TECHNISCHE
UNIVERSITÄT
DARMSTADT

MONITORING AND MANAGEMENT OF PEER-TO-PEER SYSTEMS

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

DISSERTATIONSSCHRIFT

von

DIPL.-MATH., DIPL.-INFORM. KÁLMÁN GYÖRGY GRAFFI

Geboren am 18. Juni 1982 in Neumarkt, Rumänien

Erstreferent: Prof. Dr.-Ing. Ralf Steinmetz
Korreferent: Prof. Carmen Guerrero López, Ph.D.

Tag der Einreichung: 26.05.2010
Tag der Disputation: 20.07.2010

Darmstadt, 2010
Hochschulkennziffer D17

Szüleimnek és családomnak, akik mindent lehetövé tettek,
for myself, for going this path between the sun and the ice,
und für eine glückliche Zukunft mit der Frau, die ich liebe.

ABSTRACT

The peer-to-peer paradigm has had large success in content distribution and multimedia communication applications on the Internet. In a peer-to-peer network, the participating nodes create an infrastructure to provide a desired functionality and offer their resources to host an application in a distributed manner. Besides the functional requirements of an application, the non-functional requirements to achieve a high service quality are also an important part of successful peer-to-peer networks and a major challenge is to meet these requirements in networks with unreliable nodes. In contrast to traditional centralized approaches where the quality can be measured and controlled, in a distributed environment it is challenging both to capture the status and performance of the whole distributed system in one point of time and to control its general behavior. In this dissertation, we focus on the monitoring and management of peer-to-peer systems.

We systematically engineer *SkyEye.KOM*, a fully decentralized monitoring mechanism that provides both a precise status snapshot of the peer-to-peer system and enables queries for peer capacities, such as bandwidth or storage capacities, in a large-scale peer-to-peer system. It considers individual load limits of the peers and ensures that no peer is overloaded. The core tree topology of *SkyEye.KOM* is established and maintained solely with protocol-relevant messages. It is based on local peer identifier calculations and using the underlying peer-to-peer overlay.

As a second step, we focus on the management of peer-to-peer systems and introduce P^3R^3O.KOM and *SkyNet.KOM*, two solutions to manage both the reservation of available capacities in the peer-to-peer system and the system behavior in a fully decentralized and efficient manner. P^3R^3O.KOM is a peer-to-peer protocol for reliable long-term resource reservation that overcomes the limitations of traditional peer-to-peer services, which typically are host only by single peers and cease once the service providing peer fails. Resource reservations are fulfilled with adjustable guarantees (even 100%) in the presence of strong churn through the automated and fully decentralized management of the resource provision redundancy.

With *SkyNet.KOM*, we present a fully decentralized approach for automated management of peer-to-peer systems following the principles of autonomic computing. It allows the user or system provider to set service quality goals for the peer-to-peer system, which are automatically verified by the monitoring solution *SkyEye.KOM* and analyzed, aligned and enforced by the other components of *SkyNet.KOM*. Preset quality goals for the peer-to-peer system are reached and held through automated systematic re-configuration of the individual components of the peer-to-peer system.

At the end, we present *LifeSocial.KOM*, a peer-to-peer-based platform for online social networks that incorporates the proposed monitoring mechanism to show the feasibility and application scope of the monitoring and management solutions.

ZUSAMMENFASSUNG

Das Peer-to-Peer-Paradigma findet breite Verwendung in der Verteilung von Daten und Dokumenten sowie direkter Multimedia-Kommunikation im Internet. In einem Peer-to-Peer-Netzwerk erschaffen die beteiligten Netzwerkknoten unter Einsatz ihrer Ressourcen eine Infrastruktur, die eine gewünschte Funktionalität oder Anwendung völlig dezentral anbietet und betreibt. Neben der Erfüllung der funktionalen Anforderungen an eine Internet-Anwendung ist die Erfüllung der nicht-funktionalen Anforderungen, wie die erbrachte Dienstgüte, eine große Herausforderung in hochskalierenden Peer-to-Peer Netzwerken, die aus unzuverlässigen Einzelknoten bestehen. Im Gegensatz zu zentralisierten Ansätzen mit einem überwach- und anpassbaren zentralen Anbieter ist es in einer verteilten Umgebung schwierig eine globalen Sicht auf den Zustand eines Peer-to-Peer-Systems, sowie die Anpassung der Dienstgüte eines Peer-to-Peer-Systems zu bieten, da es keinen direkten Ansatzpunkt gibt. Die Dissertation widmet sich dieser Fragestellung und fokussiert auf die Qualitäts- und Kapazitätenüberwachung sowie die Dienstgüte-Verwaltung von Peer-to-Peer-Systemen.

In der Dissertation wird *SkyEye.KOM* systematisch entwickelt und erforscht, ein völlig dezentraler Monitoringmechanismus, der auf bestehende Peer-to-Peer-Overlays aufsetzt und sowohl eine globale Sicht auf den Status des Peer-to-Peer-Systems, als auch die Funktion bietet, gezielt nach freien Kapazitäten im hoch-skalierenden Peer-to-Peer-Netzwerk zu suchen. Der Ansatz ist robust bei dynamischer Knotenanzahl im Netzwerk als auch präzise und leichtgewichtig im Betrieb.

Mit P^3R^3O.KOM und mit *SkyNet.KOM* werden zwei Mechanismen zur Dienstgüte-Verwaltung von Peer-to-Peer-Systemen vorgeschlagen, die es unter der Nutzung von *SkyEye.KOM* erlauben, sowohl die Kapazitäten im Peer-to-Peer-System vollständig und für langfristige Dienste zu nutzen, als auch die erbrachte Gesamt-Dienstgüte des Peer-to-Peer-Systems zu überwachen und zu steuern. P^3R^3O.KOM ist ein Peer-to-Peer-Mechanismus für verlässliche Ressourcen-Reservierung, der die Einschränkung in der Ressourcennutzung, die durch die begrenzte Lebenszeit der Peers gegeben ist, aufhebt. Mit P^3R^3O.KOM werden Ressourcenreservierungen durch eine kontrollierte und völlig dezentrale Anpassung der Redundanz der Ressourcenallokation mit einer einstellbaren Garantie bis zu 100% verlässlich erfüllt.

Die Verwaltung oder das Management von Peer-to-Peer-Systemen zielt auf die Steuerung des Verhaltens des Peer-to-Peer-Systems ab, das durch verschiedene Performanz- und Kostenmetriken bestimmt wird. In der Dissertation wird der Bedarf in Peer-to-Peer-Systemen nach automatisierter Qualitätssteuerung unter Anwendung der Prinzipien des Autonomic-Computing-Ansatzes untersucht und motiviert. Für die völlig dezentrale Steuerung der Dienstgüte von Peer-to-Peer-Systemen wird mit *SkyNet.KOM* ein Rahmenwerk vorgestellt, der es Nutzern und System-Anbietern ermöglicht verbindliche Qualitätsziele für das System zu definieren. Diese werden dann im Peer-to-Peer-System kontinuierlich durch *SkyEye.KOM* überwacht und anschließend analysiert. Bei einem Verfehlen der Dienstgüteziele wird automatisiert eine optimierte Rekonfiguration des gesamten Systems initiiert. Vorgegebene Dienstgüteziele in Form von validen Metrikintervallen werden von dem Peer-to-Peer-System automatisch erreicht und gehalten.

Um die Umsetzbarkeit und das Anwendungsfeld der Monitoring- und Management-Mechanismen zu demonstrieren, wird in dieser Dissertation das Fallbeispiel der digitalen sozialen Netzwerke untersucht und mit *LifeSocial.KOM* eine Plattform für Peer-to-Peer-Anwendungen im Allgemeinen und digitale soziale Netzwerke im Speziellen entwickelt, die sowohl auf die vielfältigen Anforderungen an die Peer-to-Peer-Plattform eingeht als auch eine breite Basis an Grundfunktionalität für digitale soziale Netzwerke in Form von Plugins bietet. Die Monitoring-Lösung *SkyEye.KOM* ist integriert und ermöglicht eine detaillierte Sicht auf die Funktionsweise und Qualität der dezentralen *LifeSocial.KOM*-Plattform im laufenden Betrieb.

Time - this is the one thing that not even a grateful person can give back.

- Lucius Annaeus Seneca

ACKNOWLEDGMENTS

Die Fertigstellung der Promotion ist vergleichbar mit einem Marathonlauf, Ausdauer und Anstrengung sind verbunden mit einem Ziel, das bis zum Schluss in weiter Ferne scheint. Dankbar ist man im Ziel allen, die einen auf dem Weg begleitet haben, die Jahre mit Freude erfüllt haben und einem Möglichkeiten boten über sich hinauszuwachsen.

Sehr herzlich möchte ich mich bei Ralf Steinmetz bedanken, der mit dem Multimedia Communications Lab, KOM, eine Umgebung geschaffen hat, die einem Möglichkeiten und Chancen in einer breiten Fülle bietet. I also thank my co-supervisor Carmen for the nice atmosphere and constructive discussions in Madrid on my thesis. Your feedback is most helpful and appreciated. Die Jahre in KOM wären nicht so angenehm gewesen ohne die freundliche Hilfe der vielen Kollegen. Zuallererst danke ich Gisela, Jan, Karola, Marion, Moni, Sabine, Sandra S. und Liselotte, für ihre Hilfe, ihren Einsatz und die gute Atmosphäre. Dem Admin-Team, Tomas, Frank, Silvia und Max, danke ich auch, dafür dass eine Infrastruktur bereitstand, die stets funktionierte und nur die üblichen Nachteile für Client/Server Lösungen aufzeigte.

Mein Dank gilt auch besonders den Kolleginnen und Kollegen von KOM, mit denen die Zusammenarbeit viel Freude bereitet und hochqualitative Ergebnisse entstehen lässt. Ganz besonders danken möchte ich André, Apostolos, Dieter, Doreen, Julian, Lasse, Markus, Matthias[2], Michael, Nick, Nico, Parag, Philipp, Tomas, Robert, Sebastian, Stefan und Stephan. Der Peer-to-Peer Gruppe bin ich dankbar für die gute Zusammenarbeit, die guten wissenschaftlichen Ergebnisse und die Freundschaft. Ich danke Osama für seinen Humor, Sebastian für seine Motivation und Konstantin für seine gute Küche. Ganz besonders bedanke ich mich bei Dominik, Christian und Julius, durch deren Arbeit und Wissen, meine Dissertation nicht in diesem Zustand hätte vervollständigt werden können. Auch danke ich euch, dass ihr gute Freunde und immer zur rechten Zeit da seid. Sandra und Patrick möchte ich nicht nur für die tolle QuaP2P-Zeit danken, sondern auch für die tiefe Freundschaft und die besonderen Momente die Jahre über. Den KOM-Veteranen Abed, Carsten, Ivica, Krishna, Matthias, Nick, Nico und Oli danke ich für die guten Ratschläge und Anregungen. Nico besonders für seinen unermüdlichen Einsatz.

Was wäre eine Promotion ohne die Betreuung von studentischen Arbeiten und die Inspiration durch neue Ideen. Ich danke meinen Studenten für ihren Einsatz, ihre Ideen und wünsche ihnen auf ihrem Weg viel Erfolg und Glück. Ich danke meinen Studenten Mu, Kyra, Inna, Song, Daniel, Sergey, Burkhard, Dominik, Christian, Thomas, Moustafa, Florian, Luciana[2], Ho Soo, Vitaliy, Jun, Jiawei, Andreas, Yevgen, Ruben, Ling, Yue, Jonas, Moritz, Julius, Michael, Carsten, Christoph, Hoang, Damian, Alexander, Lyudmil, Johannes, Stephan, Adeniyi und Shengtian. Ferner danke ich allen Studenten, die an LifeSocial mitgearbeitet haben: Sergey, Luciana, Leo[2], Florian, Andreas[2], Tilo, Yevgen, Daniel, Florian, Martin, Immanuel, George-Petru, Tim, Simon, Holger, Hendrik, Benedikt, Benjamin und Martin.

I thank the melodic black metal band Dimmu Borgir, especially Shagrath and Silenoz, for their emotional and powerful music that carried me through several dark moments and inspired me with the beauty and opulence of their music. I also thank the GNU community for creating a world in which high quality tools are available for free. My full gratitude goes to my mates Cherie and Annabel, which helped to turn my thesis in well readable and consumable Australian English. Ich danke Marko für die schöne Schul- und Daniel für die wunderbare Studienzeit, ohne euch beide wäre das Leben weniger bunt, Zeit zum gemeinsamen Wandern müssen wir noch finden.

Doch was wäre eine Arbeit ohne den Rückhalt und die Unterstützung der Familie? Meinen Eltern und meiner Familie bin ich zu größtem Dank verpflichtet, weil sie mir die Grundlagen schufen in einer sicheren Umgebung meine Ziele zu verfolgen. Kedves szüleim, ti megadtátok a lehetöséget, hogy biztonságos, nyugodt környezetben fejlödjek elképzeléseim szerint. Én pedig céljaimat valóra váltottam, úgy, hogy büszkék lehessetek rám.

Schließlich danke ich meiner Frau Filiz, die mich mit Glück erfüllt und mit der ich die Momente des Lebens genießen kann. Danke für deine Geduld und Liebe.

CONTENTS

Part I

INTRODUCTION AND BACKGROUND

In this part, we discuss current issues of peer-to-peer (p2p) systems in terms of their ability to provide reliable quality of service. We contend that next generation p2p systems benefit from a monitoring approach for the coordinated gathering and dissemination of system-specific and peer-specific monitoring information. This helps both users and system providers to validate the quality of service provided by the p2p system. In addition to monitoring, we also contend that the quality of p2p systems should be managed through a system provider in a distributed and autonomic manner, as this helps to tune and control the quality provided by a p2p system. Chapter 1 delineates these concepts and describes the contribution of the dissertation in this field. Further, in Chapter 2 an overview of related research fields is provided; specifically we introduce p2p systems, further information technology architectures in this context and discuss quality of service in the context of p2p systems. We introduce the terminology used in the dissertation and motivate the problem statement of monitoring and managing the quality of service of p2p systems.

