

Another approach to analyse the structure of the generated topologies is to measure the distance from each peer p to the highest utility peer in the system, p_0, denoted $D_t(p, p_0)$ at time step t. Figure 5.26 shows the average value of $D_t(p, p_0)$ as a function of peer rank in Newscast and tree sub-topologies in TGTs with branching factors of 2, 4, and 10. As previously, the graph generated as a histogram,

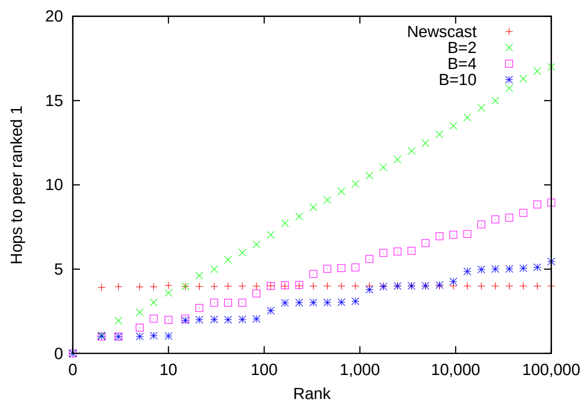


Figure 5.26: Distance to the highest-utility peer in TGT sub-topologies.

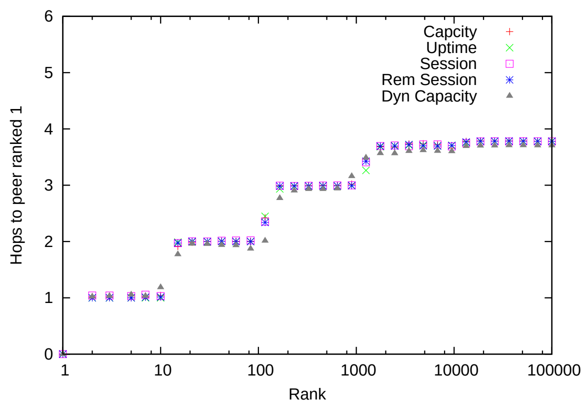


Figure 5.27: Distance to the highest-utility peer in TGTs with different utility metrics.

where each point (r_i, d_i) represents one histogram bin, where

$$d_i = \frac{1}{T} \sum_{t=1}^{T} \frac{\sum_{p \in V'_{i,r}} D_t(p, p_0)}{|V'_{i,t}|} \tag{5.19}$$

and $V'_{i,t}$ is the set of peers ranked between r_i and r_{i-1} at time t

$$V'_{i,t} = \{p : \ r_{i-1} < R(p) \leq r_i\} \tag{5.20}$$

and $V'_{0,t} = \{p_o\}$.

In the Newscast overlay, all peers have an average distance to p_0 approximately equal to 4. This is expected, as the Newscast topology is random and is constantly shuffled by peers. In the other overlays, a clear tree structure is visible, with peers gradually increasing their distance from p_0, the root of the tree. In particular, for $B = 10$, peers ranked from 1 to 9 are directly connected to p_0, peers ranked between 10 and 99 are two overlay hops away from p_0, peers ranked between 100 and 999 are three hops away, peers ranked between 1,000 and 9,999 are four hops away, and so on.

Moreover, the distance to p_0 in the tree-based overlays grows above 4 for low-utility peers, and is higher than the distance to p_0 in the Newscast overlay. This leads to an interesting observation that Newscast connections can be efficiently used for routing by low utility peers, since random neighbours in Newscast sets at these peers are statistically closer to p_0 than the parents in the tree sets.

Figure 5.27 shows the average distance to p_0 over all neighbour subsets in five different TGTs with $B = 10$. The utility function does not have a strong impact on the system topology, and a clear tree structure is visible even when peer utility (defined as capacity) randomly fluctuates at each time step, which shows that the neighbour selection algorithm is resilient to varied system configurations. Again, the distance to p_0 does not grow above 4, since peers ranked between 10,000 and 100,000 use their Newscast links to find the shortest paths to p_0.

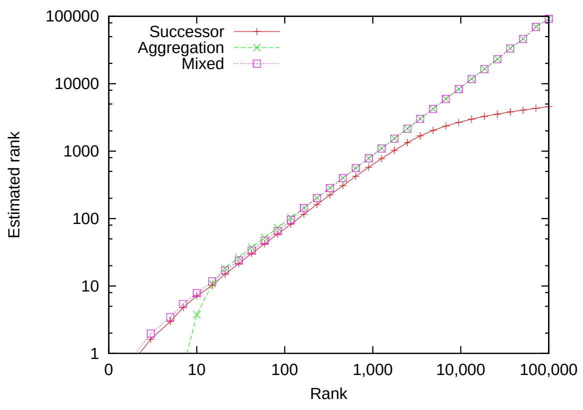


Figure 5.28: Peer ranks estimated using three different methods.

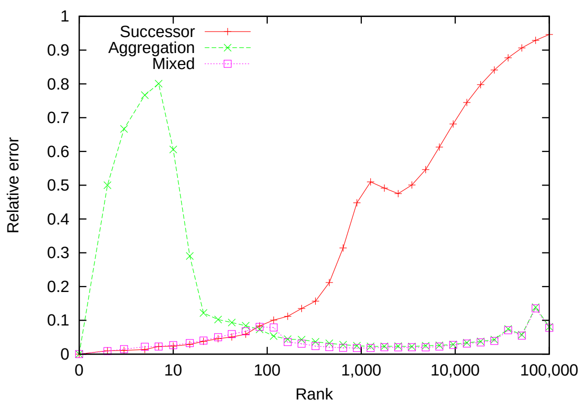


Figure 5.29: Relative estimation error in three rank estimation methods.

5.4.5 Rank Estimation

The following section evaluates peer rank estimation algorithms. The relative error in the rank estimation at peer p is defined as

$$Err_r(p) = \frac{|R(p) - R_p(p)|}{R(p)} \tag{5.21}$$

where $R_p(p)$ is the peer's estimate of its own rank. Figures 5.28 and 5.29 show the average rank estimate, and the average rank estimation error, obtained using the three rank estimation methods described in 4.6, i.e., based on utility successors, utility histograms, and mixed. Both graphs are generated as histograms, based on the peers' true ranks, in the same fashion as in the previous sections.

The successor-based method is relatively accurate for high-utility peers, but provides poor rank estimates for low utility peers. Due to the error propagation between peers, as discussed in section 4.6.2, Err_r increases together with the peer's rank, and as a consequence, peers ranked 100,000 estimate their rank as approximately 4,000.

The histogram-based method, conversely, generates relatively good estimations of peer ranks for low-utility peers, but is significantly inaccurate for high-utility peers. This is caused by the fact that the last histogram bins (those with the highest-utility peers) contain fewer peer samples, and an approximation based on them is statistically less accurate. Furthermore, some of the highest-utility peers may fall outside of the histogram range.

The mixed method combines the advantages of both approaches, achieving the best efficiency and overall estimation error below 20%.

Figure 5.30 shows the relative rank estimation error for the mixed method in a TGT with churn and without churn. In both systems, an increase in the estimation error is observed for the lowest-utility peers, which is caused by the linear interpolation of peer utility histograms. Even with perfectly accurate aggregates, as in the case of no churn, the interpolation produces a distortion. Furthermore,

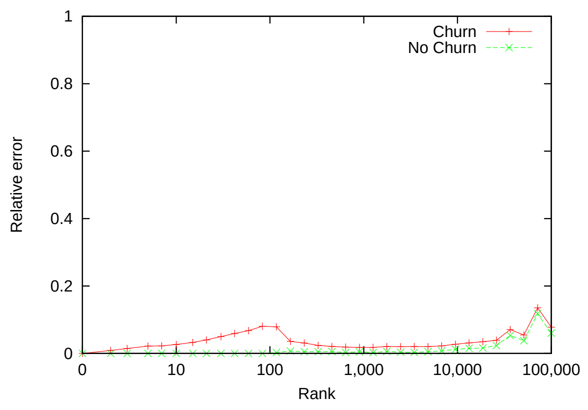


Figure 5.30: Rank estimation error with and without churn.

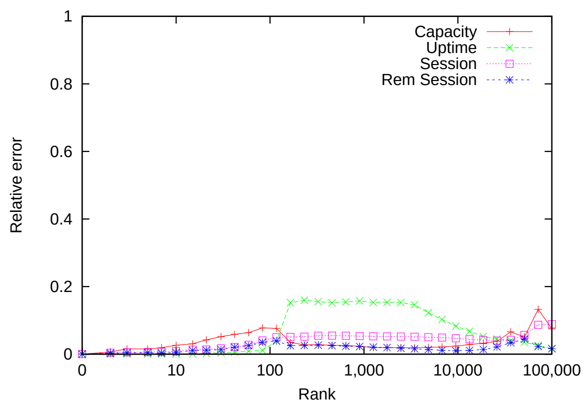


Figure 5.31: Rank estimation error with different peer utility metrics.

a certain estimation error is produced by the successor-based method (used by peers ranked from 0 to 100) due to churn.

Figure 5.31 shows the relative rank estimation error for the mixed method in experiments with different peer utility functions. The error pattern varies between the systems, due to the different peer utility distributions, but it does not exceed 20% in any experiment. As already shown in the previous sections, peers are able to generate gradient topologies with desired properties despite the reported rank estimation error.

5.4.6 Aggregation

The experiments described in this section evaluate the accuracy of the aggregation algorithm. The following notation and metrics are used. Variables $N_{p,t}$, $Avg_{p,t}$, $H_{p,t}$ and $H^c_{p,t}$ denote the current estimations at peer p of the current system size, N, average peer utility, Avg_t, utility histogram, H_t, and capacity histogram H^c_t, respectively, at time step t. The average relative error in the system size approximation, calculated over all peers and all time steps, is defined as

$$Err_N = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{p}\frac{|N_{p,t}-N|}{N}. \tag{5.22}$$

where T is the experiment duration. Similarly, the average error in utility histogram estimation, Err_H, is defined as

$$Err_H = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{p} d(H_t, H_{p,t}) \tag{5.23}$$

where d is a histogram distance function defined as

$$d(H_t, H_{p,t}) = \frac{1}{B}\sum_{i=0}^{B-1}\frac{|H_t(i)-H_{p,t}(i)|}{H_t(i)}. \tag{5.24}$$

Analogously, Err_{Avg} is defined as the average error in the average utility estimation and Err_{H^c} is defined as the average error in the capacity histogram estimation.

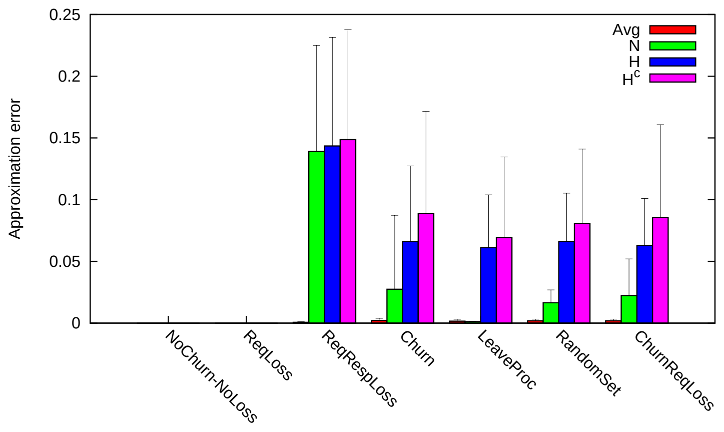


Figure 5.32: Aggregation error in various system models.