1 INTRODUCTION

Ignorance more frequently begets confidence than does knowledge: it is those who know little, and not those who know much, who so positively assert that this or that problem will never be solved by science.
- Charles Darwin

Be the change that you want to see in the world.
- Mahatma Gandhi

In today's globalized, interconnected society, the Internet is a communications and information providing platform that can be accessed at any time of the day. The Internet provides a plethora of services and applications which outperform traditional stand-alone applications in three ways: Internet-based applications are typically free, they are accessible from any device with Internet connectivity, and they also allow interconnections among users and the sharing of information. The Internet has become the main platform for computer applications.

Traditionally, the client-server paradigm is dominant for commercial and non-commercial Internet applications. This paradigm assumes that a single *server* or a set of interconnected servers provides all the resources required for hosting the application. All users, termed *clients*, connect to this server and interact with it. They store and retrieve data, operate on the stored data and may also interact with each other indirectly via the server. A typical application is a wiki or a web page, which consists of a Web server and several clients accessing it.

The centralized paradigm provides controlled quality of service (e.g. using [Ber07]) and is easy to deploy, which explains its wide usage for commercial applications. However, its main limitation lies in its operational cost, which grows with the number of users. A single server may not provide enough resources to manage client requests, thus requiring application providers to provide services through server-farms, such as Google or Yahoo, through a distributed set of servers, such as Akamai, or through servers-on-demand, such as Cloud, which eventually is an intelligent server-farm as well. The client-server paradigm scales only with scaling operational costs.

The costs for the centralized platform may outdo the profits made by a business. This was the case for YouTube [You] until they were bought by Google in 2006 and for Facebook until they announced in December 2009 that, with 350 Million users, their services had turned profitable. Providers of high quality applications, which are available for free, must drastically minimize costs. Application providers, looking for an IT architecture to base their application on, may explore alternative approaches.

One communication paradigm that became very popular in recent years promises to cut the operational costs for the application provider to a minimum and to be naturally scalable: the *peer-to-peer (p2p) paradigm*. The p2p paradigm assumes that clients volunteer the resources necessary to create and maintain the infrastructure which provides the desired functionality they use. Peers are autonomous in their online behavior and heterogeneous in their capacities, connectivity and interests. This autonomy and heterogeneity creates several challenges in the creation of reliable and high quality systems.

Napster [CG01] was the first success for the p2p paradigm. This application emerged in May 1999 and allowed users to exchange music files directly between each other from one personal computer (PC) to another, instead of uploading and downloading them to and from a server. Since the advent of Napster, several p2p applications for file sharing (e.g. KaZaA, BitTorrent), Voice-over-IP (e.g. Skype), video streaming (e.g. Zattoo, P2Plive) and Web (e.g. Freenet) have emerged, leading to an overall traffic consumption of 60-80% of total Internet usage for p2p-based traffic, according to [GDS+03].

The p2p paradigm utilizes the resources of user devices to create an infrastructure that enables the application to operate. The paradigm is scalable, as additional users add more resources to the network and help in resolving the demands and queries of other users.

Despite all the advantageous characteristics of p2p systems in comparison to client-server-based approaches, commercial applications based on completely distributed p2p platforms are rare. Due to the distributed character of such p2p-based approaches, with the wide range of resource providing peers,

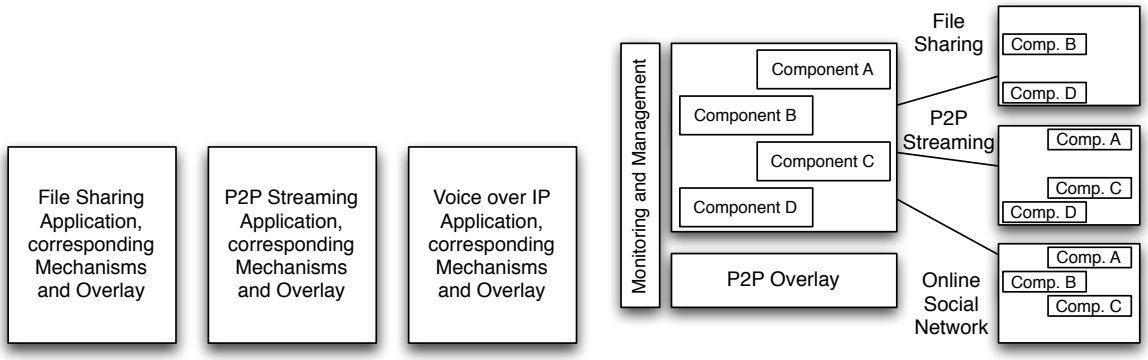
it is very challenging to provide a controlled level of quality of service in the absence of a centralized monitoring and coordinating entity. Additionally, it is very challenging to identify the capacity resources of p2p networks and to provide a reliable reservation of these resources in order to create a reliable platform for services and applications.

In conclusion, application providers using a p2p-based platform for their applications may benefit from the low operational costs and intrinsic scalability of the p2p paradigm. However, the need for controlled quality of service in a p2p application and for the reliable utilization of the resources in the p2p network becomes crucial for the paradigm to also be successful in a wide set of commercial applications.

1.1 MOTIVATION

Until now, many p2p applications have been created from scratch by designated p2p application providers, such as KaZaA [Sha], Skype [Sky04] and Zattoo [Zat]. Designing an application from scratch for a specified scenario results in an optimized stand-alone application for this given scenario. As the mechanisms involved are tightly cross-optimized, the resulting quality also aims towards the same quality goal defined by the scenario. We sketch an example of these kinds of monolithic applications in Figure 1a. This approach for the creation of p2p applications is very time consuming as individual components are not reused.

An alternative for the creation of p2p applications can be found in a modular, component-based approach. The components encapsulate dedicated functionality and allow for reusable, plug-able services. Such components could be p2p overlays like FreePastry [Ric] or JXTA [TAA+04], replication modules like PAST [DR01], multicast modules like Scribe [RKCD01] and so on. We sketch a component-based p2p platform in Figure 1b. Through the reuse and combination of existing components, applications can be created rapidly and their behavior can be predicted more easily. However, only few such modules based on p2p research exist in the field, and standardized interfaces are missing. This is in contrast to other IT architectures, where reusability is a main software engineering goal.



(a) Monolithic Application Architecture

(b) Component-based Application Toolkit

Figure 1: Monolithic and Component-based System Development

In the field of client-server architectures, several individual components can be identified. Components like databases (e.g. Oracle [Ora], mySQL [Sun]), Web servers (e.g. Apache [Apa96]), content management systems (e.g. Typo3 [Ska02]), Wikis (e.g. Mediawiki [Wik02]) and many more are common and simple to combine. The components in the field of client-server architectures can be easily used to build new applications, such as personalized web pages with wikis and forums. In the field of p2p systems such an approach would be desirable as well.

Assuming that further research in the field on p2p systems will lead to a component-based toolkit for the creation of p2p applications, we must face the question of the quality of service of such p2p applications. An application provider may state requirements on the quantitative parameters, and thus the behavior of the system. It may define boundaries for the response time of the application and the

application's maintenance protocol. A system may also state quantitative requirements on the resources provided by the peers, such as the need to provide storage space within a certain period of time. In order to provide a controllable quality of service for these well-defined requirements, considering the autonomy of peers, several challenges must be resolved in the p2p system.

Quality of service (QoS) is defined by Schmitt [SW97] as

"the well-defined and controllable behavior of a system with respect to quantitative parameters."

The behavior of the system can be described in two ways:

- **System view**: The main task of the p2p system is to provide a platform for an application in a quality that fits the expectations of the users and the system providers. Quantitative parameters describe the system status, such as response time, number of participating peers, total protocol overhead and load distribution.
- **Peer view**: The main task of participating peers is to provide resources for the system in a reliable manner, so that mechanisms and applications can operate on these resources. Quantitative parameters describe the status of the resources, such as available storage space, bandwidth capacities and CPU power.

Currently, in p2p systems, these views are not generally available. Typically they are, if at all, only acquired and used in a very specific context of a single mechanism. One main challenge in the context of a general view is that the system behavior and resources in the p2p system have to be monitored in order to decide whether the well-defined requirements regarding the system behavior are met. Second, in the case of a QoS violation, the system must react accordingly in order to restore the desired behavior of the system or the resource provision. In the following, we motivate the monitoring and the management of the quality of service as well as the reliable reservation of resources in p2p systems in more detail.

Monitoring System- and Peer-specific Information in P2P Systems

In Figure 2a we depict a p2p network with dedicated information on the topology, system state and peer resources. A single peer has only a limited view of this knowledge, which is depicted in Figure 2b.

Our goal with regard to monitoring in structured p2p systems is:

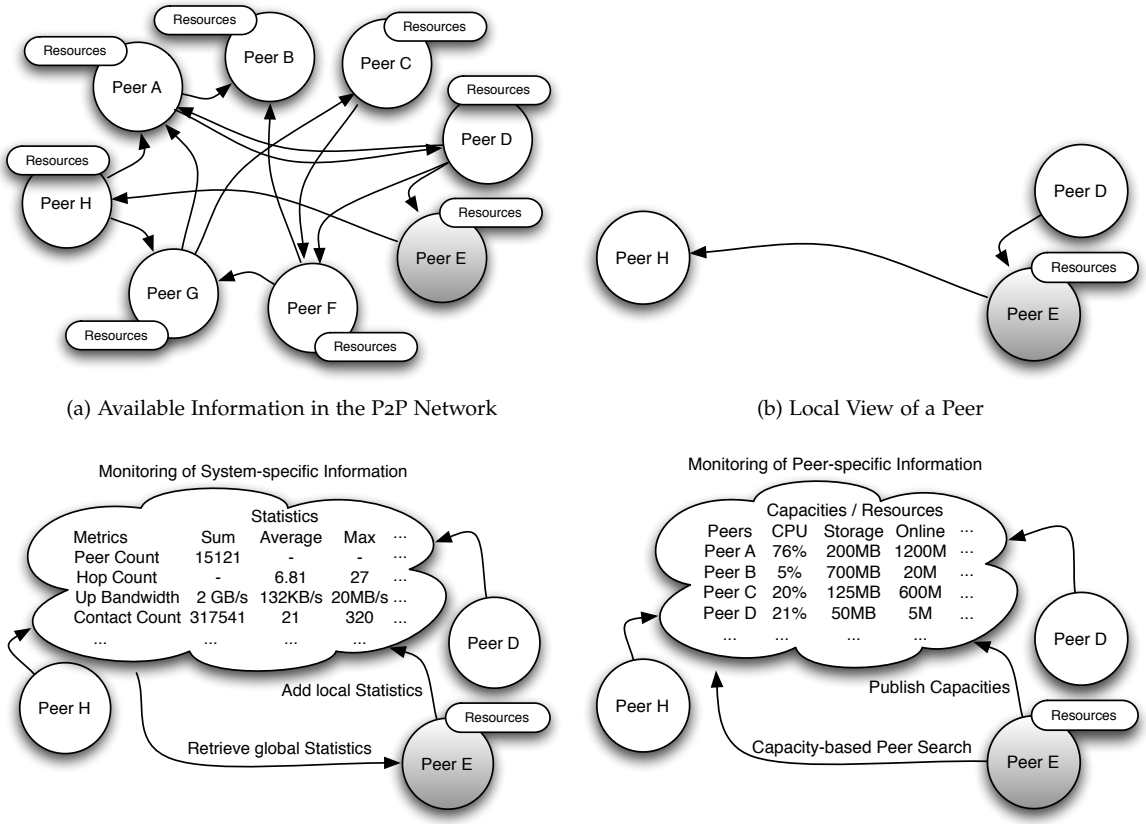
- To obtain a global statistical view on the quality of service provided by the p2p system itself. This global view, as depicted in Figure 2c, should be gathered over all peers in the p2p network and disseminated to all peers as well.
- To gather peer-specific information in the network to create an overview of the available resources in the p2p network and to provide the function of capacity-based peer search for this information, as depicted in Figure 2d.

There are several benefits of monitoring the status and resources of the p2p system. First, quality violations are detected and misleading trends can be identified. Second, based on the monitoring information, new decisions can be made which counteract quality of service violations.

A monitoring solution for p2p systems should be as general as possible in order to be applicable to as many p2p overlays as possible. The main challenges for the establishment of a monitoring architecture are that it needs to be as overlay agnostic as possible, robust against churn, simple and lightweight. Simplicity combined with effectivity is the key to rapid deployment and acceptance in the community.

The system view should cover all peers and contain a large set of metrics about the p2p system, such as the number of nodes and upload bandwidth utilization, in a statistical representation, including the average, minimum and standard deviation.

Peers should report their capacities to the system, these information are gathered and used to provide the function of capacity-based peer search. This function handles queries for a given number of peers with given capacity requirements and provides a set of peers matching these requirements for every query.



(a) Available Information in the P2P Network

(b) Local View of a Peer

(c) Monitoring System-specific Information

(d) Monitoring Peer-specific Information

Figure 2: Monitoring in Peer-to-Peer Systems

Managing the Quality of Service of P2P Systems

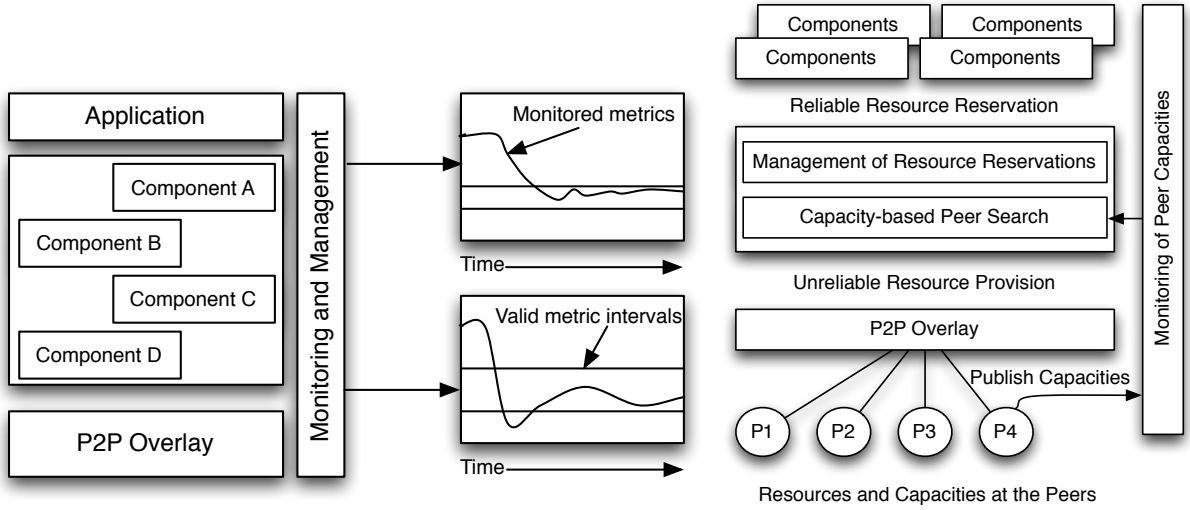
Assuming that a p2p application consists of one to several individual components, with each containing several parameters to configure, it is very difficult or impossible to find a single, static configuration that results in a desired quality of service for all environmental factors. We sketch in Figure 3a the goal on managing the quality of service of p2p systems.

A p2p system, consisting of one to several p2p components, needs to be adequately configured to provide the desired quality of service in the context of the following environments:

- P2P application (e.g. video on demand streaming, distributed data backup). For example, a p2p application for data backup may use the same components like a p2p application for multimedia streaming. However, the first application would require high availability and tolerate high transaction delay while the second application would require low transaction delay and jitter. An adequate configuration needs to be found for both applications.

- Scenario (e.g. mobile peers, strong peer heterogeneity). It describes the given resources for the application, including the number of peers, their capacities and connectivity. A p2p system operating in a working environment with well connected desktop PCs would require a different configuration than when operating on low capacity devices interconnected over the Internet.

- Peer behavior (e.g. access patterns, churn behavior). It defines the characteristics of the workload, such as the distribution and access of the working documents, as well as the frequency of peer arrivals and departures. A network with frequent churn would require, for instance, more frequent checks for the activity level of peers. In particular, the dynamics of peer behavior require changes to the configuration of the p2p system during its runtime in order to meet quality goals.



(a) Managing the Quality of Service in P2P Systems (b) Managing Resource Reservations in P2P Systems

Figure 3: Management of Peer-to-Peer Systems

Although each component may have been individually tested in various scenarios and optimized settings have been deduced, when combined, the dynamics of the scenario and the resulting quality may differ. The main challenge arises from the fact that p2p components are used and operated in a dynamic environment.

Managing Resources Reservations in P2P Systems

Our goal with regard to the reliability of resources provided in a p2p system is to enable p2p systems to explicitly search for certain peer capacities in the p2p network and to reserve these resources reliably. With this functionality it is possible to operate mechanisms and services on top of these resources and to thus provide a reliable platform for mechanisms and services in the p2p platform.

In order to do so, we identify two mechanisms that are needed to provide this task. We sketch them in Figure 3b. First, we need a mechanism that provides the functionality of capacity-based peer search, or in other words finding peers with specific capacities in the p2p network. Peers provide resources to the p2p system in the form of data (e.g. documents and objects) and capacities (e.g. CPU power, bandwidth, connectivity, storage space). Current p2p overlays are data-centric, providing functionality to store, retrieve and search for objects in the p2p overlay. They lack the ability to search and reserve the capacities of the peers in the p2p network. A mechanism providing the functionality of capacity-based peer search shifts the focus in p2p systems from the data-centric view to the resources-centric view, allowing peers to look up certain data in the p2p overlay and look up peers with certain capacities as well. Thus, the whole potential of p2p systems is usable.

The functionality of capacity-based peer search allows to find peers with given capacities, but cannot guarantee the provision of these capacities, as providing peers may go offline. A second mechanism is needed to provide a reservation service on the capacities of peers and to overcome the issue of churn.

A resource reservation mechanism allocates a specific amount of resources in the p2p network for a specific amount of time. The quality of this reservation is maintained by the resource reservation mechanism, leading to reliable resources that may be used by further mechanisms and services to provide high-quality functionality. The main purpose of this functionality is to overcome the limitations of the p2p systems, while preserving its advantageous characteristics.

1.2 GOAL

The dissertation aims to overcome the limitations of current p2p overlays and to provide a reliable basis for p2p-based mechanisms and applications. The *main goal* is to demonstrate the feasibility of a

completely distributed monitoring and management of the quality of service of p2p-based systems and the efficient search and reliable reservation of resources in a p2p network consisting of unreliable peers. In order to reach this goal we set the following objectives:

- A mechanism for *monitoring the system status* needs to be devised to observe the quality perceived by all peers and to generate a global statistical view on this. This is challenging as the monitoring component should cover all peers, as well as be lightweight and precise.
- In order to reach preset quality goals of an application provider, a mechanism for *automated quality of service adaptation* in the p2p system needs to be developed. Based on the monitored quality metrics, the system autonomously decides on actions to impact the adaptation of the quality of the system towards the desired system quality goals.
- For the purposes of controlling the resource provisioning of peers, a mechanism for *monitoring peer-specific information* is needed to get an overview on the available resources and to provide the function of *capacity-based peer search* on these resources.
- In addition to this, a mechanism for *reliable resource reservation* is required, allowing mechanisms and applications to reliably request and use a specified amount of resources for a given time.
- As proof of concept, an *application scenario* needs to be elaborated and created for the proposed mechanisms as a component-based next generation p2p application.

All of the above mentioned mechanisms need to be *thoroughly evaluated*, giving deep insights into the parameter study and interdependencies.

1.3 CONTRIBUTIONS

This dissertation provides the following contributions that address the aforementioned goals in three categories: (1) methodologies, (2) mechanisms and protocols and (3) evaluation and feasibility studies.

In the category of *methodologies*, we conduct the following contributions:

- **Problem statement and related work:**
  We motivate and describe the problem statement for reliable quality of service in p2p systems in detail. We further present and discuss the state-of-the-art of the research in the field of monitoring and management of p2p systems.
- **Methodology for benchmarking of p2p-based monitoring mechanisms:**
  We design and present a systematic methodology for evaluating distributed monitoring mechanisms for p2p systems. We describe the scenario, metrics and workload that enable the benchmarking of monitoring solutions and the comparability of the evaluation results.
- **Investigation of the quality of service adaptation strategies in p2p systems:**
  We investigate the possible approaches and design decisions that could impact the quality of service of p2p systems. Specifically, we analyze the effects of the configuration of mechanisms in a p2p system, as well as the scope of the monitoring view on the quality of a system as input for deriving and deciding on behavioral strategies. We motivate that a systematic adaptation of the configuration of the p2p system based on the monitored quality of service of the system allows for an autonomic management of the system's quality of service.

In the category of *mechanisms and protocols*, we first define the functional and non-functional requirements, the assumptions and the design decisions for a mechanism and subsequently present and discuss a mechanism.

The contribution in this category consists of:

- **Monitoring topology on top of structured p2p overlays:**
  This allows for systematic gathering and dissemination of monitoring information between peers. The main focus of the mechanism lies in its fault tolerance and quick adaptation to churn, providing a reliable infrastructure for various monitoring solutions.

- **A protocol for monitoring system-specific information:**
  In combination with the monitoring architecture, the protocol gathers and disseminates system-specific information. The fully distributed monitoring mechanism for structured p2p overlays provides a global view on the quality of service of the p2p system. It is very precise, lightweight, and due to the fault tolerance of the monitoring architecture also very reliable.

- **A mechanism enabling capacity-based peer search:**
  The mechanism gathers peer-specific information in the monitoring architecture (i.e. information on their capacities) and provides the functionality of capacity-based peer search on this information. It provides the functionality of finding peers in a structured p2p network for a given set of requirements on capacities of the peers, such as their storage space or processing power.

- **A mechanism for reliable resource reservation:**
  It is designed for use on top of structured p2p overlays and relies on the previously discussed monitoring mechanisms. It manages reservations for a set of resource requirements within a given time and provides information on the set of peers providing the desired resources during the reservation time. The mechanism guarantees within a parametrizable probability that this set of peers providing the resources, and thus the reservation, will not fail.

- **A self-configuration framework for p2p systems:**
  A mechanism is needed implementing the management of the quality of service based on the observations of the monitoring mechanism according to the quality of service requirements stated by the p2p system provider. Through a self-configuration approach, the system observes its current quality status, checks the validity of the current quality in comparison to the preset quality goals and initiates in the presence of quality violation a configuration change in the system. The mechanism is totally distributed and enables a p2p system to automatically reach and hold preset quality goals using the self-configuration cycle.

In the category of *evaluation and feasibility studies*, we performed systematic, thorough evaluations of the proposed mechanisms and conducted a feasibility study that shows the potential use of the proposed mechanisms. The following contributions are made in this category:

- **Evaluation of the proposed mechanisms:**
  We thoroughly evaluate the proposed monitoring architecture and the monitoring protocols for peer- and system-specific information through simulations, analytical modeling and testbed evaluation. Thus, we present a deep understanding of the behavior of the proposed mechanisms in various conditions and parameter studies. We show that the monitoring architecture results in a low node degree and traffic overhead, as well as a monitoring freshness which is logarithmic to the number of nodes in network. We further demonstrate that the monitoring architecture is very fault-tolerant and robust against churn. Regarding the monitoring of system-specific information, we substantiate that our mechanism is lightweight and very precise. For the mechanism enabling capacity-based peer search, we show that the monitoring architecture adapts to the heterogeneity of the peers and provides quick, valid results.

  We evaluate the mechanism for reliable resource reservation through simulations and show that the reservations can be provided with the envisioned 100% success ratio in a system with churn while having low reservation maintenance costs.

  We evaluate the mechanism for the quality-oriented self-configuration of p2p systems through simulations at the example of Chord [SMK+01]. We show that preset quality goals (e.g. in terms of metric ranges for the hop count in the overlay) are quickly and reliably reached and held. For this, the framework automatically adapts the configuration of Chord nodes (i.e. their routing table sizes). The adaptation reacts instantly and validates the goal of the design of the mechanism.

- **Prototypical design and implementation of an application scenario:**
  In order to show the feasibility of the application range of the proposed mechanisms for monitoring and management of the quality of service in p2p systems, we designed and prototypically implemented a p2p-based application and included the proposed monitoring mechanism. Our application, LifeSocial.KOM, is a p2p-based platform for social online networks.

1.4 OUTLINE

In this chapter, we introduced the issue of unmonitored and unreliable quality of service in p2p systems, presented motivations for solving the corresponding challenges and summarized our contributions in this field. The remainder of the dissertation is structured as follows:

Chapter 2 gives the background to the dissertation. It presents the current trends in p2p research and points out the current approach for the engineering of p2p applications. P2P overlays as main functional modules are introduced and current p2p applications are discussed. Further, a wide set of functional p2p components is presented. We discuss quality in p2p systems, both on the functional and non-functional level. The chapter closes with the discussion of approaches to evaluate p2p research and to provide a comparability of the evaluation of various mechanisms.

Chapter 3 motivates and presents SkyEye.KOM, a mechanism for monitoring system- and peer-specific information in structured p2p systems. First, the functional and non-functional requirements, assumptions and design decisions are discussed. The mechanism provides both a global view on the system behavior to all of the peers and a capacity-based peer search functionality, which allows users and other peers to find suitable peers for given resource requirements. The monitoring component is a main building block to satisfy the goal of the dissertation.

Chapter 4 presents a detailed evaluation of the monitoring mechanism, SkyEye.KOM, as discussed in Chapter 3. First, we introduce the methodology of the evaluation, present the metrics and the evaluation setup and subsequently present the evaluation results based on analytical modeling, simulations and testbed-based evaluation. The evaluation shows that the monitoring solution is both precise and lightweight for the monitoring of system-specific information and that it is precise and fair for the monitoring of peer-specific information.

Chapter 5 focuses on the issue of unreliable resource provisioning in structured p2p systems. It presents P^3R^3O.KOM, a mechanism for reliable long-term resource reservation in p2p system with unreliable peers. For that, we first discuss the requirements, assumptions and the design goal. The mechanism we propose operates on structured p2p overlays and assumes SkyEye.KOM as a monitoring component. Based on these assumptions, it provides the service of reserving resources in the p2p network reliably for a requested time. The solution overcomes the limitations of the peer lifetimes and provides the resources even for much longer periods through an automated redundancy control. The chapter concludes with a simulation-based evaluation and shows that the goals for reliable resource reservation are met.

Chapter 6 focuses on the issue of unreliable quality of service in structured p2p systems. We first discuss approaches for providing quality of service in a p2p system and then present SkyNet.KOM, a self-configuration framework for the quality management of p2p systems. This framework is totally distributed and is able to operate on top of any structured p2p overlay. It uses the current monitoring status retrieved from the monitoring mechanism SkyEye.KOM, and the quality goals given by an application provider to align the quality of service of the p2p system towards the provider set quality goal. The chapter concludes with an evaluation for the use case of Chord. Evaluation shows that the quality management approach enables p2p systems to automatically reach and keep predefined quality goals quickly and reliably, turning the p2p system in a reliable platform with well-defined quality of service behavior.

Chapter 7 presents an application scenario for the proposed mechanisms. The monitoring and management of the quality of service of p2p systems will take a great role in next generation high-quality p2p-based applications, such as online social networks. We present LifeSocial.KOM, a p2p-based platform for social online networks with integrated quality monitoring. We advocate for this application field, present the architecture and interdependencies and show within the evaluation that the component-based p2p application benefits strongly from the proposed mechanisms.