Figure 5.32 shows Err_{Avg}, Err_N, Err_H and Err_{H^c} in a number of experiments with varied system configurations. In the experiments with no churn and no message loss, the aggregation algorithm produces almost perfectly accurate system property approximations, with the approximation error below 0.001%. A similarly low approximation error is observed in the configuration with no churn, where request messages are lost with probability P_{loss}, but response messages are always delivered (labelled "ReqLoss"). This result is consistent with the expectations outlined in section 4.4.7.

In the system where both request and response messages can fail, but still in the absence of churn ("ReqRespLoss"), the approximation error reaches approximately 15%. In a system with no message loss, but a positive churn rate ("Churn"), the observed error is lower, approximately 10% for the histograms and 3% for N. The Err_a error does not exceed 0.3% in all experiments, since the approximation of a system average does not introduce any systematic bias, such as an aggregation weight loss, and the errors incurred by peers in individual gossip exchanges cancel out.

The experiment labelled "LeaveProc" evaluates the performance of aggregation when peers perform the leave procedure described in section 4.4.6. The procedure significantly improves the accuracy of N estimation, as it prevents aggregation weight loss. However, it does not significantly affect the error in the histogram estimation, since the population of peer changes during the execution of aggregation, unlike N, which is constant, and when an aggregation instance ends, the produced results are diverge from the current system state. As it is hard to estimate how many peers in a realistic P2P system perform a leave procedure when disconnecting from the network, in all experiments reported in this thesis, it is conservatively assumed that no peers execute the procedure when leaving.

An additional experiment, labelled "RandomSet", compares the performance of aggregation in topologies generated using random sets instead of Newscast sets, which are used in all other experiments. The approximation errors produced in the two systems are similar, indicating that both topologies are suitable for running aggregation algorithms.

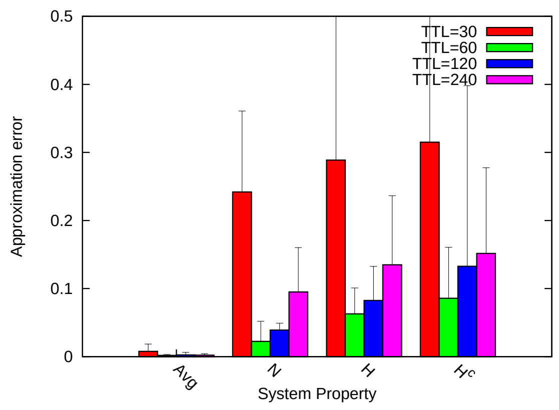


Figure 5.33: Aggregation error versus instance TTL.

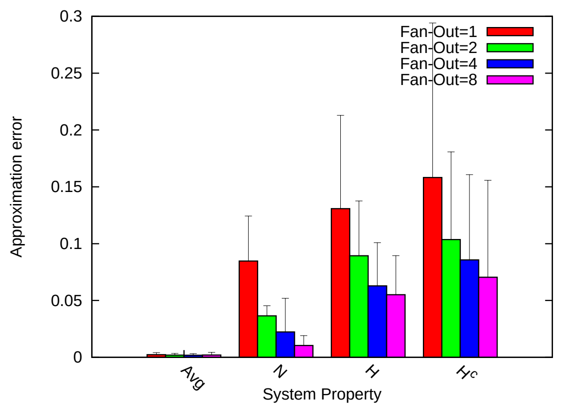


Figure 5.34: Aggregation error versus gossip fan-out.

The last experiment included in Figure 5.32, labelled "ChurnReqLoss", shows the aggregation error in a TGT with the settings that are used by default in all other sections, i.e., with churn, request loss, aggregation over Newscast topologies, and no leave procedure.

There are three parameters that control the cost and accuracy of aggregation, which are the frequency of instance initiation, F, an instance time-to-live, TTL, and the aggregation fan-out, G. Additionally, the histogram resolution, b, impacts on the accuracy of utility distribution approximation.

When F is decreased, peers perform aggregation more frequently, and have more up-to-date estimations of the system properties. However, in the experiments described in this thesis, the system size and the probability distributions of peer utility and capacity are constant, and hence, running aggregation more often does not affect the results. At the same time, when F is decreased, the average message size increases, as an average peer participates in a higher number of aggregation instances.

Similarly, when the TTL parameter is increased, aggregation instances last longer, peers store more local tuples, and aggregation messages become larger. Moreover, as shown in Figure 5.33, when aggregation instances run longer, they suffer higher weight loss, and as a consequence, generate less accurate results. Conversely, if TTL is low, the aggregation instances are too short to produce high quality results, since a certain number of algorithm steps is required to distribute the weight and average out the tuples stored by peers. The optimum performance is achieved for $TTL \approx 60$, and this value for TTL is used in the experiments described in this thesis. The frequency parameter, F, is also set to 60 so that nodes run on average one aggregation instance at a time.

Further, the performance of aggregation can be improved by increasing the fan-out factor, G. As discussed in section 4.4.8, a higher fan-out setting requires that peers exchange more messages when they participate in aggregation instances, but it also shortens the duration of the instances. As a consequence, high fan-out does not increase the average number of messages sent by a peer per time step, as long as $\frac{TTL}{F \cdot G} < 1$, according to formula 4.29. However, when instances are shorter, the

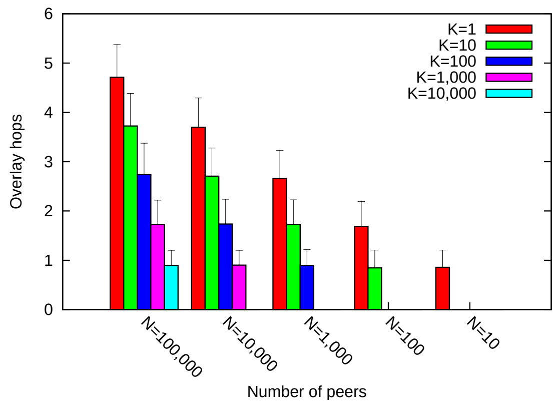


Figure 5.35: Gradient search hop counts in TGTs with variable sizes.

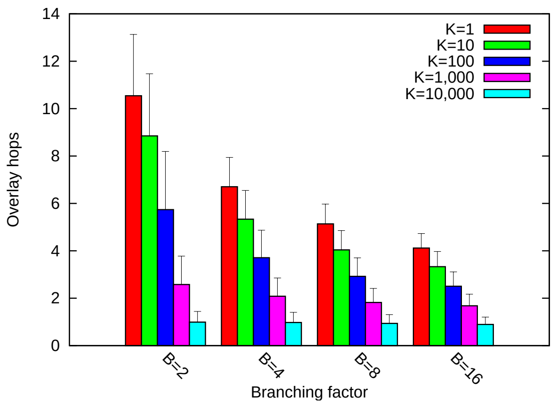


Figure 5.36: Gradient search hop counts in TGTs with different branching factors.

aggregation results are more accurate, since fewer peers join and leave the system when an aggregation instance is running, and the system is less likely to change during the instance execution. This can be confirmed experimentally, as shown in Figure 5.34. Based on these results, G is set to 4 in the other experiments described in this thesis.

Finally, the accuracy of utility distribution approximation can be improved by increasing the histogram resolution, b. Clearly, the message size grows linearly with the number of histogram bins. The actual accuracy improvement depends on the shape of the distribution function and the histogram interpolation method. In this thesis, linear interpolation is used and histograms have 200 bins. This way, aggregation messages have approximately 1.6kB, and would fit well into UDP packets, assuming this protocol was used for the aggregation implementation.

5.4.7 Gradient Search

This section describes a series of experiments that evaluate the performance of gradient search. In each experiment, K super-peers are elected using a top-K threshold, and messages are routed using gradient search from random peers in the topology to super-peers. The experiments measure the average number of edges (also called overlay hops) a message traverses before it is delivered to a super-peer, and the average message loss rate.

5.4.7.1 Overlay Hops

Figure 5.35 shows the average number of message hops in gradient search as a function of the system size (N) and the number of super-peers (K) in a TGT with a branching factor of 10. Furthermore, Figure 5.36 shows the average number of message hops as a function of the branching factor (B) and the number of super-peers (K) in a TGT with 100,000 peers. Both figures demonstrate the average number of message hops grows proportionally to $\log_B N$ and decreases proportionally to $\log_B K$. Thus,

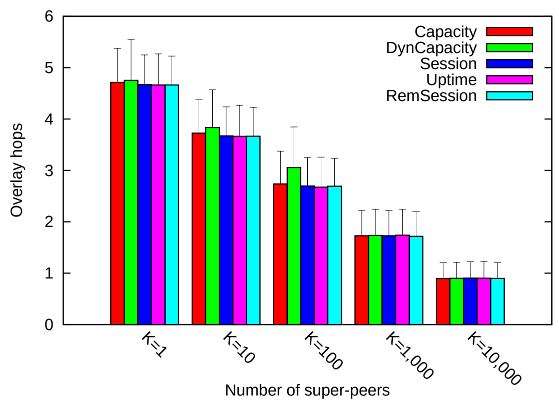


Figure 5.37: Message hop counts in TGTs with different utility metrics.

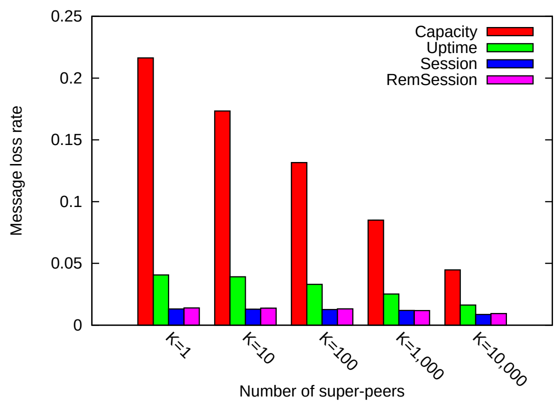


Figure 5.38: Message loss rate in TGTs with different utility metrics.

the experiment empirically confirms Theorem 3.3, which states that every peer in TGT is connected to a super-peer through at most $O(\log_B \frac{N}{K})$ overlay edges.

Figure 5.37 shows the average number of message hops when routed using gradient search in five TGTs with different utility metrics, where $N = 100,000$ and $B = 10$. As expected (see Figures 5.25 and 5.27 for comparison), the choice of utility metric does not affect significantly the structure of the topology and the performance of routing. The only noticeable difference in gradient search performance is found in the experiment with dynamic and unpredictable peer utility ("DynCapacity"), which puts more stress on the neighbour selection algorithm.

5.4.7.2 Loss Rate

In the simplest message failure model, where each message transmission has a fixed failure probability, the average message loss rate is simply proportional to the number of message hops. However, in the proportional model, where the failure probability for a message transmission from peer p to q is proportional to $\frac{1}{Ses(q)}$, as explained in section 5.5, the choice of the peer utility metric has a strong impact on the overall message loss rate in a gradient topology, as shown in Figure 5.38.

In the capacity-based TGT, the utility metric and the system topology are independent from peer stability, and hence message loss rate grows linearly with the message path length, as in the simple message model. In the uptime-based TGT, message loss rate is greatly reduced compared with the capacity-based TGT, since messages are gradually forwarded to peers with increasingly higher uptimes, and hence more stable and less likely to lose messages. In the TGT with session-based utility metrics, the average message loss rate is even lower, and is nearly constant with K. In these systems, a message forwarded over two or more overlay hops reaches high-stability peers for which the probability of message loss is extremely low.