Chapter 8 concludes the dissertation with a summary of the main contributions and gives an outlook on the application scenarios and implications of the dissertation.

2 BACKGROUND AND RELATED WORK

For how many things, which for our own sake we should never do, do we perform for the sake of our friends.
- Marcus Tullius Cicero
It has been my experience that folks who have no vices have very few virtues.
- Abraham Lincoln

In the previous chapter we motivated the problem of monitoring, providing and managing reliable quality of service in p2p systems, both in terms of reliable resource reservation and reliable system behavior. In order to comprehend the challenges arising for solving this problem in a large-scale p2p network, consisting of autonomous unreliable peers, we discuss the background to this topic in this chapter. This chapter introduces the history and classification of p2p systems, specifically p2p overlays. The next section defines the terminology used throughout the dissertation. In addition to p2p systems, we present further IT architecture paradigms and discuss the comparative strengths and limitations of the p2p paradigm for both users and system providers. We discuss the current and possible application scenarios of the p2p paradigm and point out the need for considering the quality of service as well as quality properties throughout the design and operation of p2p systems.

2.1 BACKGROUND ON PEER-TO-PEER SYSTEMS

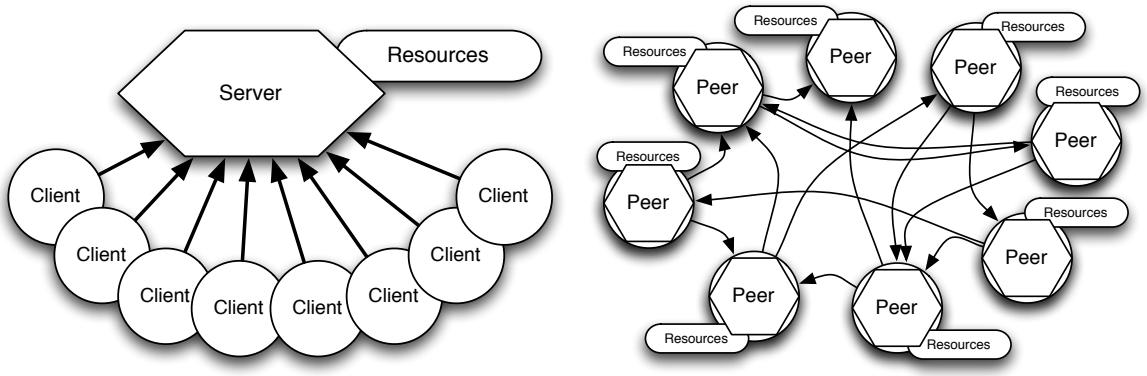
Before introducing a more formal definition of p2p systems, we first describe the broader scope in which they fit. The term *system* is derived from Greek "sýstema" and describes a set of interacting or interdependent entities forming an integrated whole. Systems exist in every research field, describing a union of elements, and thus their context is relevant for the meaning of the terminology. In the field of computer science, various fields with individual system definitions exist. Distributed systems are a dedicated superset of p2p systems, and are defined by Coulouris in [CD88] as follows:

> *A distributed system is one which components located at networked computers communicate and coordinate their actions by passing messages.*

Thus, a distributed system is the set of networked computers which interact through messages. Typically, they coordinate their actions to follow a common goal and have clear role assignments providing various functionality in the system. A p2p system has additional requirements for participating networked computers, Steinmetz and Wehrle listed the requirements in [SW05b]:

- P2P overlay: Peers organize themselves in an overlay on top of the Internet and thus create a self-organizing p2p network. This network typically provides lookup or search functionality.
- Heterogeneity of the peers: The peers are assumed to vary in the capabilities, connectivity and the online behavior. They join and leave the p2p network autonomously, a behavior termed *churn*.
- Twofold roles of a peer: Peers offer resources autonomously to the other peers and use the resources of other peers in the p2p network in a coordinated fashion. Thus, peers are both servers and clients at the same time.

The main aspects of self-organizing p2p networks, churn and heterogeneity, and the ambivalent role of consumer and provider characterize p2p systems as a subclass of distributed systems. We depict in Figure 4 the twofold role of a peer in comparison to other popular IT infrastructures. The figure views the location of resources (square objects) and of consuming nodes (round shape). In the p2p system, these roles are combined. We term the topology and the overlay network created by the peers as a *p2p network*. An *overlay network* is a network (e.g. p2p overlay) on top of another network (e.g. Internet),



(a) Client - Server Architecture

(b) Peer-to-Peer Architecture

Figure 4: Overview on the Client-Server and Peer-to-Peer Paradigm

termed *underlay network*. The overlay abstracts from details in the underlay and creates a topology with virtual links that may be resolved through operations in the underlay. Often, overlays also use an ID space and routing scheme independent of the underlay. We summarize as *p2p system* the mechanisms, application and protocol stack on each peer as well as the p2p network and set of available resources. A *mechanism* is hereby a dedicated functional component with clear interfaces and functionality. For example the specific solution Chord [SMK+01] is a mechanism implementing the functionality of a structured p2p overlay. In order to do so in a distributed system, the overlay mechanism implemented in the protocol stack of a peer comprises one or several *protocols* which define rules for messaging with other peers in the network. For example a routing protocol describes how to forward a given message to a dedicated destination peer and is part, for example, of the Chord mechanism. However, further protocols for overlay maintenance are also part of Chord. The set of mechanisms in a p2p system form a *p2p platform* which encapsulates all mechanisms below and defines a basis functional interface that can be used for various applications. The p2p system also comprises the *p2p application*, which is built on top of a p2p platform and provides the main functionality interface to the user such as a file sharing application or an online social network.

In the following sections, we describe the various types of mechanisms that have evolved in the history of p2p research, and point out their diversity and functional range. We point out the fields in which p2p mechanisms are elaborated and the usefulness of modular components for the creation of p2p systems. We also motivate that the quality of a p2p system is relevant for monitoring, managing and challenging in the presence of several mechanisms in a p2p system.

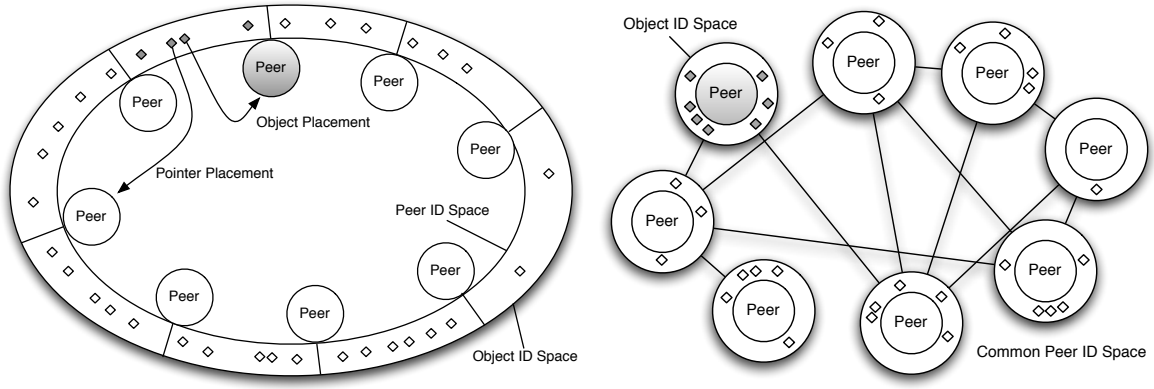
2.1.1 *Peer-to-Peer Overlays*

Certain aspects of the p2p paradigm, especially the twofold role of a peer, have been popular concepts since the advent of the Internet as ARPANET [DAR69] in the late 1960s. At that time, only a few relatively powerful machines were interconnected, sharing resources and consuming services from each other. The dominant applications until the late 1980s were mainly Email [Cro82], Telnet [PR83] and FTP-based file transfers [PR85]. When the Internet became accessible to general society members, less powerful machines were interconnected and as a result a partitioning of the roles in the network was initiated. Some powerful machines acted as hubs and servers of different services, whereas the majority of the nodes in the network was just consuming as clients. The *client-server paradigm* with the two dedicated roles and assumptions of the capacities of the server eventually became the dominant paradigm of the Internet, leading to the invention of the *World Wide Web* (WWW) [BL91] by Sir Tim Berners-Lee in 1989. The WWW replaced Email as the main application of the Internet, and with the opening of the Internet for commercial purposes, it initiated a large growth of Internet users. In the case of the WWW, the client-server paradigm was used in order to provide a reliable service quality through a Web server to handle the numerous client requests.

In 1999, Shawn Fanning developed Napster [CG01], an application for sharing music files over the Internet. Although a server is used in Napster to maintain the index of offered music files and allow users to search for these, the files themselves are transferred directly from one client to another client. This application is cited as the first p2p application, as the bandwidth resources of the clients are used to serve the content to other users. The clients act as servers and clients at the same time, are heterogeneous and are interconnected through the Napster server; therefore, they almost fulfill the characteristics of peers in a p2p system. From 2000 onwards, the research in the field of p2p started to provide several kinds of p2p overlays aiming at eliminating the centrally host index and distributing the functionality to search for specific content and ID-based routing among the peers.

Gnutella [Cli02] was the first decentralized file sharing network. The direct node-to-node or peer-to-peer file transfer is similar to that in Napster, but the index information of the available files is not centrally managed. Instead, the peers form a mesh network, an overlay on top of the Internet, that is aimed at creating a random graph. In this network, the peers may state queries for specific files or keywords and initiate broadcasts to their neighboring peers which forward these, resulting in flooding the network up to a specific hop count range. Peers that have matching files for the query propagate their contact information back along the route of the flooding message. Gnutella provided the functionality of keyword-based search for objects that are located at the peers which actually induced the objects. It is classified as an *unstructured* p2p overlay, as no specific rule or structure is applied to assign objects to specific peers. Gnutella became very popular, but eventually collapsed due to the large traffic overhead on each peer generated through the flooding protocol. In order to overcome this limitation, the design decision of a *hierarchical* topology was picked up, introducing *super-peers*, a second layer of peers providing the functionality of a distributed index. Normal peers are connected to super-peers with strong capacities, which maintains an overlay for the super-peers. Queries are stated to super-peers, which flood the queries in the small overlay of capable super-peers. P2P overlays following this approach are Gnutella 0.6 [KM02] and FastTrack [LKR06].

Chord [SMK+01] is one of the first and the most cited structured p2p overlay, it implements a structured storage of objects on peers. Although the objects may be stored at the inducing peers, a referencing link is stored at another peer in the p2p overlay, which is identified based on the identifier (ID) of the object to be stored. This structure enables the storage and lookup of objects based on their IDs. Thus, structured p2p overlays aim at storage and fully retrievable lookup of objects or their references. The two functions of distributed setting and resolving of a key/value pair define a *Distributed Hash Table* (DHT). Here the key is an object ID and the value is either the reference to the object or the object itself. However, a generalization of these two functions is the function of ID-based routing to a peer responsible for a given object ID. Messages containing objects, object references and lookup queries may be then sent to this peer. We depict the difference between structured and unstructured p2p overlay in Figure 5. It is characterized by the management of the ID space by the peers. Chord implements the functionality of a structured p2p overlay by building a ring topology of peers that follows the order of



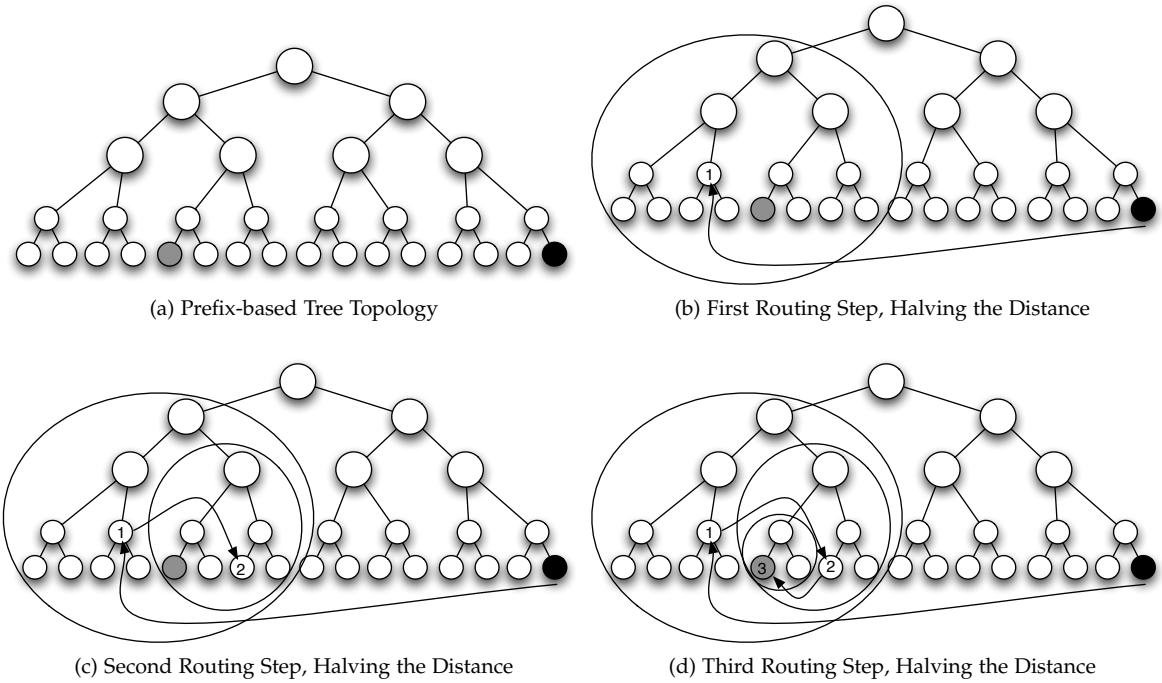
(a) Structured P2P Overlay

(b) Unstructured P2P Overlay

Figure 5: Difference of Structured and Unstructured P2P Overlays

the peer IDs. Objects are assigned to peers that have the next largest ID in the ring. In order to speed up the routing in the ring, Chord also uses "fingers", which form short cuts in the ring. Each peer maintains a set of fingers; the set is of logarithmic size (with the basis 2) in relation to the number of IDs in the overlay and the fingers point to IDs in exponentially increasing distance to the power of 2. Thus, in the routing protocol the distance to the destination ID is halved in every hop, leading to a hop count of $O(\log N)$ for a lookup with N being the number of peers in the overlay.

Since the time Chord was introduced, several other structured p2p overlays were proposed. Pastry [RD01], Tapestry [ZKJ02], P-Grid [ACMD+03] and Kademlia [MM02], in contrast to Chord, relax the rules for adding contacts to the routing table, and build a routing table for each distance range, which grows exponentially. Thus, a closer ID distance range is small (e.g. $[2^3, 2^4[$) and has the same number of contacts as a distance range that is further away (e.g. $[2^{100}, 2^{101}[$). Such a routing table is calculated for each peer and used to route messages based on their ID to peers closest to the specific ID. The proposed overlays vary in their maintenance protocols, distance metrics, load balancing approaches and the roles of the peers as described in [Dar05]. In Figure 6, we sketch a distance-based routing protocol similar to that in Kademlia, which routes with logarithmic message and delay overhead. Karl Aberer gives in [AAG+05] a mathematical analysis and classification of the design steps needed to build a p2p overlay. The steps comprise the choice of the ID space, mapping resources and peers to the ID space, and the management of the ID space by the peers. Also relevant is the structure of the overlay, as well as routing and maintenance strategies.



(a) Prefix-based Tree Topology

(b) First Routing Step, Halving the Distance

(c) Second Routing Step, Halving the Distance

(d) Third Routing Step, Halving the Distance

Figure 6: Distance-based Routing in Structured P2P Overlays

The main function of p2p overlays is thus for unstructured p2p overlays, the keyword-based search for objects (i.e. files), and for structured p2p overlays, the DHT function of ID-based storing and lookup of objects in a p2p network. The main application for p2p overlays since their emergence has been file sharing, which allows users to search and retrieve files free of charge. Several applications with this function became popular, including Morpheus, KaZaA, LimeWire, eMule and many more, and resulted in p2p-related traffic becoming the dominant traffic on the Internet. It accounted for between 60% to 80%, as stated in [SGG02] and [GDS+03].

2.1.2 *Content Distribution and Streaming*

In 2003, Bram Cohen proposed in [Coh] BitTorrent as a novel approach for content distribution networks. Previously, files were distributed in total and shared only after successful download. In BitTorrent a file is split in smaller chunks, allowing for exchanging previously downloaded chunks during the download of other chunks. One main contribution of BitTorrent is the tit-for-tat policy that determines the contacts of a peer with which it exchanges chunks and aims at accelerated downloads. In BitTorrent, a tracker is typically used to maintain the list of peers which are uploading and downloading chunks of a specific file. However, the corresponding tracker is implemented in a Kademlia DHT [MM02]. This shows how well two components, one for the optimized distribution of content and the other for the retrieval of relevant tracker information, may be combined.

BitTorrent provides a mesh-based architecture for content distribution. It is also used as a basis for video streaming applications, while adapting the chunk selection strategy to meet the playback deadlines. While BitTorrent applies the rarest-first strategy for chunk selection to keep files available in the swarm, the mesh-based approach is effective in video on demand applications where chunk selection strategies are applied that prefetch chunks related to the playback position. For live streaming applications, multicast trees are typically deployed in order to systematically distribute the content. Here, a server injects the original live content in the multicast tree, while the peers adapt their positions in the tree to optimally propagate the content to all nodes. Measurements on BitTorrent widely appreciated in the community and presented, for example, in [KRT+09]. Content distribution infrastructures gained large attention lately, discussion arose whether the Internet itself should be content-centric (see [PGM+06]).

Skype [Sky04] has gained large popularity in recent years as a chat application supporting text messaging, as well as audio and video streaming. However, in contrast to the previously mentioned video streaming applications, streaming in Skype only takes place between the peers involved in the conversation and scalability goals regarding the number of viewers are not addressed. Skype-like architectures [MYGRM09] aim at resolving this limitation while using the successful hierarchical overlay idea of Skype (e.g. [MYRGM08]).

2.1.3 *Further Functional Components*

Besides the traditional components for distributing large amounts of data, further functionality on p2p basis has been proposed in literature. Next, we present a wide set of functionality to emphasize the trend and potential of having various dedicated functionality components in p2p systems.

STORAGE AND REPLICATION

With regard to the data-centric characteristics of most p2p applications, research on reliable and available distributed storage based on the p2p paradigm has been conducted since the birth of p2p systems. While in unstructured p2p overlays data objects are not deployed proactively in the network, in structured p2p overlays they are. In order to maintain the availability of this data in the presence of churn, the data is replicated on several peers. PAST [DR01], for example, extends the p2p overlay Pastry [RD01]. It provides the functionality to store and retrieve data from the p2p network, while maintaining the replication rate of the data. OceanStore [KBC+00] is a prominent distributed storage solution, which inherits the overlay functionality. For regarding the quality of availability (see [On05]), a more dedicated layer for replication (e.g. [DKK+01]) that solely focuses on maintaining the availability of the objects stored in the p2p network, facilitates usage on a wide set of overlays, which may be optimized for specific environments.

PUBLISH / SUBSCRIBE

A publish/subscribe mechanism enables network nodes to asynchronously exchange messages through a notification service that is in charge to maintain subscriptions for a channel or topic and forwards messages to subscribed nodes once a notification is published. Channel- or topic-based publish/subscribe mechanisms as in [PB03, RKCD01] aim at providing an asynchronous communication channel. Content-based publish/subscribe mechanisms (e.g. [AX02, LC08, TBF+03]) aim to deliver content and notifications to subscribers based on their interests without the need to subscribe to dedicated channels.

Here, the challenge arises from matching the published notifications to the interests of the subscribers in a distributed manner.

The presented approaches marked the beginning of p2p-based publish/subscribe mechanisms. They also use an underlying structured overlay to maintain the subscriptions. As an additional component in p2p systems they enable further mechanisms to communicate asynchronously, i.e. to retrieve messages that were sent while the addressed user was offline.

APPLICATION LAYER MULTICAST

Application layer multicast has been discussed for long time [BBK02] as a function in p2p systems, supporting the routing of messages or objects to a dedicated set of peers. The involved peers maintain a group membership protocol themselves, and a protocol is used to construct the multicast delivery topology, replicate the messages of the multicast and send them systematically, typically in a tree structure, to further peers in the multicast set. The tree may be created first, like in Narada [CRZ00] or Scattercast [Cha03], or a more dense mesh network may be established initially, whereby spanning trees are created on demand, like in ALMI [PSVW01] or Host-Multicast [ZJZ02]. An overview on the extensive field of application layer multicast protocols is given in [HASG07].

The functionality of multicast is a useful tool that could be used by further components in p2p systems. For example, in a p2p-based chat conference application, one could use a DHT to look up desired contact partners and set up a multicast communication channel for a conference call. Through the combination of existing components, in this case the p2p overlay and a mechanism for application layer multicast, rapid development of p2p applications becomes feasible.

DISTRIBUTED COMPUTATION

Besides their bandwidth and storage space for data-centric applications, peers in a p2p network also provide their computational power. Since SETI@Home [ACK+02], edge computing has also gained large popularity in general society. In Seti@Home, extra-terrestrial signals are dispatched in small chunks to home computers and analyzed on them for traces of alien intelligence. Boinc [And04] extends this concept to further distributed computing fields. The concept of distributed computing in GRIDs was presented in [MH05].

Architectures for harnessing idle cycles in a p2p network have been initially presented in [LZZ+04, FCC+03]. While in these early works, the authors propose an architecture that combines the overlay and the distributed computation, the authors of [BFHM04, GSS06, MK05] rely on existing p2p overlays and provide a component for distributed cycle and job allocation. The latter approach, enables the use of the computational component in a multi-component platform with richer functionality than a solely computational application.

ACCOUNTING

In order to enable the commercial use of the p2p paradigm, accounting and logging of user contributions have been researched and presented in [HAL+02, LDHM05, HS05]. These approaches operate on a structured p2p overlay and monitor the individual contribution of the peers. Through an integrated reputation scheme (e.g. in [Lie08]), the p2p system is able to take the user contributions into account and refund for these actions. Community-based content distribution networks (e.g. [CGM09]) are positive examples on the potential to create community-centric mechanisms. Micro payment models have been proposed for p2p applications in [YGM03, DG05]. However, although a dedicated accounting mechanism is very useful and well investigated, it only unfolds its potential in use with further components that create an attractive p2p application.

SUMMARY

We introduced the field of p2p overlays and content distribution, as well as additional functional components of a p2p system. The components pointed out a subset of functionality using the p2p paradigm, showing the wide range of possible functionality. The depicted components show small overlap in their functionality and are suitable for being combined. Through a combination of these components, applications can be quickly built that consist of evaluated parts whose behavior is well

understood. For instance, relying on Kademlia as an overlay substrate relieves the burden of other components to maintain a routing infrastructure.

Although each individual component and mechanism may be well understood and evaluated, the effects occurring through the combination of various components cannot be estimated in the design phase of a mechanism. In order to support the monitoring and management of the components in a p2p system and the p2p system itself, a dedicated component for monitoring and management is needed. With this, the potential of well elaborated components can be used in combination with the potential to control the quality provided by the combination of these components

2.2 QUALITY OF PEER-TO-PEER SYSTEMS

Having discussed the functionality provided by various components in a p2p system, next, we focus on the quality of specific implementations of a functionality as well as the quality of the overall p2p system. We first define and describe the term *quality of service* and broaden its scope to include the concept of *quality properties* in the sections below.

2.2.1 *Quality of Service*

The term *quality of service* has been introduced in Chapter 1, where we noted that Schmitt [SW97] defines the term as the well-defined and controllable behavior of a system with respect to quantitative parameters. In the field of telecommunications it is defined in the ITU standard X.902 [Int98] as a set of quality requirements on the collective behavior of one or more objects. The term was shaped in the context of network quality of service, whereby it describes the behavior of routing and communication protocols on the network layer. In a more general scope, quality of service reflects the quantifiable properties of a component or system (i.e. the service quality).

The quality of service of a component or system is described in form of *metrics*. A metric describes a measure of the behavior of the component under test. For example, in a p2p overlay, the hop count, lookup delay, bandwidth consumption and number of peers are metrics that are measurable and describe a property of the p2p overlay. In terms of the functionality of reliable data storage and replication, metrics refer to such as the number of objects per peer, the data retrieval success rate and the average storage load per peer. Each component provides a dedicated functionality, whose performance and costs can be measured. Any measure on the quality is related to a metric, describing the behavior of a component or system in specific. In the dissertation, we aim at monitoring and managing these metrics globally in the system.

Metrics are defined by measures and can only be observed and not directly influenced by the system or the application. They are influenced by the scenario, the workload, as well as the components' configuration. The *scenario* defines the given resources a component can operate on, typically identified by the capacity distribution of the peers. The availability of the capacities of the peers is closely related to the *workload*, which defines both the characteristics of peer presence as well as the access and application behavior of the peers. The *configuration* of the individual components and their interaction in the system affects the quality of the resulting p2p system greatly; typically a trade-off between the induced costs and the resulting system performance is observable. We use the term *parameter* to describe the configurable variables of a component.

In a p2p system, we cannot influence the workload, online behavior of the peers or the scenario. However, the configuration of the components is modifiable, a fact that provides a handy tool to indirectly influence the quality of service of the p2p system.

2.2.2 *Quality Properties*

Besides this general view on components in p2p systems, the design of the components has a great effect on the scope of quality a component can provide. For example, a p2p overlay which is designed and implemented with ineffective stabilization protocols and routing strategies does not provide the

same range of quality of service as a p2p overlay with enhanced protocols and strategies. For the design of p2p mechanisms and components, several quality properties can be aimed at.

Quality properties provide a generalized view on the properties of a p2p mechanism or p2p system. In contrast to quality metrics (e.g. hop count = 10), which describe only a single characteristic of a mechanism within a given scenario, workload and configuration, quality properties describe the characteristics of the mechanism or system over various individual measurements of quality metrics. They consider a subset of all metrics and interpret their changes over a wide set of scenario, workload and configuration setups to deduce statements on the general abilities and characteristics of the mechanism or system being tested. For example, *scalability* is a quality property that describes the ability of a mechanism to provide a desired standard regarding performance metrics and induced costs with increasing number of peers and workload in the system. In Figure 7, we present the considered quality properties that we aimed at in this dissertation, based on previous work [HSL+06] at the department of the author.