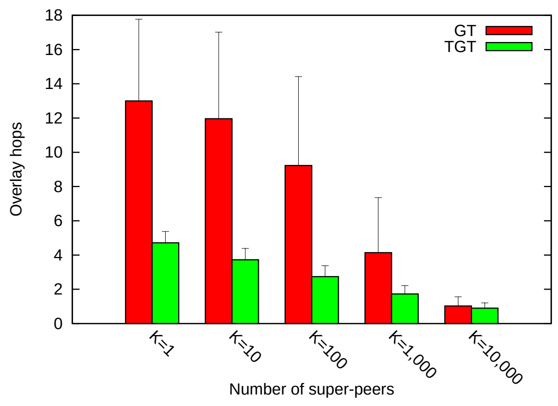


Figure 5.39: Average message hop count in GT and TGT.

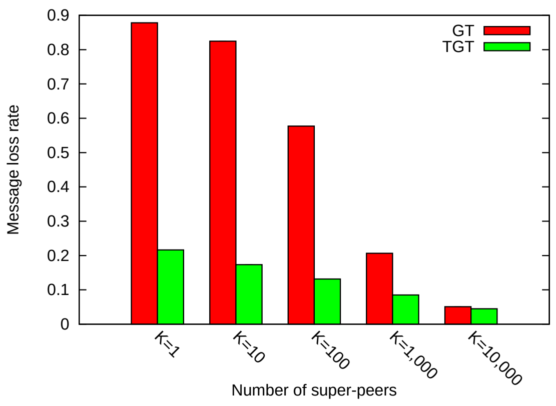


Figure 5.40: Average message loss rate in GT and TGT.

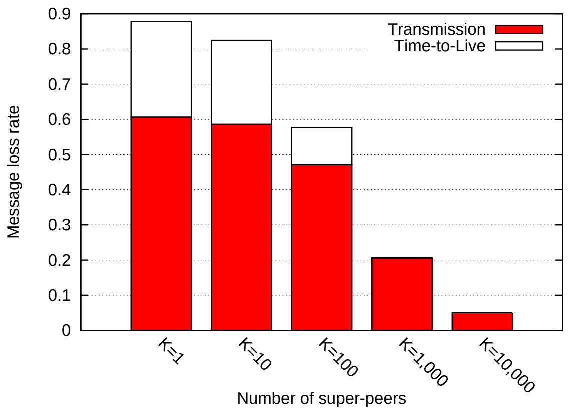


Figure 5.41: Message loss rate in GT.

5.4.7.3 GT versus TGT

In the following set of experiments, TGTs are compared with GTs based on the performance of gradient search. Figures 5.39 and 5.40 show the average number of message hops, and the message loss rate, respectively, in two topologies generated using a capacity-based utility metric and 100,000 peers. With the exception of the configuration where the super-peer ratio is equal to $\frac{1}{10}$, where super-peer can be trivially discovered using random sets, TGTs clearly outperform GTs, both in terms of message hops and loss. GTs exhibit particularly poor performance when the super-peer ratio is below $\frac{1}{100}$.

Figure 5.41 shows in more detail the message loss in GT. In gradient search, a message is lost either when is exceeds its time-to-live (TTL) or when a failure occurs when it is forwarded. As shown in Figure 5.41, GT with a low number of super-peers ($K \leq 10$) suffers a very high message loss rate (more than 80%), since peers are unable to discover the super-peers and discard messages as they exceed their TTL. In TGTs, due to the topology structure, gradient search has a guaranteed cost of $O(\log N)$ overlay hops, and if there is only one super-peer in the system, messages are delivered to this super-peer and TTL is exceeded in a marginal number of cases.

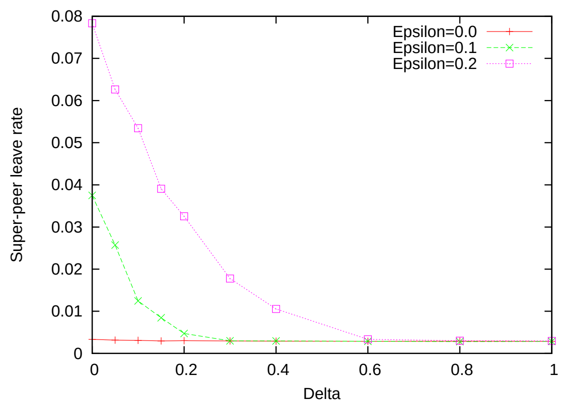


Figure 5.42: Total super-peer leave rate as a function of Δ.

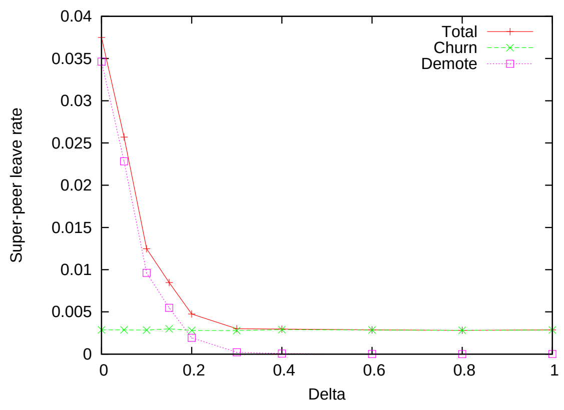


Figure 5.43: Super-peer leave rate for $\varepsilon = 0.1$.

5.4.8 Two-Threshold Election

The following experiments evaluate the stability of super-peers, and the super-peer election error, in a system where peers dynamically change their utility. The capacity of a peer p is calculated at each time step as $C(p) = C^*(p) \cdot (1 - \varepsilon)$, as in section 5.4.3, where $C^*(p)$ is the maximum peer capacity, ε is randomly chosen between 0 and ε_{max}, and ε_{max} is parameter that models the influence of external applications on the peer's capacity. In order to reduce the number of switches between super-peers and clients, super-peers are elected using two top-K utility thresholds, following the approach in outlined in section 4.5.2.

Each experiment is set up with two parameters: ε_{max}, labelled "Epsilon" on the graphs, which determines the amplitude of peer capacity change, and Δ, denoted "Delta" on the graphs, which determines the distance between the super-peer election thresholds. The upper and lower thresholds, t_u and t_l, are calculated in such a way that the number of super-peers is between K and $K - \Delta$, i.e., $D(t_u) = K$ and $D(t_l) = K - \Delta$, where D is a peer utility distribution. The system size is $N = 100,000$ and $K = 1,000$.

Figure 5.42 shows the average rate of super-peer switches with clients as a function of Δ. The experiment demonstrates that the rate of switches sharply decreases as Δ is increased. However, it does not converge to zero, but rather to a constant positive value, since some super-peers always leave the system due to churn, and are continuously replaced by ordinary peers.

Hence, super-peers are swapped with clients for two reasons. First, as super-peers leave the system, ordinary peers are promoted in order to maintain K super-peers in the system. Second, since the utility of individual peers and the super-peer election thresholds constantly fluctuate, some super-peers are occasionally demoted to clients, and clients are occasionally promoted to super-peers.

Figure 5.43 shows the average rates of super-peer demotions and departures in a system with $\varepsilon_{max} = 0.1$. The rate of super-peer departures does not depend on Δ and is determined by the overall

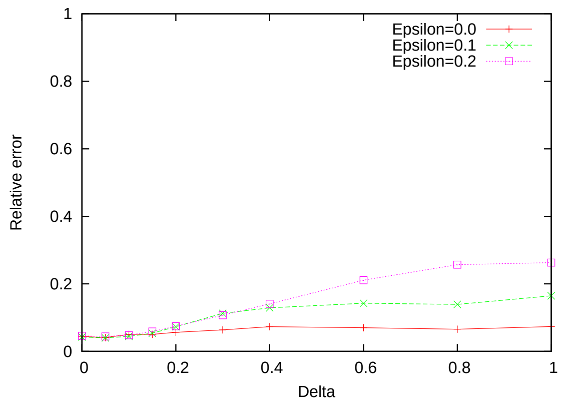


Figure 5.44: Relative error in the number of elected super-peers as a function of Δ.

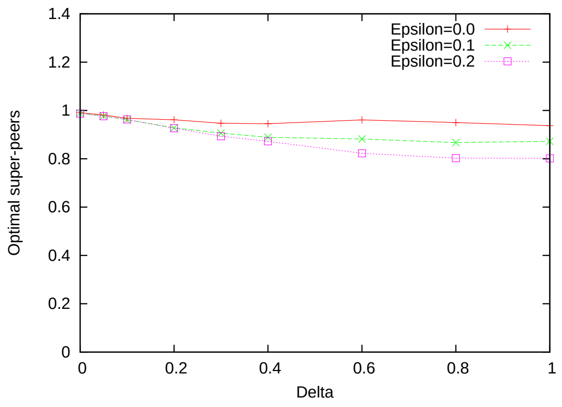


Figure 5.45: Fraction of optimal super-peers as a function of Δ.

system churn. However, the experiment shows that the number of super-peer demotions, caused by peer utility and threshold fluctuations, can be reduced to a negligible level by using an appropriate Δ.

Figure 5.44 shows the impact of Δ on the super-peer election error, Err_s, as defined in section 5.3.3.1. As expected, the error grows together with Δ, since a larger gap between t_u and t_l relaxes the constraints on the number of super-peers in the system. Similarly, the fraction of globally optimum super-peers, Opt, defined in section 5.3.3.2, decreases as Δ is increased, as shown in Figure 5.45. Hence, Δ enables a trade-off between restricting the constraints on the super-peers set and reducing the frequency of switches between super-peers and clients.

5.4.9 Load-Based Election

The final set of experiments evaluates the load-based approach to super-peer election described in section 3.1.1. In these experiments, at each time step, each peer generates a request, or more generally, a unit of load, with probability P_{req}. From the central limit theorem (law of big numbers), the total load in the system, L, produced at a time step follows a normal distribution with a mean of $N \cdot P_{req}$ and a variance of $N \cdot P_{req}(1 - P_{req})$.

The requests are routed to super-peers and distributed between them. For the purpose of these experiments, each super-peer receives a fraction of load proportional to its capacity. Load-balancing strategies are not in evaluated in this study. Peer capacity values follow a Pareto distribution with a mean of 1 and shape parameter of 2, and it is assumed that a super-peer p cannot handle more requests than it capacity value, $C(p)$.

Super-peers are elected using a load-based threshold, defined by formula 3.8 in section 3.1.1. In the performed experiments, three values for the super-peer utilisation parameters, W, are considered: 1 (full utilisation), 0.9, and 0.75.

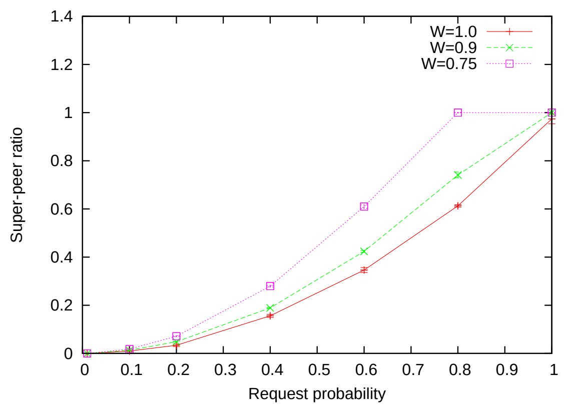


Figure 5.46: Number of super-peers versus average request probability.

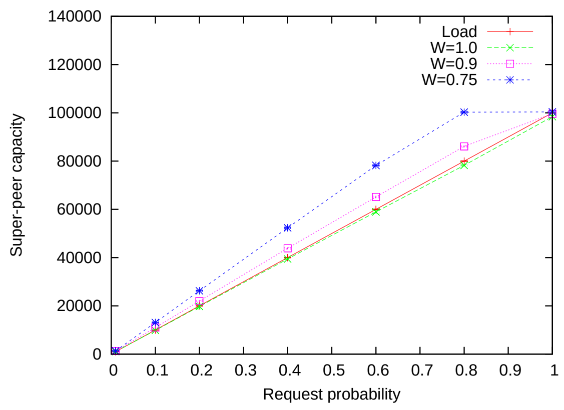


Figure 5.47: Total load and super-peer capacity versus average request probability.

Figure 5.46 shows the relationship between the request probability P_{req} and the total number of super-peers in the system. It can be seen that the system adapts the super-peer set to the increasing load. The number of super-peers initially grows slowly, as high capacity super-peers are available, but the growth rate gradually increases as P_{req} becomes higher, and eventually all peers in the system become super-peers. Moreover, the growth in the number of super-peer is faster for lower values of W, as expected.

Figure 5.47 shows the total system load and super-peer capacity in the same set of experiments. The load in the system increases proportionally to the request probability, as predicted, and the total super-peer capacity scales linearly with the system load. For $W < 1$, the total super-peer capacity exceeds the total system load. Thus, the system achieves its objective and adjusts the super-peer set according to the existing demand. For $W = 1$, the super-peer capacity is marginally below L, due to the aggregation error (weight loss) and histogram interpolation error.

5.4.10 Summary

This section describes a series of experiments that examine whether the algorithms described in chapter 4 generate topologies and super-peer sets that have the properties derived analytically in chapter 3.

The first set of experiments, presented in sections 5.4.1 to 5.4.4, evaluate the neighbour selection algorithms through an analysis of the average path lengths and distances between peers in the generated topologies. The results clearly show that the created topologies have a tree-based structure. In particular, the distance from a peer, p, to the highest utility peer in the system, p_0, grows logarithmically with p's rank, where the logarithm base is equal to the topology branching factor, B, as expected in Theorem 3.2. Moreover, the neighbour selection algorithms manage to construct and maintain such tree-based topologies in a variety of system configurations, with different branching factors and peer utility metrics. The analysis also shows that peers have low degrees, and the fractions of peers that become overloaded by excessive neighbour connections are negligible.

A further set of experiments in section 5.4.7 evaluate gradient search, and show that the generated topologies can be efficiently exploited for routing. As predicted in Theorem 3.3, gradient search delivers a message from a peer to a super-peer in $O(\log_B \frac{N}{K})$ overlay hops, where N is the system size, K is the number of super-peers, and B is the branching factor, which is verified in a number of experiments with varied N, K, B, and peer utility metrics. Moreover, gradient search significantly reduces message loss rate in topologies where the peer utility metric is based on peer stability.

Another set of experiments, described in section 5.4.6, evaluates the aggregation algorithm and shows that it approximates global system properties with an average relative error below approximately 15%. The algorithm generally conforms to the expectations outlined in sections 4.4.6, 4.4.7, and 4.4.8, and its performance can be tuned using parameters such as instance frequency, F, instance duration, TTL, and gossip fan-out, G.

The rank estimation algorithms are evaluated in section 5.4.5. As expected in section 4.6.3, the average error produced by the mixed method, based on aggregation and utility successor sets, is the lowest, and is below 20%. Moreover, such a level of rank estimation error does not prevent peers from generating tree-based gradient topologies.

The thesis also introduces three super-peers election techniques for gradient topologies, i.e., based on single-thresholds, double-thresholds, and with no super-peer demotion. The first of these techniques is evaluated in section 5.3.3, which shows that a single-threshold election allows a precise restriction on the number of super-peers and generates close-to-optimum super-peer sets. The second election method, with no super-peer demotions, is evaluated in section 5.3.4, which confirms that this election technique entirely eliminates switches between super-peers and clients and maximises the average super-peer session length. The two-threshold election method is evaluated in section 5.4.8, which shows that the use of two thresholds enables a trade-off between imposing constraints on the number and utility of super-peers and the reducing the frequency of switches between super-peers and clients.

5.5 Simulator Validation

This section validates the custom-built simulator, which has been used to generate the results presented in this thesis. The validation is performed by running two identical sets of experiments in the custom simulator and PeerSim. In the latter, the event-driven mode is enabled and node communication is asynchronous. In order to determine message latencies, PeerSim is fed with a trace containing nearly 3 million wide-area network latency measurements generated by the King tool [69]. Due to the computational cost, the system size is reduced to 10,000 nodes, but all other simulation parameters are set in the same way as in the previous sections, as summarised in Table 5.6.

Figures 5.48, 5.49 and 5.50 show the average path lengths, distances to the highest utility peers,

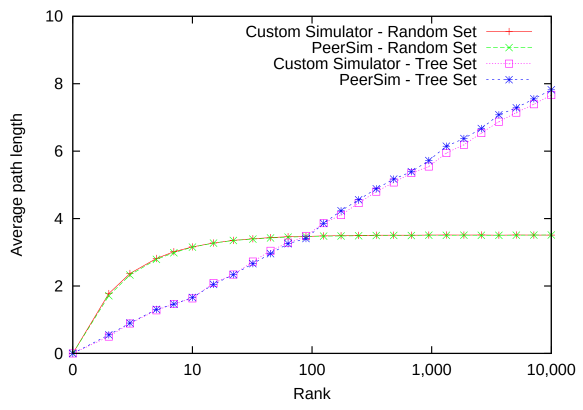


Figure 5.48: Average path lengths in TGT sub-topologies in PeerSim and the custom simulator.

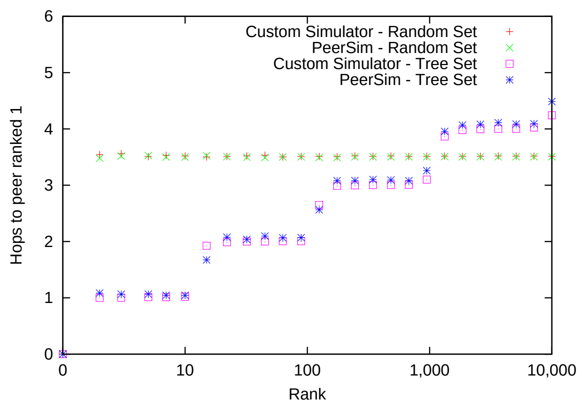


Figure 5.49: Average distance to the highest utility peer in TGT sub-topologies in PeerSim and the custom simulator.

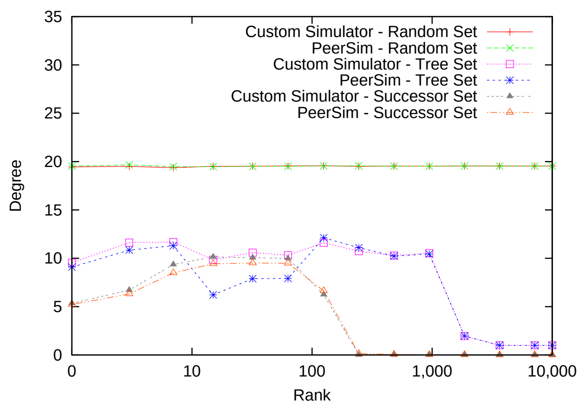


Figure 5.50: Average peer degree versus peer rank in PeerSim and the custom simulator.

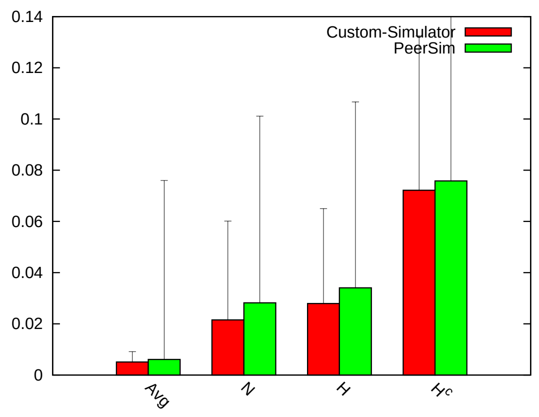


Figure 5.51: Aggregation error in PeerSim and the custom simulator.

and peer degrees, respectively, in TGT topologies generated in PeerSim and the custom simulator. Remarkably, the results generated by the two simulators overlap almost ideally in most experiments. This empirically confirms the hypothesis stated in section 5.2.2 that message latencies observed on the Internet do not have a significant impact on the neighbour selection algorithms described in this thesis. In both simulators, the algorithms generate nearly identical topologies.

Figure 5.51 shows the aggregation error for a number of system properties in PeerSim and the custom simulator. As expected, message latencies on the order of 100 milliseconds (typically experienced on the Internet) only marginally reduce the accuracy of aggregation, where the gossip period is 5 seconds. As a consequence, the super-peer election algorithm generates almost the same results in PeerSim and the custom simulator.

Finally, Figures 5.52 and 5.53 show the average number of overlay hops and failure rates for messages routed using gradient search in PeerSim and the custom simulator. The number of super-peers is denoted by K. Again, both simulators produce similar results in most experiments. The only

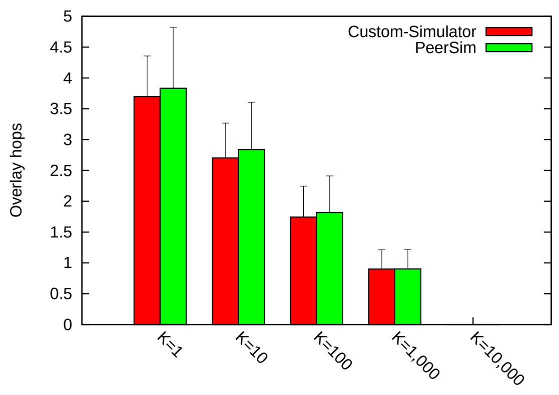


Figure 5.52: Average number of hops for gradient search in PeerSim and the custom simulators. K represents the number of super-peers.

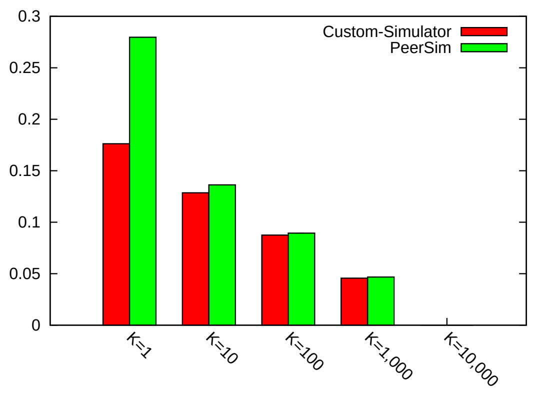


Figure 5.53: Average failure rate for gradient search in PeerSim and the custom simulators.

exception is the experiment for $K = 1$, where PeerSim significantly differs from the custom simulator. Higher message latency in PeerSim causes relatively more frequent and severe deviations in the TGT structure, which in turn impact on the routing performance. Overall, the experiments described in this section demonstrate that the custom-built simulator generates consistent results with PeerSim.