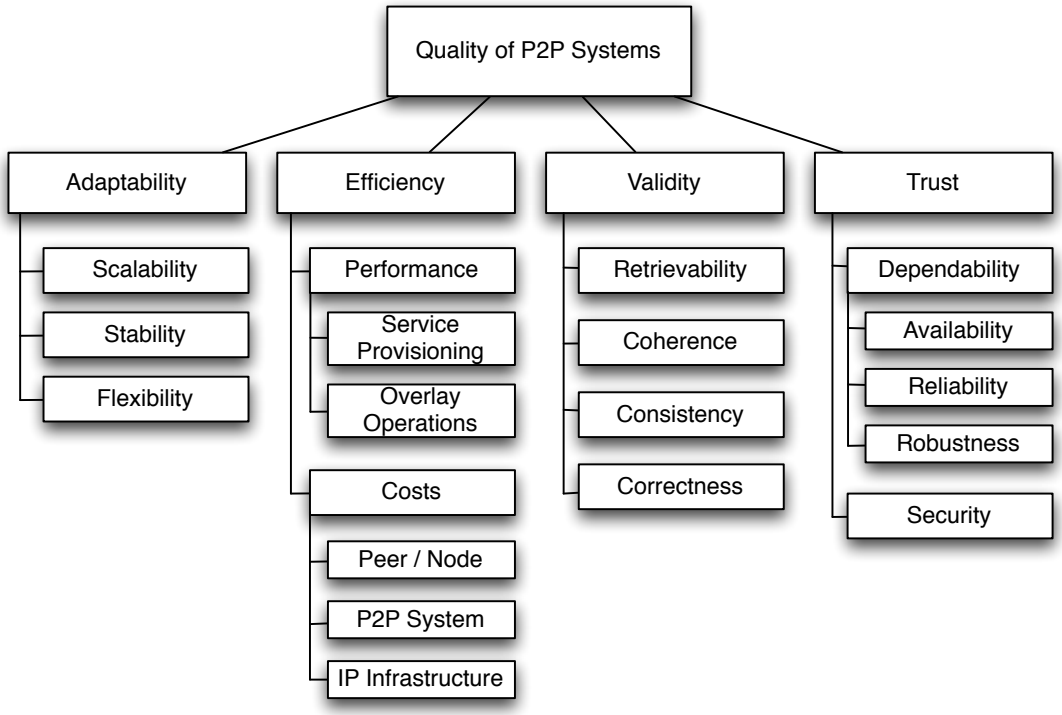


Figure 7: Quality Properties of P2P Systems

Adaptability aims at adjusting the functionality and operation mode of a p2p system to the dynamic modification of the scenario and workload. *Scalability* as a subordinated quality property focuses on the characteristics of a mechanism with increasing numbers of peers in the p2p system. A system is scalable, when it maintains its performance and costs with the increase of the number of peers. The quality property *flexibility* focuses on the adaptation of the p2p system to the context of its use, such as the workload induced by users. In addition, *stability* describes the ability of a system to hold or quickly recover a quality level despite changes to the scenario and workload.

The quality property of *efficiency* focuses on the ratio of the *performance* of a mechanism, on the level of peer, system and network, in relation to the induced *costs*. An efficient p2p mechanism or system aims at identifying the optimum performance-to-cost ratio and maintaining this ratio. At this point, it is already possible to observe the interdependencies and possible conflicts between the quality properties. For example, a flexible mechanism may adapt to the environmental conditions and maintain a certain performance level, but, the induced costs for the adaptation process may drastically increase the costs in the system, leading to a worse performance-to-cost ratio than without flexibility-enabling protocols.

The *validity* of a p2p system considers the properties of the distributed data or information in p2p systems. The property *retrievability* focuses on the availability of the data in the network as well as the

ability to find it. *Coherence* refers to the freshness of information or data being used by individual peers in the p2p network, such as whether it can be updated quickly on all corresponding peers upon a change of the data or information. The quality property of *consistency* focuses on individual copies of a document or information at various peers in the p2p system. Consistency aims at providing all peers in the network the same view of the data stored in the p2p network. Finally, *correctness* extends the scope from a single data object to an interconnected mesh of objects or information. A correct mechanism ensures that the links and dependencies between the documents in the mesh are maintained and considered, even under strong dynamism in the p2p system. Here, some trade-offs and interdependencies are also noticeable. For example, a mechanism aiming at coherency might lead to an inefficient state while not necessarily improving the service quality.

The *trust* in a system arises from its ability to provide dependable and secure services under various conditions. *Dependability* aims at the continuous provision of the functionality under normal conditions all the time (*availability*) and at maintaining of the functionality under rapid fluctuations of the scenario and workload (*reliability* and *robustness*). The *security* issues in a p2p system are related to the misbehavior of the peers and need to be resolved in order to protect confidentiality, integrity, authenticity, non-repudiation and privacy.

2.2.3 *Benchmarking Methodology*

Having discussed the quality metrics and properties in p2p systems, next, we discuss how the quality of various components implementing the same functionality in a p2p system can be compared. Typically in p2p research, evaluation results of specific mechanisms are compared to well-known representative implementations of the same functionality. In the field of structured p2p overlays, for example, the behavior of a new overlay might be evaluated in comparison to the behavior of Chord or Kademlia. However, the evaluation setups and observed metrics are rarely the same, so that two new overlays (i.e. not Chord or Kademlia) cannot be compared to each other. No statements can be deduced regarding the comparison of two new overlays (i.e. which one is better), when one overlay is, for example, evaluated in a large scale p2p network with decent workload and the other overlay in a mid scale p2p network with heavy workload.

The comparability of evaluation results is a general issue in research and has been discussed in other fields as well. As a result, benchmarking methodologies have been deduced allowing for comparable evaluation. A *benchmark* is a standard by which the quality or characteristic of something (e.g. a component or mechanism) can be measured and judged. The evaluation results of all components that are evaluated according to this standard are comparable. The Standard Performance Evaluation Corporation (SPEC) [Kel90] lists benchmarks for several research fields and allows researches and industries to estimate the quality of their components. The SPEC provides and motivates benchmark standards for CPUs [JNJH09], graphics/workstations [BCCW88], Java Enterprise Servers [SKAB09], mail servers [PVM+01], network file systems [Rob99], power systems [SBN06], SIP [Hri06], SOA [SSM+00], virtualization solutions [GPW05] and Web servers [CT02].

In order to enable the comparability of p2p mechanisms, we presented in [KGK+08] an approach for benchmarking p2p overlays specifically and p2p mechanisms in general. By applying a systematic benchmarking methodology in the evaluation of new p2p mechanisms, these mechanisms do not have to be compared to one or several specific similar mechanisms, but within well-defined evaluation setup and metrics. Such approaches were also presented for event-based systems [KSBB08] or ECA rule engines [GSBB08].

A benchmark consists of a tuple of functionality requirements, scenario, workload, metrics and quality properties. A *functionality requirement* is a functional description of the interfaces a component being tested has to provide. For structured p2p overlays, for example, it is the ID-based routing function. The scenario and the workload define, as mentioned before, the environment in which the component operates, such as peer capacity distribution, the behavior of the peers, churn frequency and access patterns. A solution for the functionality requirements operates in this environment and its behavior is measured in quality metrics that relate to its main functionality (e.g. lookup delay for a structured p2p overlay). These numerical measures of individual metrics are interpreted with regard to the quality properties and enable statements on the overall quality of the component being tested. For example,

two solutions for the same functionality may be compared according to several quality properties, such as their efficiency being the relation of lookup delay to the traffic overhead, or their scalability, the slope of the overhead increase in the traffic load with increasing number of peers in the p2p network. Following this systematic methodology enables researches to create comparable evaluation results while evaluating individual p2p mechanisms.

As a main result, we state the requirement that for the evaluation of a p2p mechanism, its main functionality interfaces must be specified, the evaluation setup and metrics well described, and the evaluation outcomes summarized in common characteristic results.

2.3 CONCLUSIONS

In this chapter, we introduced the history of the p2p paradigm and sketched various fields of p2p mechanisms providing different kinds of functionality. We motivated that each individual mechanism provides a specific function that it is optimized for.

The quality of p2p systems is important for their applicability and usefulness in real applications. The quality is made up of the intrinsic ability of each component to provide quality of service in a specific range. The quality properties of a mechanism are basically defined by the design decisions made in the creation of the mechanism. However, the design decisions influence the measurable quality, (i.e. the metrics related to the performance and costs) that the mechanism is able to achieve in various scenarios, workloads and configuration settings. The overall quality of a mechanism is described according to a benchmarking standard that unifies the evaluation methods for a category of mechanisms and thus enables comparative evaluations.

We focus on metrics as the main indicator of the service quality of a component or system. Metrics are influenced by the scenario, workload and configuration of the system. While we cannot influence the first two, the third aspect is able to be influenced. In this dissertation, we aim to provide a global view of the metrics of a p2p system and enable the management of them through an automated self-configuration approach for the p2p system. We further aim to observe the resources available at the level of peers, in order to provide a reliable service for resource reservation that allows mechanisms and components to fully harness the potential of the p2p system. In the following chapters, we introduce problem statements regarding the monitoring and managing of p2p systems, present solutions, evaluations and discuss related work.

Part II

MONITORING PEER-TO-PEER SYSTEMS

In this part, we motivate, present and evaluate SkyEye.KOM, a mechanism for distributed monitoring in structured p2p systems. SkyEye.KOM provides a global view on the p2p system's behavior in terms of a statistical representation. Through a distributed and lightweight approach, the status of all peers is systematically gathered, aggregated and combined to a global view, which is then disseminated to all peers again. Regarding the peer-specific information, SkyEye.KOM gathers and prepares information on the capacities of the peers in the p2p system and provides the function of capacity-based peer search on them. The monitoring solution SkyEye.KOM is evaluated in Chapter 4 thoroughly through simulations, analytical modeling and prototypical evaluation in a testbed. The evaluation shows that the proposed solution is both lightweight and precise in the monitoring of the global system status, as well as load balanced and efficient in terms of monitoring peer-specific information.

3 A MONITORING SOLUTION FOR PEER-TO-PEER SYSTEMS

In circles of dominance, emotional deeps unite, fiction and transcendence woven together
in the essence of purity lies wisdom, join the forces, the spiritual black dimensions.
- Dimmu Borgir
All our knowledge has its origins in our perceptions.
- Leonardo da Vinci

In the previous chapters we motivated monitoring and management mechanisms in p2p systems and gave an overview on the background of p2p quality and evaluation methodologies. In the following sections, we describe the problem statement of monitoring p2p systems in more detail, present our systematic approach to achieve solutions and conclude with an evaluation in the next chapter.

3.1 MOTIVATION

The goal of a monitoring mechanism for p2p systems is to monitor system- and peer-specific information in structured p2p systems and to provide a global view on the system status as well as the functionality of capacity-based peer search. We describe the functional and non-functional requirements in this section. Our solution, SkyEye.KOM, is a mechanism that addresses and fulfills these requirements. The design decisions, requirements and architecture are given in Section 3.2. The main idea of SkyEye.KOM is to establish a tree topology including all peers in the p2p overlay in a deterministic, fault-tolerant way with very low maintenance overhead. The establishment of the tree topology is described in detail in Section 3.3. Each peer locally measures its status and sends periodic update messages to its own parent peer in the tree. The parent peer aggregates or filters the information of its child peers as well as its own information and passes it on to its parent peer. Eventually, the updates of all peers reach the root node, which then has a global view on the p2p system. This global view is pushed to all peers down the tree in the case of system-specific information (e.g. average hop counts). In the case of peer-specific information, such as bandwidth capacities of a specific peer, the information remains in the tree and is used for capacity-based peer search. The protocols for monitoring system-specific information and peer-specific information are described in Section 3.4 and Section 3.5. This chapter closes with the discussion of related work in Section 3.6, conclusions in Section 3.7 and an outlook on the evaluation of the proposed mechanisms in Chapter 4.

3.1.1 *Functional Requirements*

Structured p2p systems are networks of autonomous peers up to a large-scale. The peers are interconnected by participating in a p2p overlay which provides dedicated functionality by the cooperation of the peers. The peers in this network typically not only run the overlay implementation but also a storage and application layer, which in combination we call a p2p system. The problem we address in this section is the monitoring of the peer capacities in a p2p system and the quality of service provided by a p2p system. Next, we delineate the problem statements for these two functions.

MONITORING PEER-SPECIFIC INFORMATION

The goal of monitoring peer-specific information is to create an overview on the capacities offered by each peer in the network in order to make it available for capacity-based peer search. The overview of capacities contains information on peer-specific resources, like available bandwidth, CPU power and storage space, and also additional peer-specific information like the node degree of the peer. Capacity-based peer search is a function provided to the peers in the network to find a desired number

of peers with a set of desired capacities. A query can be stated by any peer and results in a list with the desired number of peers offering the desired capacities and the corresponding peer identifiers.

In order to formalize the desired functionality, we describe suitable interfaces and depict them in Figure 8. Let Att_i be an attribute that describes the capacity of a peer, such as CPU power, upload bandwidth capacity or its online time. Let $Att(p)$ be the set of all attributes a peer p offers. Let $Cons_i$ be a constraint on Att_i in terms of an upper or lower bound, such as $Cons_1 : Att_1 < 50$, describing that the CPU power should not exceed 50%. Constraints may also be empty and are typically not addressing every single attribute. Then following functions are provided by the mechanism for monitoring of peer-specific information, which we visualize in Figure 8:

- **(**$Monitor^{peer}_{function}$ **1):** void publishCapacity($Att(p)$) - publishes the offered set of capacities of a peer p
- **(**$Monitor^{peer}_{function}$ **2):** PeerID-list ← capacity-based-peer-search(n, $Cons_1$, $Cons_2$,...,$Cons_i$) - query for n peers fulfilling a set of constraints

In order to provide these services, the mechanism states following dependencies:

- **(**$Monitor^{peer}_{requirement}$ **1):** Att_i ← getLocalSensor(i) - local measurement of a capacity
- **(**$Monitor^{peer}_{requirement}$ **2):** Structured p2p overlay with KBR-functionality as described in [DZD+03]

Peers communicate local sensor information on their capacities over the KBR-compliant structured p2p overlay. The gathered information is prepared and can be used for capacity-based peer search. The stated dependencies are necessary in order to obtain local information and to communicate it to other peers. The KBR interface provides an access to a wide set of compatible overlays.