5.6 Application Example

This section demonstrates the practical viability of gradient topologies by applying the experimental results, derived in the previous sections, to a sample application scenario. The selected application scenario is based on the storage system described in section 3.3. In order to evaluate the impact of a gradient topology, two variations of this system are considered: a traditional DHT overlay, where all nodes participate in the data storage, and a system based on a gradient topology, where only the super-peers store the data and run the DHT protocol. For consistency with the previous experiments, it is assumed that peer properties follow a Pareto distribution, mean peer session duration is equal to $\mu = 360$ time steps, mean peer downstream bandwidth capacity is equal to $\mu_c = 10$, and the ratio of super-peers to the system size is equal to $\frac{1}{100}$. The storage system is run by N peers and is used to permanently store D bytes of data.

5.6.1 Traditional DHT Storage System

In a traditional DHT, in order to access a data item, a user first discovers the node that hosts the requested item and then communicates directly with this node. Thus, the discovery operation requires $O(\log N)$ message transmissions, and the average download rate from the hosting node, assuming a low contention ratio, is approximately equal to μ_c.

The average peer departure rate in the DHT is equal to $\frac{1}{\mu}$. If the data stored in the DHT is not replicated, the average rate at which the data is lost is proportional to the average peer leave rate, and is equal to $\frac{D}{\mu}$. In a DHT with k-replication, i.e., where each data item is replicated at k independent nodes, the probability of a data item loss can be estimated in the following way. Let t_r be the minimum amount of time needed to transfer a data item between two nodes, $t_r \ll \mu$. A data item is lost if all k nodes that host it leave the system during t_r time units. Hence, the probability for a data item loss during time t_r can be estimated as $(\frac{t_r}{\mu})^k$.

Finally, k-replication requires a certain maintenance cost. Each time a node leaves, its data needs to be restored on another node using the remaining replicas. Since nodes store in total kD data and leave the system at an average rate of $\frac{1}{\mu}$, the average background traffic needed to maintain k-replication is equal to $\frac{kD}{\mu}$ bytes per time unit.

5.6.2 Gradient Topology Storage System

In a storage system with a tree-based gradient topology, the cost of data discovery is comparable to a traditional DHT. As shown in section 3.3, and confirmed in the experiments in section 5.4.7, a DHT running on top of a TGT requires $O(\log N)$ message transmissions to route a messages to a selected data item.

However, by selecting the highest utility peers for storing data, a gradient topology can improve data access performance and reliability. In a gradient topology, where utility is defined as peer bandwidth capacity, the average super-peer bandwidth capacity is equal to $10\mu_c = 100$, as predicted by Theorem 3.1 and verified empirically in Figure 5.5. Thus, the average data item download rate in a gradient topology, assuming a low system utilisation, is 10 times faster compared to a traditional DHT.

In should be noted that a tenfold improvement in super-peer bandwidth capacity is not possible with the other evaluated super-peer systems. Due to the reasons discussed in the previous sections, these algorithms elect suboptimal super-peer sets, and the ratio between the mean super-peer bandwidth capacity and mean peer capacity μ_c is only 5 for SPChord and between 8 and 9 for H-DHT and SG-1 (see Figure 5.5). SOLE does not improve super-peer capacity over μ_c due to its simple election criteria.

Super-peers can also be used to reduce the data loss rate and maintenance cost (in case of replication) if peer utility is based on peer stability. Depending on the heuristic used to estimate peer session duration, super-peer sessions in a gradient topology are from 3 times (in case of an uptime-based utility function) to 10 times (accurate knowledge of peer session durations) higher than the average peer session $\mu = 360$, as shown in Figure 5.9. Thus, in the gradient topology, the average super-peer leave rate is between $\frac{1}{3\mu}$ and $\frac{1}{10\mu}$, and is 3 to 10 times lower compared to the overall peer leave rate in a traditional DHT overlay. If no data replication scheme is applied, the average data loss rate in the gradient topology is then up to 10 times lower (i.e., $\frac{D}{10\mu}$) compared to the traditional DHT. Moreover, in a gradient topology with data k-replication at super-peers, the probability of a data item loss rate during interval t_r can be estimated as $(\frac{t_r}{10\mu})^k$, and is up to 10^k times lower compared to the traditional DHT.

At the same time, gradient topology reduces the replication overhead. A super-peer stores on average $kD\frac{100}{N}$ data. During a short interval t, approximately $\frac{t}{10\mu}\frac{N}{100}$ super-peers leave the system, and hence, $k\frac{tD}{10\mu}$ data must be transferred between the remaining replicas. This implies that the average maintenance traffic is $\frac{kD}{10\mu}$ bytes per time unit, and hence is 10 times lower compared to a traditional DHT.

Finally, it should be noted that the other evaluated super-peer systems achieve significantly lower super-peer stability due to suboptimum super-peer election and swappings between super-peers and clients. In particular, even with a perfect knowledge of peer session times, H-DHT extends super-peer session only 7 times compared to the mean μ, SG-1 (with all optimisations enabled) generates super-peer session with lengths merely equal to μ (i.e., there is no improvement in data stability), and both SOLE and SPChord significantly reduce super-peer session lengths compared to the mean μ.

5.6.3 Summary

This section evaluates the performance benefits of using a gradient topology in a DHT-based storage system. By increasing the utility of data storing peers, the gradient topology improves both the data access performance and reliability. In particular, while keeping the same $O(\log N)$ latency and message cost for data discovery, it improves the data download rate and data loss probability by an order of magnitude compared to a traditional P2P storage system. Moreover, due to the increased data stability, the gradient topology also reduces the replica maintenance cost. Comparable performance improvements cannot be achieved using other known super-peer election techniques.

Chapter 6

Conclusion

This chapter summarises the main contributions of the thesis, and outlines directions for future work.

6.1 Accomplishments

This thesis introduces a novel approach to dealing with heterogeneity in P2P systems using gradient topologies. A gradient topology has a fundamental property that for any given utility threshold, all peers with utility above this threshold are located close to each other, in terms of overlay hops, and form a connected sub-overlay. Such high-utility peers can be then exploited by higher-level applications in a similar fashion as super-peers in traditional P2P systems. Furthermore, the information captured in the topology enables a search heuristic, called gradient search, that enables efficient discovery of high-utility peers.

The thesis introduces a subclass of gradient topologies, called tree-based gradient topologies, which have a logarithmic diameter and allow routing messages from an ordinary peer to a super-peer in $O(\log_B \frac{N}{K})$ overlay hops, where N is the system size, K is the number of super-peers, and B is a constant system parameter called branching factor. TGTs are simple and easy to generate. A node in a TGT maintains only one link to a parent node and a few randomly links to other nodes.

TGTs have been designed to support a wide class of large-scale P2P applications, such as storage systems, name services, file-sharing applications, and semantic registries, where the system performance and reliability can be improved by assigning relevant system tasks, such as hosting system data, running services, or participating in certain distributed algorithms, to the most stable and best performing peers. The thesis describes two such proof-of-concept applications.

TGTs can be generated by a periodic neighbour selection algorithm executed at each peer. The thesis describes the design of such an algorithm and evaluates it using a custom-built P2P simulator. The evaluation confirms that the algorithm constructs topologies that have the desired tree-based structure.

The thesis describes a number of super-peer election thresholds, which impose different constraints on the super-peers sets. In particular, the thesis introduces a top-K threshold that elects a fixed number of super-peers, a proportional threshold that maintains a fixed ratio of super-peers to clients, a capacity threshold that restricts the total super-peer capacity, and other thresholds that allow more sophisticated super-peer management. All thresholds are calculated using a decentralised aggregation algorithm which approximates global system properties, such as the system size, total load, and peer utility distribution. The thesis also describes super-peers election techniques, based on utility thresholds, that restrict the number of super-peers in the overlay and reduce the frequency of switches between super-peers and clients.

In a range of experiments, it is shown that gradient topologies, together with aggregation-based election techniques, generate better-quality and higher-stability super-peer sets, at a similar maintenance cost, compared to state-of-the-art super-peer systems. Moreover, it is shown that gradient topologies offer more flexible and more powerful super-peer election mechanisms compared with the existing P2P systems, and thus extend the current state-of-the-art knowledge on heterogeneous P2P systems.

Even though the thesis describes gradient topology as a single and unified architecture, its individual components can be treated as separate contributions, and can be used as independent building blocks in other P2P systems. In particular, the aggregation algorithm can be used as a generic tool for estimating global properties in decentralised systems. Through a distribution function approximation, the algorithm can address classic problems such node ranking [126] and slicing [79, 56, 66, 127].

Similarly, the election strategies described in this thesis can be applied separately from gradient topologies. For example, a system with a random P2P topology can elect super-peers by aggregating peer utility information and calculating utility thresholds using the described algorithms. The information about such elected super-peers can be then disseminated to ordinary peers using a gossip-based broadcast algorithm [54, 80].

Nevertheless, gradient topologies complement well with these components, since they guarantee that super-peers, elected using utility thresholds, are well connected with each other and can be discovered using gradient search. While the election thresholds can be changed, peers in the gradient topology do not need to be migrated or re-connected. Moreover, in a natural way, gradient topologies can support multiple super-peer sets by assigning multiple utility thresholds.

6.2 Future Work

This section briefly outlines a sample of promising research topics that can be identified based on the work described in this thesis.

6.2.1 Security

One of the most important aspects for future work on gradient topologies is security. Nearly all techniques and algorithms developed in this thesis assume a fully collaborative P2P environment. This section outlines the main challenges, and points out potential approaches, when designing a security model for gradient topologies.

The main security threat in a gradient topology is posed by malicious peers which may take harmful actions against other peers. A secure P2P system should be able to function correctly (without a significant performance degradation) despite an existence of such malicious peers. In the context of this thesis, this means that peers should be able to construct a gradient topology, elect appropriate super-peers, and enable access to these super-peers.

Utility

The most critical information in a gradient topology is that about peer utility. By providing fake utility status, a malicious peer may change its position in the topology and become a super-peer. A secure gradient topology must then provide a mechanism for peers to verify the utility of their neighbours. For example, peers may give each other feedback in order to assess their neighbours' utility. Such techniques, based on the notion of trust in a decentralised system, are described in [86, 193, 28]. Another potential approach is to compute, in a decentralised fashion, the reputation for each node, and discard peers that have a low reputation (i.e., provide fake utility information) [61, 47, 138, 72].

Aggregation

Another vulnerable component in the gradient topology, and perhaps the most challenging to secure, is the aggregation algorithm. By disseminating fake information, a malicious node can influence the global aggregation outcome, and this way, manipulate the super-peer election and node ranking (and hence topology construction). A malicious peer can also intentionally increase the periodicity of its aggregation algorithm in order to have a greater impact on the aggregation result. Moreover, a peer can initiate a large number of aggregation instances in order to increase the system overhead and potentially cause a denial of service.

The non-malicious nodes can protect themselves from these attacks in a number of different ways. First, peers may try to identify and discard fake responses from their neighbours based on the knowledge from the current and previous aggregation instances. As each instance gradually converges over time, peers can refine their knowledge on the expected instance outcome and detect outliers (i.e., forged responses). Second, peers may cache the responses received from their neighbours and compare them with the final aggregation outcome. Misbehaving peer can then be identified and isolated.

Again, reputation management techniques [61, 47, 138, 72] can be used to black-list malicious nodes. Similarly, peers can correlate the information on gossip exchanges in order to detect nodes that increase their periodicity or maliciously initiate large numbers of instances. This may require using cryptographic mechanisms to trace node identity. Finally, peers can impose restrictions on which nodes can initiate aggregation instances, for example based on their reputation scores.

Routing

A common threat in a P2P system is a refusal to route messages by certain peers. Malicious peers may also want to route messages in an illegal way and tamper with forwarded messages. In the gradient topology, this may prevent access from clients to super-peers. A simple way to alleviate this problem is to randomise the routing algorithm and allow message retransmissions in case of failures. Assuming that malicious peers constitute only a small fraction of all peers in the system, this approach allows each peer, with a high probability, to successfully route messages. Techniques to randomise gradient search are described below in section 6.2.3.

Neighbour Selection

Malicious nodes in a gradient topology may also refuse to participate in the neighbour selection algorithm, or to provide illegal neighbour candidates to gossipping peers. Since the gossipping algorithm is highly randomised and each peer verifies its new neighbours through direct connections (symmetric neighbourhood model), the neighbour selection algorithm may be already able to tolerate a certain fraction of malicious peers.

Super-Peer Election

A special situation occurs when a malicious peer becomes a super-peer. However, since the super-peers functionality is entirely application-specific, it is difficult to asses in the general case the impact of a malicious super-peers on the system performance. For the same reason, incentives for peers to serve as super-peers can only be considered in a particular application scenario.

6.2.2 Multiple Utility Functions

Another interesting research issue is whether a single gradient topology can support multiple applications with different utility requirements. Without loss of generality, two applications can be considered, A and B, where each application introduces its own utility function, U_A and U_B, respectively. In such a scenario, the goal of a gradient topology is to elect super-peers with high values of U_A for use by application A, and super-peers with high values of U_B for use by application B.

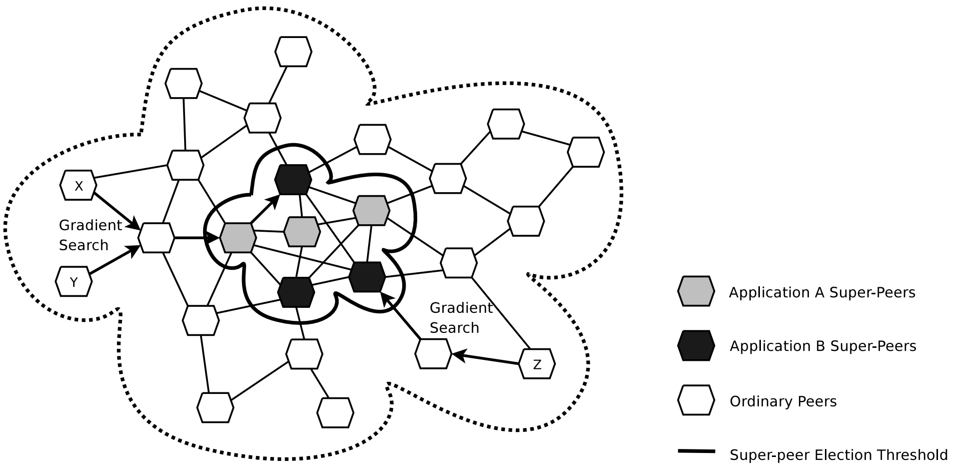

X
Y
Z
Gradient Search
Gradient Search
Application A Super-Peers
Application B Super-Peers
Ordinary Peers
Super-peer Election Threshold

Figure 6.1: Two applications sharing a common gradient topology.

A naive approach to this problem is to generate two independent gradient overlays, using the two utility functions and the algorithms described in this thesis. However, this would double the system overhead. A better approach is to combine the two utility functions into one general utility function U and to generate one gradient overlay shared by both applications. A convenient way of defining such a common utility function is

$$U(p) = \max(U_A(p), U_B(p)). \tag{6.1}$$

This has the advantage that both peers with high value of U_A and peers with high value of U_B have high utility U, and hence are located in the core and can be discovered using gradient search. The only change required in the routing algorithm is that a search message, once delivered to a high utility peer p in the core, may have to be forwarded to a different peer in the core, since p either has a high value U_A or U_B. This last step, however, with a high probability can be achieved in one hop, since peers in the core are well-connected.

The proposed approach is illustrated in Figure 6.1. A sample gradient topology supports two applications, A and B. Ordinary peers X, Y, and Z perform gradient search to discover super-peers for application B. Peers X and Y locate an "A-type" super-peer in the core and their request is forwarded to a "B-type" super-peer. Peer Z discovers a "B-type" super-peer directly.

The super-peer election thresholds for the two applications, t_A and t_B, can be estimated using the aggregation algorithm, where the histograms for both U_A and U_B are generated through the same aggregation instance in order to reduce the number of generated messages. However, a potential problem may appear if the two utility functions, U_A and U_B, have significantly different value ranges, since the composed utility U may be dominated by one of the utility functions. For example, if U_A has values within range [0..1] and U_B has values in range [1..100], then U is essentially equal to U_B, and searching for peers with high U_A becomes inefficient.

One way to mitigate this problem is to define the two utility functions in such a way that both have the same value ranges, e.g., [0..1]. However, this requires system-wide knowledge about peers. Simple transformations or projections onto a fixed interval, for example using a sigmoid function, do not fix the problem, since if one function has higher values than the other function, the same relation holds when the transformation has been applied. A better approach is to scale one of the two utility functions using the current values of the super-peer election thresholds, for example in the following way

$$U(p) = \max(\frac{t_B}{t_A} U_A(p),\, U_B(p)). \tag{6.2}$$

This has the advantage that the core of the gradient topology, determined by the threshold t_B, contains peers with U_A above t_A and peers with U_B above t_B, since if $U(p) > t_B$ for a peer p then either $U_A(p) > t_A$ or $U_B(p) > t_B$.

6.2.3 Multi-Path Routing

The gradient search algorithm has the drawback that it always forwards messages along the same paths, unless the topology changes, which may lead to an imbalance in the routed traffic between peers. This is especially probable in the presence of "heavy hitters", i.e., peers generating large amounts of traffic, as commonly seen in P2P systems [173]. Moreover, faulty or non-cooperating peers, as well as broken peer connections, may prevent peers from reaching their super-peers.

These problems can be addressed by allowing multi-path routing. In a TGT where peers maintain multiple parent neighbours (i.e., the size of the tree set is greater than one), a message can be forwarded to a parent neighbour chosen randomly or using a round-robin strategy.

A more general approach, suitable for any gradient topology, would be to route messages probabilistically. In Boltzmann routing, peer p selects neighbour q for the next-hop destination for a routed message with a probability $P_p(q)$ calculated using the Boltzmann exploration formula [182]

$$P_p(q) = \frac{e^{(U_p(q)/T)}}{\sum_{r \in \mathcal{N}_p} e^{(U_p(r)/T)}} \tag{6.3}$$

where T is a parameter called the temperature that determines the "greediness" of the algorithm. Setting T close to zero causes the algorithm to be more greedy and deterministic, as in gradient search, while if T grows to infinity, all neighbours are selected with equal probability as in random walking. Thus, the temperature enables a trade-off between exploitative (and deterministic) routing of messages towards high-utility peers, and random exploration that spreads the traffic more equally between peers.

6.2.4 Histograms Adaptation

The thesis describes a relatively simple approach to the construction of utility histograms and utility distribution approximation, where all histograms bins have equal width and evenly divide the interval between the minimum and maximum peer utility. The accuracy of the peer utility distribution approximation may be potentially improved, if the bins in the histogram, and hence the interpolation points, are dynamically adjusted by peers at system runtime. In particular, is may be desirable to increase the density of bins close to the super-peer election thresholds. Potentially, the choice of the histogram bins can be dictated by the interpolation method, especially if a more sophisticated approach, for example based on cubic splines, is used instead of the linear interpolation. A proactive adjustment of histogram bins may also eliminate the need for utility successor sets in the rank estimation algorithm.

6.2.5 Locality-Aware Gradient Topologies

Gradient topologies have been designed in this thesis with one main intention: to allow a P2P system to exploit its highest utility peers. The next research question then is whether gradient topologies can be built based on two constraints: peer utility and location. In a locality-aware gradient topology, a peer selects its neighbours based on not only their utility, but also their distance, defined according to some metric. Potentially, node locations and distances can be determined using a virtual coordinate system, such as GNP [136] and Vivaldi [45]. The goal of the neighbour selection algorithm is to construct a topology that preserves its general gradient structure (in particular, supports gradient search and maintains super-peer connectivity), but also preferentially connects nodes that are close to each other. This way, the topology can improve the performance of gradient search and allows peers to discover super-peers in their proximity.

6.2.6 Higher-Level Applications

The thesis describes the design of two proof-of-concept applications for gradient topologies, a storage system and a name service. These two applications are described only briefly, and a significant amount of work is required to fully specify them, validate, and implement. The proposed design can be then treated as a starting point for a wider research topic.

As shown through a theoretical analysis and simulation experiments, the properties of gradient topologies are promising and suggest that gradient topologies can be successfully applied to many different large-scale and heterogeneous P2P systems. Supporting such systems is ultimately the farthest-reaching goal of this thesis.

Bibliography

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the 9th International Conference on Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2001.

[2] K. Aberer and M. Hauswirth. An overview of peer-to-peer information systems. In *Distributed Data & Structures 4, Records of the 4th International Meeting (WDAS'02)*, volume 14 of *Proceedings in Informatics*, pages 171–188. Carleton Scientific, 2002.

[3] A. Acharya and J. Saltz. A study of internet round-trip delay. Technical Report CS-TR-3736, University of Maryland, January 1996.

[4] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, June 2002.

[5] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[6] V. Agarwal and R. Rejaie. Adaptive multisource streaming in heterogeneous peer-to-peer networks. *Proceedings of the SPIE*, 5680:13–25, January 2005.

[7] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, Jan 2002.

[8] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[9] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*, pages 73–80, Washington, DC, USA, 2006. IEEE Computer Society.

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

[11] N. B. Azzouna and F. Guillemin. Analysis of adsl traffic on an ip backbone link. In *Proceedings of the Global Telecommunications Conference*, volume 7, pages 3742–3746. IEEE, 2003.

[12] N. B. Azzouna and F. Guillemin. Impact of peer-to-peer applications on wide area network traffic: an experimental approach. In *Proceedings of the Global Telecommunications Conference*, volume 3, pages 1544–1548. IEEE, 2004.

[13] S. Bajaj, L. Breslau, D. Estrin, K. F. andSally Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, Los Angeles, CA, USA, 1999.

[14] A.-L. Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, 288:60–69, 2003.

[15] S. A. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. In *Proceedings of the 25th INFOCOM IEEE International Conference on Computer Communications*, pages 1–11. IEEE, April 2006.

[16] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th Global Internet Symposium in conjunction with IEEE INFOCOM*, pages 79–84. IEEE, May 2007.

[17] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker. Replication strategies for highly available peer-to-peer storage. In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 153–158. Springer, 2003.

[18] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 256–267. Springer, 2003.

[19] K. Birman, B. Constable, M. Hayden, J. Hickey, C. Kreitz, R. Van Renesse, O. Rodeh, and W. Vogels. The horus and ensemble projects: accomplishments and limitations. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, pages 149–161. IEEE, 2000.