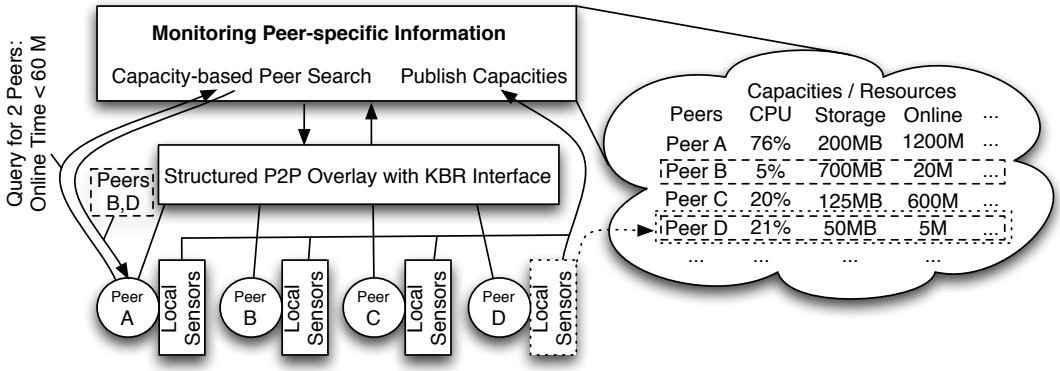


Figure 8: Overview on the Functionality of Monitoring Peer-specific Information

MONITORING OF SYSTEM-SPECIFIC INFORMATION

The goal of the monitoring of system-specific information is to retrieve an exhaustive statistical view on a wide set of metrics on all peers in the network and to disseminate it to all peers in the network. The set of metrics includes the upload and download bandwidth consumption, the hop count, message count, and so on in the network. It is an extendible list, which is common to all peers in the network and contains metrics which are based on local measurements of all peers. A statistical view is created for each individual metric by calculating the average value over all peers, as well as the minimum and maximum, the sum and standard deviation, and the count of considered peers. The statistical view on the metrics is taken over the measurements of all peers in the network, thus leading to a global view on the system statistics. The goal of the gathering of this global view is to disseminate the global view to all peers in the network and thus let them know about the status of the system. It is very useful to see metrics or key performance indicators of a running p2p system (e.g. in order to identify weaknesses, bugs and misleading trends).

In order to formalize the desired functionality, we describe suitable interfaces and present a corresponding overview in Figure 9. Let $m_i(p)$ be a locally measured metric (i) in a peer p, such as the hop

count of all lookups the peer performed. And let further $m_i(P)$ be the corresponding global metric (i) for the set of all peers P, such as the hop count of all lookups of all peers. Let $M(p)$ be the list of all metrics of peer p in a local view and $M(P)$ the list of all metrics of all peers in a global view. Then following services are provided by the mechanism for monitoring of system-specific information:

- **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **1):** addLocalStatistics($M(p)$) - adds the local observations to the global view
- **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **2):** $M(P) \leftarrow$ getStatistics() - retrieves the global view on all metrics in the system

In order to provide these services, the mechanism states following dependencies:

- **(**$\text{Monitor}^{\text{system}}_{\text{requirement}}$ **1):** $M_i(p) \leftarrow$ getLocalMetric(i) - local measurement of metrics
- **(**$\text{Monitor}^{\text{system}}_{\text{requirement}}$ **2):** Structured p2p overlay with KBR-functionality as described in [DZD+03]

The main goal of the monitoring of system-specific information is to obtain a global view on the status of the system **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **2)**, based on local measurements of all peers **(**$\text{Monitor}^{\text{system}}_{\text{requirement}}$ **1)**. For that, the peers are able to communicate their status **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **1)** while using a KBR-compliant structured p2p overlay **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **2)**. Please note that the required dependencies **(**$\text{Monitor}^{\text{system}}_{\text{requirement}}$ **2)** and **(**$\text{Monitor}^{\text{peer}}_{\text{requirement}}$ **2)** are identical.

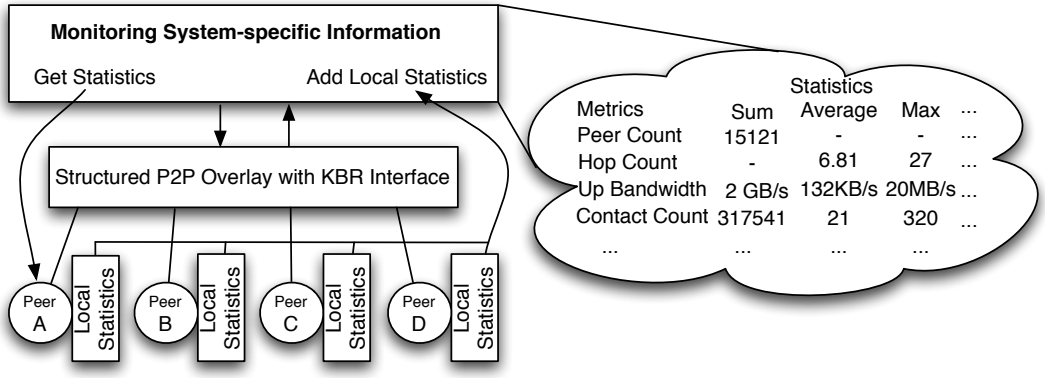


Figure 9: Overview on the Functionality of Monitoring System-specific Information

3.1.2 *Non-functional Requirements*

In the following, we address the non-functional requirements on the monitoring solution concerning its quality properties. While functional requirements refer to the task and interfaces of a solution, non-functional requirements refer to the quality of the solution, such as precision or costs, assuming that the functional requirements are fulfilled.

Scalability
We require that the proposed monitoring solution is scalable with the number of peers. Scalability is achieved in the case that the average and maximum load on the peers is limited and not linear or otherwise related to the number of peers in the network. This requirement is tightly related to the requirement that no peer must be overloaded. Each peer is assumed to have a maximum contribution and load level, which can be individually set and must be respected by the monitoring mechanism. The load generated by the monitoring solution must be controllable, as the monitoring is considered just as an extension of the p2p system and not as its main application.

Stability
The effects of churn cause expected and unexpected failures in the protocol. A stable system quickly

adapts to the new situation and provides a steady performance after a short stabilization phase. Thus, the monitoring solution should be able to cope with churn and show uneven results only for a short time in the presence of churn.

Efficiency
The efficiency of a system is described as the ratio of its performance to its costs. The main performance characteristic for monitoring solutions is the accuracy by which the information is monitored. The precision is represented as a relative monitoring error, which should be small.

The costs of the monitoring solution are measured in terms of the bandwidth consumption of the peers. Other costs, like the computational time used for processing the monitoring information, are not crucial. The solution should provide a low relative monitoring error while consuming only a small fraction of bandwidth.

Freshness
The monitoring view should be as fresh as possible when disseminating the global system view to the peers or using the peer-specific information for capacity-based peer search. Typically, a high freshness (i.e. low information age) of the monitoring data also leads to high accuracy and to higher costs. The ratio of performance gain to cost increase needs to be reasonable, while considering the freshness of the monitoring information.

Consistency
The monitoring information presented to the individual peers may vary strongly in terms of accuracy and freshness. Consistency demands that the relative difference in the monitored information among the peers is small.

Reliability
The monitoring information may cover various scopes of the p2p system, thus leading to an incomplete view. However, a reliable monitoring mechanism for system-specific information requires that all peers in the network are covered. Regarding the monitoring of peer-specific information and the provision of capacity-based peer search, we require that a desired set of peers must be found if there are a sufficient number of peers fulfilling a queried capacity.

3.2 DESIGN AND COMPONENTS OF SKYEYE.KOM

Having discussed the functional and non-functional requirements of a monitoring solution, in this section, we introduce our solution: SkyEye.KOM. We first discuss the design decisions and assumptions for SkyEye.KOM and then present the functional components of SkyEye.KOM. The solution is introduced stepwise, first the monitoring topology and then the protocols for monitoring system-specific and peer-specific information in the p2p system.

3.2.1 *Design Decisions*

In the following section, we discuss the various design decisions for creating a monitoring topology. Hereby, we follow the engineering guideline for p2p overlays as described in [Kov09]. We motivate a new layer for monitoring that creates a tree topology on top of a p2p overlay. Using the tree topology, monitoring information is gathered and disseminated in a structured and coordinated manner. To conclude, we discuss the assumptions made on the p2p overlay and the behavior of peers.

CENTRALIZED VS. DISTRIBUTED

A simple monitoring approach assumes a centralized approach in which all peers register at a server and send their monitoring information periodically. Although this approach is efficient and precise, the costs are unbalanced and this may lead to scalability issues. The server has an intrinsic limit on how many peers it can handle and thus reaches this limit sooner or later. A model for this effect is presented in the evaluation section in Subsection 4.3.4. A distributed approach on the other side does

not have capacity limitations, as the capacities increase in the p2p system with every joining peer. Thus, we follow a distributed approach.

INTEGRATED VS. NEW LAYER

A monitoring solution for p2p overlays can be integrated both in a p2p overlay or a dedicated component, which is overlay independent, using general overlay interfaces. While the first approach could reuse the overlay infrastructure in more detail and thus result in lower maintenance costs, it is still limited to only one overlay. Additionally, extensions for the same overlay from different authors may be incompatible. Introducing a new layer on top of the p2p overlay for SkyEye.KOM requires general p2p overlay interfaces, such as KBR [DZD+03], which provides a new interface that is independent of the overlay functionality. Thus, functionality is separated, easier to maintain and the monitoring solution is more generally applicable on all p2p overlays which comply to the assumed p2p overlay interface.

MONITORING TOPOLOGY

Various topologies can be used to build a monitoring infrastructure. The purpose of the monitoring topology is to connect all peers and allow for a coordinated information flow so that the information of all peers can be gathered in one spot to create a global view. The following homogeneous topologies have been considered as candidates for the monitoring infrastructure:

- Bus: The bus topology connects all peers over a shared medium. This kind of medium does not exist in p2p systems, where only direct communication from peer to peer is possible.
- Ring: A ring topology connects all peers in a circular row. While the system-specific information can be passed and aggregated in the ring, one loop involving all peers takes $O(N)$ hops, which results in bad performance in terms of monitoring freshness.
- Star: A star topology connects all peers to a dedicated node. This single node may act as a monitoring server and receive and aggregate the information of all peers. This is a client-server-based approach with high load unbalance when comparing the server and the clients. Our goal is to provide a distributed solution for p2p overlays sharing all of the load.
- Mesh: A mesh topology is an uncoordinated network of peers which follows no specific form. It is very robust against churn and easy to maintain. However, a mesh topology does not support the purpose of a monitoring infrastructure, which requires the monitoring information to flow in a structured way in order to have a fresh monitoring view and a minimal traffic overhead. However, several solutions exist which follow a gossip-based approach in a mesh topology.
- Tree: A tree topology locates the peers in a structure with fixed node degree and inherent support for the monitoring infrastructure. For all peers involved in the topology, there is a path to the root of $O(\log(N))$ hops. Furthermore, as the node degree is fixed, with each peer having a single father node and a fixed size of children nodes, information from the children nodes can be aggregated and sent to the father node. Finally, all peer information arrives at the root, which can then form a global view on the network. The tree topology offers a controllable overhead due to the limited node degree.

We have chosen to use a tree-based topology as it provides fresh monitoring results while keeping the load on the peers bounded through the limited node degree.

POSITION ASSIGNMENT IN THE TREE

For any tree structure operating on a set of available nodes, one has to choose which peers to assign to which positions in the tree. The position assignment depends on the load a node in the tree has to handle; more loaded positions should be covered by more capable peers. On the other hand, it is costly to find capable peers and to disseminate their position in the tree.

We face a trade-off between picking capable peers and having low topology maintenance overhead. A low maintenance overhead can be reached by picking peers in a deterministic fashion independent of their capabilities. Picking peers by their capabilities requires maintaining and announcing of the peer positions by various protocols, in order to identify and be able to address messages to peers in a

specific tree position. We chose to combine the benefits of both ideas. To do so, we first create a core tree based on a deterministic position assignment. This core tree is used to find capable peers which are assigned on demand for a period of time as *Support Peers* for the deterministically chosen inner tree nodes. Support Peers fulfill the role of assisting the regular, but weak nodes in the tree.

We gather on the one hand aggregatable system statistics that do not vary in their size. Thus, the load generated by statistics related to a set of peers is independent of the size of the peer set. The root as well as an inner node have the same load. These observations lead us to the conclusion that for monitoring system statistics, the capacities of the peers in the tree are not relevant. We pick the peers for the core tree deterministically based on peer identifiers.

In the case of monitoring unaggregatable peer-specific information the size of the information grows with the size of the monitored peer set. Deterministically chosen peers being responsible for this information may be overloaded. In that case, they may pick a more capable Support Peer using the gathered peer-specific information. These Support Peers take over one part of the load and support the deterministically picked peer in its duties for a specified period of time. Thus, the core tree is enhanced in the weaker positions by capable Support Peers, resulting in a strong support tree.

REQUIREMENTS ON THE UNDERLYING P2P OVERLAY

The assumptions stated on a solution define its dependencies, such as the scenarios in which the solution can be applied. General and few requirements lead to less dependencies for the monitoring mechanism and thus to a broader application range (e.g. structured and unstructured p2p overlays). However, more specific requirements lead to limited application scopes, but typically also to more efficient solutions, as the assumptions are used beneficially

Furthermore, we assume that the peers in the overlay are autonomous, going online and offline according to an uncoordinated, individual pattern (e.g. arrive and leave in a Poisson process). This results in a great challenge for the creation of a monitoring mechanism, as no peer is expected to be reliable. We also assume that peers are not misbehaving by modifying, deleting or forging messages and monitoring information.