[20] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 123–138. ACM Press, 1987.

[21] B. Biskupski, M. Schiely, P. Felber, and R. Meier. Tree-based analysis of mesh overlays for peer-to-peer streaming. In *Proceedings of the 8th International Conference on Distributed Applications and Interoperable Systems*, volume 5053 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2008.

[22] J.-C. Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High-Speed Networks*, 2(3):305–323, December 1993.

[23] M. S. Borella, D. Swider, S. Uludag, and G. B. Brewster. Internet packet loss: Measurement and implications for end-to-end qos. In *Proceedings of the 1998 International Conference on Parallel Processing Workshops*, pages 3–12, Washington, DC, USA, 1998. IEEE Computer Society.

[24] J. M. Boyce and R. D. Gaglianello. Packet loss effects on mpeg video sent over the public internet. In *Proceedings of the 6th ACM international conference on Multimedia*, pages 181–190, New York, NY, USA, 1998. ACM.

[25] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33:59–67, May 2000.

[26] G. A. D. D. Brito and A. M. de Carvalho Moura. ROSA - P2P: a peer-to-peer system for learning objects integration on the web. In *Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, pages 1–9, New York, NY, USA, 2005. ACM.

[27] F. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in P2P protocols. In *Proceedings of the 8th international workshop on Web content caching and distribution*, pages 233–246. Springer Netherlands, 2004.

[28] V. Cahill, E. Gray, J.-M. Seigneur, C. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using Trust for Secure Collaboration in Uncertain Environments. *IEEE Pervasive Computing Magazine*, 2(3):52–61, 2003.

[29] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[30] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 9–18. IEEE Computer Society, 2004.

[31] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation*, pages 85–98. USENIX Association, 2005.

[32] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20:1489–1499, 2002.

[33] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 298–313, New York, NY, USA, 2003. ACM.

[34] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 407–418, New York, NY, USA, 2003. ACM.

[35] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying skype user satisfaction. *SIGCOMM Computer Communication Review*, 36(4):399–410, 2006.

[36] J. Chu, K. Labonte, and B. N. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, volume 4868, pages 310–321, 2002.

[37] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002.

[38] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.

[39] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[40] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the 1st International Workshop on Designing Privacy Enhancing Technologies*, pages 46–66. Springer-Verlag, 2000.

[41] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, pages 251–260, June 2003.

[42] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 177–190. ACM, 2002.

[43] N. Collier. Repast: An extensible framework for agent simulation. *http://www.econ.iastate.edu/tesfatsi/RepastTutorial.Collier.pdf* – Accessed in February 2007.

[44] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 23–34, Washington, DC, USA, 2002. IEEE Computer Society.

[45] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the SIGCOMM 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26. ACM, 2004.

[46] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 33–44. Springer, February 2003.

[47] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 207–216. ACM, 2002.

[48] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, January 1990.

[49] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.

[50] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[51] J. R. Douceur. Is remote host availability governed by a universal law? *ACM SIGMETRICS Performance Evaluation Review*, 31(3):25–29, 2003.

[52] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, pages 75–80. IEEE, May 2001.

[53] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *Proceedings of the 16th international conference on World Wide Web*, pages 883–892, New York, NY, USA, 2007. ACM.

[54] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[55] A. Fasbender and P. Davis. Measurement, modelling and emulation of internet round-trip delays. In *Proceedings of the 8th International Conference on Modelling Techniques and Tools*

for Computer Performance Evaluation, volume 977 of *Lecture Notes In Computer Science*, pages 401–415, London, UK, 1995. Springer-Verlag.

[56] A. Fernández, V. Gramoli, E. Jiménez, A.-M. Kermarrec, and M. Raynal. Distributed slicing in dynamic systems. In *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems*, page 66. IEEE, 2007.

[57] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE Network*, 17(6):6–16, 2003.

[58] R. M. Fujimoto. Parallel discrete event simulation. In *Proceedings of the 21st Conference on Winter Simulation*, pages 19–28. ACM, 1989.

[59] L. Garcés-Erice, E. W. Biersack, P. Felber, K. W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *Proceedings of the 9th International Euro-Par Conference on Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 1230–1239. Springer, 2003.

[60] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, 31(1):48–59, 1982.

[61] A. Garg, R. Battiti, and G. Costanzi. Dynamic self-management of autonomic systems: The reputation, quality and credibility (RQC) scheme. In *Proceedings of the 1st International IFIP Workshop on Autonomic Communication, Revised Selected Papers*, volume 3457 of *Lecture Notes in Computer Science*, pages 165–178. Springer, 2004.

[62] D. M. Geels. Data replication in oceanstore. Technical Report UCB/CSD-02-1217, EECS Department, University of California, Berkeley, December 2002.

[63] T. J. Giuli and M. Baker. Narses: A scalable flow-based network simulator. Technical Report cs.PF/0211024, Stanford University, September 2006.

[64] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

[65] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[66] V. Gramoli, Y. Vigfusson, K. Birman, A.-M. Kermarrec, and R. van Renesse. A fast distributed slicing algorithm. In *Proceedings of the 27th ACM symposium on Principles of distributed computing*, pages 427–427. ACM, 2008.

[67] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, pages 1–6, 2006.

[68] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of Symposium on Operating Systems Principles*, pages 314–329, 2003.

[69] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18. ACM, 2002.

[70] K. Guo, W. Vogels, and R. van Renesse. Structured virtual synchrony: exploring the bounds of virtual synchronous group communication. In *Proceedings of the 7th ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, pages 213–217. ACM, 1996.

[71] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 160–169. Springer, Ocober 2003.

[72] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 144–152. ACM, 2003.

[73] M. Hayden. *The Ensemble System.* PhD thesis, Cornell University: Dept. of Computer Science, 1997.

[74] Q. He, M. H. Ammar, G. F. Riley, H. Raj, and R. Fujimoto. Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems. In *Proceedings of the 11th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 71–78. IEEE Computer Society, 2003.

[75] O. Heckmann, A. Bock, A. Mauthe, and R. Steinmetz. The edonkey file-sharing network. In *GI Jahrestagung (2)*, volume 51 of *LNI*, pages 224–228. GI, 2004.

[76] H.-C. Hsiao and C.-T. King. Tornado: a capability-aware peer-to-peer storage overlay. *Journal of Parallel and Distributed Computing*, 64:747–758, June 2004.

[77] M. Jelasity and Ö. Babaoglu. T-man: Gossip-based overlay topology management. In *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[78] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer, 2004.

[79] M. Jelasity and A.-M. Kermarrec. Ordered slicing of very large-scale overlay networks. In A. Montresor, A. Wierzbicki, and N. Shahmehri, editors, *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, pages 117–124. IEEE Computer Society, October 2006.

[80] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands, 2003.

[81] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 102–109. IEEE Computer Society, 2004.

[82] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23:219–252, August 2005.

[83] G. P. Jesi, A. Montresor, and Ö. Babaoglu. Proximity-aware superpeer overlay topologies. In *Proceedings of the 2nd IEEE International Workshop on Self-Managed Networks, Systems, and Services*, volume 3996 of *Lecture Notes in Computer Science*, pages 43–57. Springer, June 2006.

[84] X. Jiang, Y. Dong, D. Xu, and B. Bhargava. GnuStream: a P2P media streaming system prototype. In *Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 325–328. IEEE, 2003.

[85] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, number 2735 in Lecture Notes in Computer Science, pages 98–107. Springer, October 2003.

[86] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International World Wide Web Conference*, pages 640–651. ACM, 2003.

[87] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *Proceedings of the Global Telecommunications Conference*, volume 3, pages 1532–1538. IEEE, 2004.

[88] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 36–43, New York, NY, USA, 2004. ACM.

[89] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

[90] M. Kleis, E. K. Lua, and X. Zhou. Hierarchical peer-to-peer networks using lightweight superpeer topologies. In *Proceedings of the 10th IEEE Symposium on Computers and Communications*, pages 143–148. IEEE Computer Society, 2005.

[91] D. Kostić, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using random subsets to build scalable network services. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 19–19. USENIX Association, 2003.

[92] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 282–297. ACM, 2003.

[93] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi. A statistical theory of chord under churn. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 93–103. Springer, 2005.

[94] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, 2000.

[95] K. Kutzner and T. Fuhrmann. Measuring large overlay networks - the overnet example. In *KiVS*, pages 193–204. Springer, 2005.

[96] G. Kwon and K. D. Ryu. An efficient peer-to-peer file sharing exploiting hierarchy and asymmetry. In *Proceedings of the 2003 Symposium on Applications and the Internet*, pages 226–233. IEEE Computer Society, 2003.

[97] K. Lakshminarayanan and V. N. Padmanabhan. Some findings on the network performance of broadband hosts. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 45–50, New York, NY, USA, 2003. ACM.

[98] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa network. In *Proceedings of the 3rd International Workshop on Internet Applications*, pages 112–120. IEEE Computer Society, June 2003.

[99] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, number 2735 in LNCS, pages 207–215. Springer, 2003.

[100] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-*

to-Peer Systems, number 3279 in Lecture Notes in Computer Science, pages 87–99. Springer, January 2005.

[101] Y. Li, X. Huang, F. Ma, and F. Zou. Building efficient super-peer overlay network for DHT systems. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing*, volume 3795 of *Lecture Notes in Computer Science*, pages 787–798. Springer, 2005.

[102] J. Liang, R. Kumar, and K. Ross. The KaZaA overlay: A measurement study. *Computer Networks*, 50:842–858, April 2006.

[103] LimeWire. How gnutella works. *http://wiki.limewire.org/index.php?title=How_ Gnutella_ Works* Accessed on 25th March 2008.

[104] S. Lin, A. Pan, Z. Zhang, R. Guo, and Z. Guo. WiDS: An integrated toolkit for distributed system development. In *Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 17–17. USENIX, 2005.

[105] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, and J. Li. Scalable supernode selection in peer-to-peer overlay networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 18–27, Washington, DC, USA, 2005. IEEE Computer Society.

[106] D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, pages 2–9. IEEE Computer Society, 1995.

[107] P. G. López, C. Pairot, R. Mondéjar, J. P. Ahulló, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. In *Proceedings of the 4th International Workshop on Software Engineering and Middleware*, volume 3437 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[108] A. Löser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the 1st International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, volume 2944 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2003.

[109] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing*, pages 84–95. ACM, 2002.

[110] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 94–103. Springer, 2002.

[111] A. Madhukar and C. Williamson. A longitudinal study of P2P traffic classification. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation MASCOTS*, pages 179–188, Washington, DC, USA, 2006. IEEE Computer Society.

[112] N. Magharei and R. Rejaie. PRIME: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1415–1423. IEEE, 2007.

[113] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2003.

[114] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[115] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140. USITS, March 2003.

[116] C. Mastroianni, D. Talia, and O. Verta. A super-peer model for building resource discovery services in grids: Design and simulation analysis. In *Proceedings of the European Grid Conference*, volume 3470 of *Lecture Notes in Computer Science*, pages 132–143. Springer-Verlag, February 2005.

[117] C. Mastroianni, D. Talia, and O. Verta. A super-peer model for resource discovery services in large-scale grids. *Future Generation Computer Systems*, 21(8):1235–1248, 2005.

[118] P. Maymounkov and D. Maziéres. Kademlia: A peer-to-peer information system based on the xor metric. In *1st International Workshop on Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer, March 2002.

[119] A. Medina, A. Lakhina, I. Matta, and J. W. Byers. Brite: An approach to universal topology generation. In *Proceedings of the 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 346–356. IEEE Computer Society, 2001.

[120] S. Merugu, S. Srinivasan, and E. W. Zegura. p-sim: A simulator for peer-to-peer networks. In *Proceedings of the 11th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 213–218. IEEE Computer Society, 2003.

[121] J. W. Mickens and B. D. Noble. Exploiting availability prediction in distributed systems. In *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation*, pages 73–86. USENIX Association, 2006.

[122] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[123] A. T. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proceedings of the 3rd IEEE Workshop on Internet Applications*, pages 104–111. IEEE Computer Society, June 2003.

[124] A. Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 202–209. IEEE Computer Society, August 2004.

[125] A. Montresor, M. Jelasity, and O. Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 19–28. IEEE Computer Society, June 2004.

[126] A. Montresor, M. Jelasity, and O. Babaoglu. Decentralized ranking in large-scale overlay networks. In *Proceedings of the 2nd International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 208–213. IEEE, 2008.

[127] A. Montresor and R. Zandonati. Absolute slicing in peer-to-peer systems. In *Proceedings of the 22nd International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE, 2008.

[128] S. B. Moon, J. Kurose, P. Skelly, and D. Towsley. Correlation of packet delay and loss in the internet. Technical Report 98-11, University of Massachusetts, Amherst, MA, USA, January 1998.

[129] G. E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):114–117, April 1965.

[130] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, and I. Wakeman. Towards Yet Another Peer-to-Peer Simulator. In *Proceedings of the 4th International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, 2006.

[131] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *ACM SIGCOMM Computer Communication Review*, 37(2):95–98, 2007.

[132] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of the 11th international conference on World Wide Web*, pages 604–615. ACM, 2002.

[133] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proceedings of the 12th International Conference on World Wide Web*, pages 536–543. ACM, 2003.

[134] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1:177–186, February 2004.

[135] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[136] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, pages 170–179. IEEE, 2002.

[137] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 432–441. Springer, 2005.

[138] B. C. Ooi, C. Y. Liau, and K.-L. Tan. Managing trust in peer-to-peer systems using reputation-based techniques. In *Proceedings of the 4th International Conference on Advances in Web-Age Information Management*, volume 2762 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2003.

[139] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Supporting heterogeneity and congestion control in peer-to-peer multicast streaming. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems*, volume 3279 of *Lecture Notes in Computer Science*, pages 54–63. Springer, 2004.

[140] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2005.

[141] A. V. Patrick Reynolds. Efficient peer-to-peer keyword searching. In *Middleware*, volume 2672 of *LNCS*, pages 21–40. Springer, 2003.

[142] V. Paxson. End-to-end internet packet dynamics. *SIGCOMM Computer Communication Review*, 27(4):139–152, 1997.

[143] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th annual ACM symposium on Parallel algorithms and architectures*, pages 311–320. ACM, 1997.

[144] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent P2P file-sharing system: Measurements and analysis. In *the 4th International Workshop on Peer-To-Peer Systems*, Cornell, USA, February 2005.

[145] Y. J. Pyun and D. S. Reeves. Constructing a balanced, (log(n)/loglog(n))-diameter super-peer topology for scalable P2P systems. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 210–218. IEEE Computer Society, 2004.

[146] B. Qiao, G. Wang, and K. Xie. A self-organized semantic clustering approach for super-peer networks. In *Proceedings of the 7th International Conference on Web Information Systems Engineering*, volume 4255 of *Lecture Notes in Computer Science*, pages 448–453. Springer, 2006.

[147] B. Qiao, G. Wang, and K. Xie. A taxonomy-based approach for constructing semantics-based super-peer networks. In *Proceedings of the APWeb/WAIM 2007 International Workshops on Advances in Web and Network Technologies, and Information*, volume 4537 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2007.

[148] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured P2P systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2003.

[149] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[150] R. Rejaie and A. Ortega. Pals: peer-to-peer adaptive layered streaming. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 153–161. ACM, 2003.

[151] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[152] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, Berkeley, CA, USA, 2003. USENIX Association. Best Student Paper Award.

[153] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, pages 127–140. USENIX, June 2004.

[154] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. *SIGCOMM Computer Communication Review*, 35(4):73–84, 2005.

[155] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1):50–57, 2002.

[156] K. P. B. Robbert van Renesse and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[157] S. Robinson. *Simulation: The Practice of Model Development and Use.* John Wiley & Sons, 2004.

[158] J. Rosenberg. Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols. IETF Internet draft. Work in progress, October 2007.

[159] J. Rosenberg, R. Mahy, and P. Matthews. Traversal Using Relays around NAT (TURN): Relay extensions to Session Traversal Utilities for NAT (STUN). IETF Internet draft. Work in progress, July 2008.

[160] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). IETF Internet draft. Work in progress, July 2008.

[161] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs). RFC 3489, March 2003.

[162] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350. Springer-Verlag, 2001.

[163] J. Sacha and J. Dowling. A self-organising topology for master-slave replication in peer-to-peer environments. In *Proceedings of the 3rd International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, pages 52–64. Springer, July 2005.

[164] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Discovery of stable peers in a self-organising peer-to-peer gradient topology. In *Proceedings of the 6th IFIP International Conference on Dis-*

tributed Applications and Interoperable Systems, number 4025 in LNCS, pages 70–83. Springer-Verlag, June 2006.

[165] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Using aggregation for adaptive super-peer discovery on the gradient topology. In *Proceedings of the 2nd IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan)*, number 3996 in Lecture Notes in Computer Science, pages 77–90. Springer-Verlag, June 2006.

[166] D. Sanghi, A. Agrawala, O. Gudmundsson, and B. Jain. Experimental assessment of end-to-end behavior on internet. In *Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. INFOCOM*, volume 2, pages 867–874. IEEE, 1993.

[167] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9(1):170–184, July 2003.

[168] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Proceedings of the 4th International Symposium on Network Computing and Applications*, pages 109–118. IEEE Computer Society, 2005.

[169] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A scalable and ontology-based P2P infrastructure for semantic web services. In *Proceedings of the 2nd International Conference on Peer-to-Peer Computing*, pages 104–111. IEEE, 2002.

[170] C. Schmidt and M. Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2):211–229, June 2004.

[171] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the 1st International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE Computer Society, 2001.

[172] T. J. E. Schwarz, Q. Xin, and E. L. Miller. Availability in global peer-to-peer storage systems. In *the 6th Workshop on Distributed Data and Structures*, 2004.

[173] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12:219–232, April 2004.

[174] S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, and kc claffy. The rtt distribution of tcp flows in the internet and its impact on tcp-based flow control. Technical Report tr-2004-02, CAIDA, January 2004.

[175] M. Siddiqui, A. Villazón, J. Hofer, and T. Fahringer. Glare: A grid activity registration, deployment and provisioning framework. In *Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing*, pages 52–67. IEEE Computer Society, 2005.

[176] A. Singh and M. Haahr. Creating an adaptive network of hubs using schelling's model. *Communications of the ACM*, 49(3):69–73, 2006.

[177] A. Singla and C. Rohrs. Ultrapeers: Another step towards gnutella scalability. version 1.0. Technical report, Lime Wire LLC, November 2002.

[178] N. Spring, L. L. Peterson, A. C. Bavier, and V. S. Pai. Using planetlab for network research: myths, realities, and best practices. *Operating Systems Review*, 40(1):17–24, 2006.

[179] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 149–160, 2001.

[180] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1), 2003.

[181] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pages 189–202. ACM, 2006.

[182] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, 1998.

[183] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 175–186. ACM, 2003.

[184] R. Tian, Y. Xiong, Q. Zhang, B. Li, B. Y. Zhao, and X. Li. Hybrid overlay structure based on random walks. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 152–162. Springer, February 2005.

[185] B. Traversat, M. Abdelaziz, and E. Pouyoul. Project JXTA: A loosely-consistent DHT rendezvous walker. Technical report, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, USA, 2003.

[186] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, and B. Yeager. Project JXTA 2.0 super-peer virtual network. Technical report, Sun Microsystems, Inc., May 2003.

[187] D. Tsoumakos and N. Roussopoulos. A comparison of peer-to-peer search methods. In *Proceedings of the 6th International Workshop on the Web and Databases*, pages 61–66, 2003.

[188] K. Tutschku. A measurement-based traffic profile of the edonkey filesharing service. In *Proceedings of the 5th International Workshop on Passive and Active Network Measurement*, volume 3015 of *Lecture Notes in Computer Science*, pages 12–21. Springer, 2004.

[189] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Operating Systems Review. Special Issue: Peer-to-peer infrastructure*, 36:271–284, 2002.

[190] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[191] K. Wang, J. Huang, Z. Li, X. Wang, F. Yang, and J. Bi. Scaling behavior of internet packet delay dynamics based on small-interval measurements. In *Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary*, pages 140–147, Washington, DC, USA, 2005. IEEE Computer Society.

[192] T. I. Wang, K. H. Tsai, and Y. H. Lee. Crown: An efficient and stable distributed resource lookup protocol. In *Proceedings of the 1st International Conference on Embedded and Ubiquitous Computing*, volume 3207 of *Lecture Notes in Computer Science*, pages 1075–1084. Springer, 2004.

[193] Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 150–157. IEEE Computer Society, 2003.

[194] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela. SETI@HOMEmassively distributed computing for seti. *Computing in Science and Engineering*, 3:78–83, January 2001.

[195] S. Wolf and P. Merz. Evolutionary local search for the super-peer selection problem and the p-hub median problem. In *Proceedings of the 4th International Workshop on Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[196] L. Xiao, Z. Zhuang, and Y. Liu. Dynamic layer management in superpeer architectures. *IEEE Transactions on Parallel and Distributed Systems*, 16:1078–1091, 2005.

[197] Z. Xu and Y. Hu. Sbarc: A supernode based peer-to-peer file sharing system. In *Proceedings of the 8th IEEE International Symposium on Computers and Communications*, pages 1053i–1060, Washington, DC, USA, 2003. IEEE Computer Society.

[198] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, pages 345–352. IEEE, 1999.

[199] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 5–14. IEEE, 2002.

[200] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering*, pages 49–60, Bangalore, India, March 2003. IEEE Computer Society.

[201] W. Yang and N. B. Abu-Ghazaleh. Gps: A general peer-to-peer simulator and its use for modeling bittorrent. In *Proceedings of the 13th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 425–434. IEEE Computer Society, 2005.

[202] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of the 26th IEEE Conference on Computer Communications INFOCOM*, pages 594–602. IEEE, 1996.

[203] E. W. Zegura, K. L. Calvert, and M. J. Donahoo. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, 1997.

[204] X. Zhang, J. Liu, B. Li, and Y.-S. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 2102–2111. IEEE, March 2005.

[205] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of internet path properties. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 197–211, New York, NY, USA, 2001. ACM.

[206] B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pages 34–44. Springer, March 2002.

[207] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, January 2004.

[208] S. Zhao, D. Stutzbach, and R. Rejaie. Characterizing files in the modern gnutella network. *ACM/SPIE Multimedia Systems Journal*, 1:35–50, March 2007.

[209] Y. Zhu, H. Wang, and Y. Hu. A super-peer based lookup in structured peer-to-peer systems. In *Proceedings of the ISCA 16th International Conference on Parallel and Distributed Computing Systems*, pages 465–470. ISCA, 2003.