In order to deploy SkyEye.KOM on top of a p2p overlay, we define a set of interfaces and functions that the underlying overlay must provide. We assume the existence of a structured p2p overlay with DHT functionality. This assumption is made to have a common interface, on which the proposed mechanism can operate. For SkyEye.KOM, two dedicated functions must be provided by the structured p2p overlay:

- Function 1: `Boolean isMyKey(key K)`
- Function 2: `void route(key K, message M, nodehandle hint)`

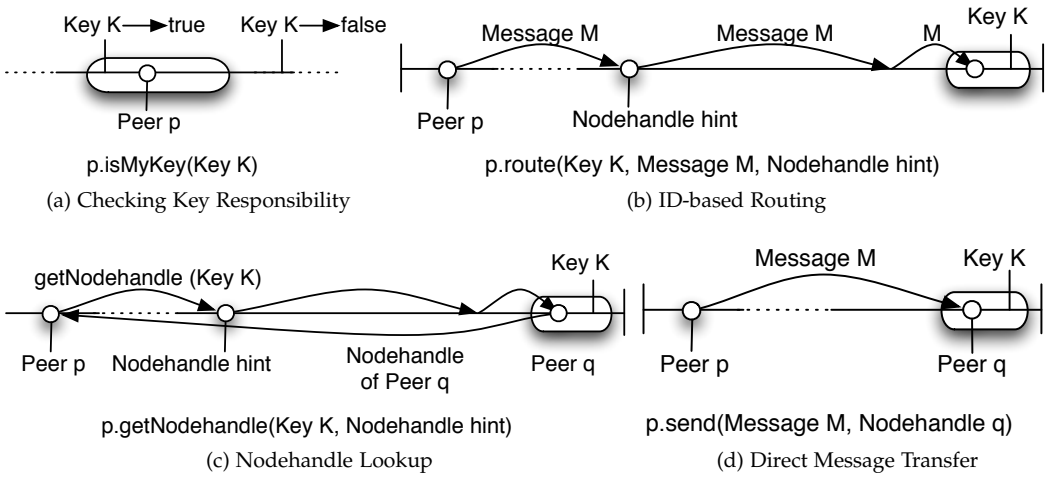
The second function can also be replaced by two individual functions:

- Function 3: `nodehandle getNodehandle(key K)`
- Function 4: `void send(message M, nodehandle P)`

We depict these assumed functions in Figure 10. Function 1 allows to check whether the local peer is responsible for a given key. We further assume, but do not require, that the used overlay resolves this function based on local information without sending messages. The scope of every peer should be maintained and kept up to date by the overlay, allowing quick and cheap checks for the responsibility for object IDs.

Function 2 makes it possible to send messages to peers responsible for a given key. Here typically the routing functionality in the structured overlay is used. Optionally, a next hop in routing can be defined by setting a hint. This function was initially described as part of the common API for Key-based Routing in [DZD+03]. The function enables a node in SkyEye.KOM to send a message to a node which is responsible for a specific key in the structured p2p overlay. This key may represent a role in the p2p network, for example, a deterministically chosen peer responsible for monitoring the peers in a given ID range. Dependable routing is a main criteria in many communication systems, wired networks are considered as dependable (e.g. the Internet), for wireless networks several approaches for dependable routing exist as well, such as [Holo5].

Function 3 can be used to look up the contact information of the peer responsible for a given key. The term *nodehandle* describes the Internet address and connection point of a peer. And Function 4 makes it possible to send a message to a dedicated peer, of which the nodehandle is already known. The Function 2 is a summarized function created with the Function 3 and 4, which are easier to optimize in the specific structured p2p overlay. Please note that having the Function 2 provided by the structured p2p overlay is sufficient for SkyEye.KOM. Current structured p2p overlays either already fulfill these requirements or can be easily extended to do so.



(a) Checking Key Responsibility (b) ID-based Routing

(c) Nodehandle Lookup (d) Direct Message Transfer

Figure 10: Assumed Functions provided by the Structured P2P Overlay

ASSUMPTIONS

While a solution may be designed to be very efficient and provide both the functional and non-functional goals, some assumptions are still made that describe the context of the solution. All peers have sensors implemented which provide them with current information on their hardware status, such as their CPU and memory utilization, as well as p2p system-specific information. This information is at the p2p overlay level describing such as the hop count needed for lookups, the lookup load, routing table sizes and so on. On the p2p application level, further information may be obtained, such as the number of files retrieved and stored, as well as the popularity of the files.

We assume that all peers in the overlay have an application installed running both the structured p2p overlay and our monitoring solution, SkyEye.KOM. Every peer receiving a SkyEye.KOM related message must be able to process it. Every peer is able to define a personal maximum load willing to contribute to the monitoring layer. This load limitation is related to the monitoring of peer-specific information. The maintenance and monitoring load for system-specific information is very similar for all peers, so the peer-specific load limit is not considered here. We assume that the offered bandwidth is at least large enough to cover the maintenance information. We further assume that the peers act compatible to the protocol and are not malicious. The peers do not try to forge, modify or delete monitoring information.

3.2.2 *Components of the Architecture*

In this subsection, we present the architecture of SkyEye.KOM. Design decisions addressing the monitoring protocols are discussed in the corresponding Sections 3.4 and 3.5. In order to keep the architecture extensible and easy to customize, it follows a modular approach, separating the functionality in components. The architecture is depicted in Figure 11 and described in the following.

Topology Module

The Topology Module establishes and maintains the topology of SkyEye.KOM and decides which peer to send the monitoring information to. Having a dedicated topology module allows for the adoption of

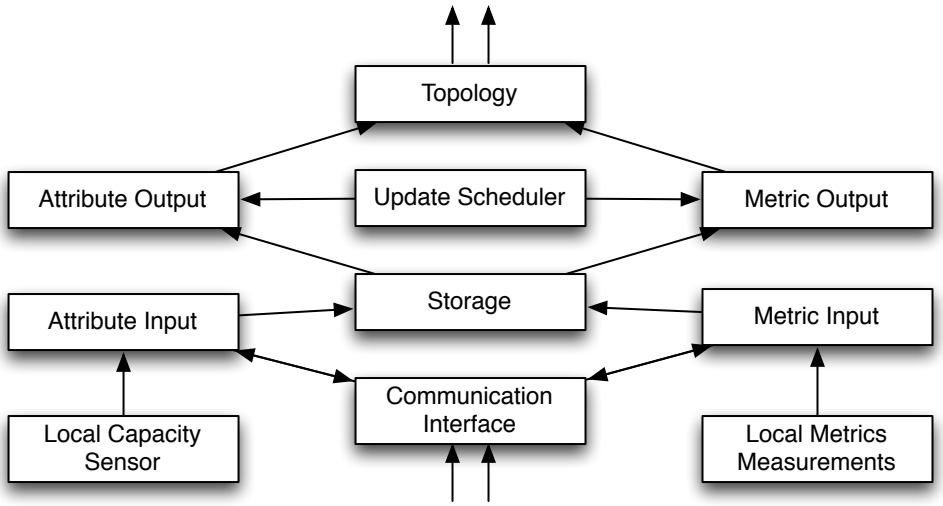


Figure 11: Overview of the Components of SkyEye.KOM

SkyEye.KOM to various kinds of overlays. Furthermore, the Topology Module provides access to the communication interfaces, allowing messages to be sent to peers.

Sensor Module
The Sensor Module measures the local metrics in a peer, considering both system-specific metrics and peer-specific capacities. The measured information is stored in the Storage Module.

Storage Module
The Storage Module offers an interface for storing and retrieving both local and system-wide information. It acts as an information container that can store sensor information and incoming information from other nodes. All outgoing information is based on the information set in the Storage Module as well.

Statistics Input Module
Both the system statistics and peer capacity information are separately managed. Both of them have a module for incoming information, which pre-process or refuse the incoming data. For the system statistics the Statistics Input Module aggregates the incoming information about the system to a new global view on the system.

Statistics Output Module
This module is tightly related to the receiving module for system statistics. The Statistics Output Module retrieves the local statistics and received statistics from the Storage Module, aggregates them and prepares an information update to be sent to another peer.

Capacity Input Module
The Capacity Input Module handles the received capacity data and stores it on the local node using the storage service. It implements a congestion control mechanism, deciding on overload through incoming messages and communicates the load limits to the message senders.

Capacity Output Module
The Capacity Output Module is responsible for processing the peer's capacity output. As the capacity information is not able to be aggregated, it grows in size with the number of monitored peers. The Capacity Output Module prepares the capacity information considering the maximum list size of the receiving peer.

Update Scheduler Module
While the output modules just prepare the monitoring data, the actual transmission is scheduled by the Update Scheduler Module. It takes care that the update messages are sent in a configurable frequency at the right time.

Having described the individual modules, in the following sections, we describe the creation and maintenance of the monitoring topology, as well as the protocols for monitoring system- and peer-specific information.

3.3 ESTABLISHING THE CORE MONITORING TOPOLOGY

In this section, we describe the idea and creation of the core topology of SkyEye.KOM. In the next sections, we present the protocols that describe how the topology is used to gather and disseminate monitoring information. In order to introduce the core topology of the proposed monitoring solution, SkyEye.KOM, we give a model of the p2p system and the monitoring solution in it. The model describes how SkyEye.KOM creates a new tree-based overlay on top of the p2p overlay below. Here, we use and elaborate the assumed dependency **(**$\text{Monitor}^{system}_{requirement}$ **2)** and **(**$\text{Monitor}^{peer}_{requirement}$ **2)** for providing a structured p2p overlay.

Let P be the set of all peers and S_{OID} be the identifier space of the overlay. The overlay ID space contains both peer identifiers (ID) and object identifiers (ID,keys). All peers have a peer ID ($p \in P$), so that $P \subseteq S_{OID}$, the set of all object identifiers is equal to S_{OID}. In a DHT, object identifiers are linked to peer identifiers. We denote the function $owner$ assigning object IDs to peer IDs. For every object ID i, a corresponding peer is assigned.

$$\forall i \in S_{OID} : !\exists p \in P \; with \; owner(i) = p \tag{3.1}$$

A peer is always responsible for a convex set of object identifiers, called the responsibility range or *scope* of the peer.

$$\begin{aligned} &scope(p) : P \rightarrow \wp(S_{OID}) \\ &scope(p) := owner^{-1}(p);\ p \in P\ ; scope(p) \subseteq S_{OID} \\ &\forall p, q \in P :\ p \neq q \Rightarrow scope(p) \cap scope(q) = \{\} \end{aligned} \tag{3.2}$$

The structured overlay provides the function *Boolean isMyKey(key K)* that tells whether the local peer p is responsible for a given key K or not. For $p \in P$ and $K \in S_{OID}$, following counts:

$$\texttt{Boolean p.isMyKey(key K)} = \begin{cases} true & if \quad K \in scope(p) \\ false & if \quad K \notin scope(p) \end{cases} \tag{3.3}$$

For the consistency of the formulae, we also add:

$$p.isMyKey(key\ K) \Leftrightarrow K \in scope(p) \Leftrightarrow owner(K) = p \tag{3.4}$$

The second assumed function is *void route(key K, message M, nodehandle hint)*. It is used to route a message M to a peer p with $scope(p) = K$. As a next hop in routing, the peer $hint$ should be used if set. The third function is *nodehandle getNodehandle(key K)* which returns the contact information of a peer p.

$$getNodehandle :\ S_{OID} \rightarrow P :\ \ getNodehandle(key\ K) = owner(K) \tag{3.5}$$

The forth assumed function, *send(message M, nodehandle P)*, sends a message M directly to the nodehandle P. Having introduced the nomenclature used in the description of the p2p overlay, next, we introduce the monitoring layer on top of the p2p overlay.

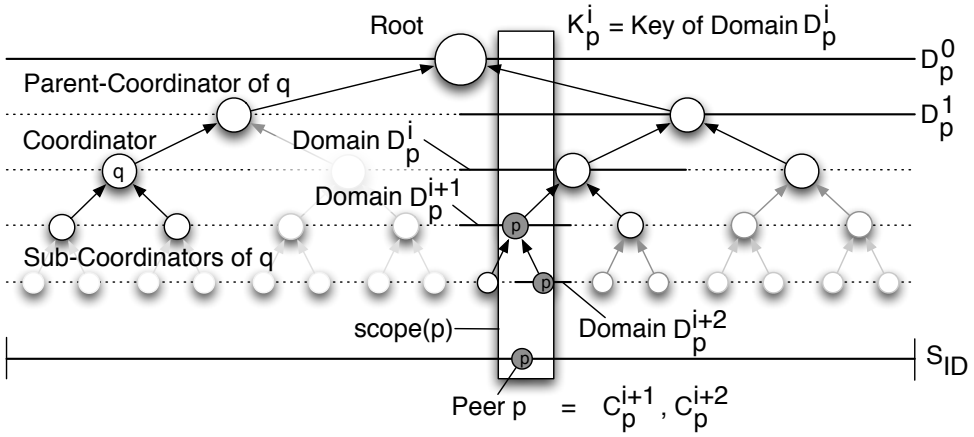


Figure 12: Definitions used in SkyEye.KOM

A UNIFIED ID SPACE

SkyEye.KOM should be usable on all structured p2p overlays fulfilling the assumptions specified in this chapter. SkyEye.KOM reuses the routing and ID-mapping functionality of the underlying structured p2p overlay. Each structured p2p overlay comes with its own overlay ID space $S_{OID} \subseteq \mathbb{N}$ (e.g. 0 to $2^{160}-1$ in Chord [SMK+01], 0 to $2^{128}-1$ in Pastry [RD01]). In order to have an ID space in SkyEye.KOM independent of the overlay, we propose a unified ID space $S_{ID} \subseteq \mathbb{R}$ that ranges from 0 to 1. It can be parametrized in its granularity by defining minimal steps from 2^{-128} to 2^{-160} or even less. Any ID used in current structured overlays can be mapped to this interval, for example by dividing the overlay ID by the size of the overlay ID space. The benefit of floating-point numbers is that the granularity may be varied while keeping the distribution of the identifiers in the ID space. In contrast to that, an extension of an integer ID space would lead to an unpopulated new range.

Let S_{OID} be the overlay ID space and $IDmap$ the function mapping from S_{OID} to S_{ID}. The mapping function should retain convexity of identifier subsets and be linear in relation to the responsibility function:

$$\begin{aligned} &\forall i \in S_{OID} : \ IDmap(owner(i)) = owner(IDmap(i)) \\ &\forall p \in P : \ IDmap(scope(p)) = scope(IDmap(p)) \end{aligned} \tag{3.6}$$

This requirement on the mapping function allows easy mapping of the responsibility areas of a peer in both ID spaces. To give an example for Chord:

$$IDmap_{Chord} : [0, 2^{160}-1] \rightarrow [0,1), \ i \rightarrow i/2^{160} \tag{3.7}$$

In the unified ID space S_{ID} of SkyEye.KOM we reuse, by remapping of the IDs, the routing functionality of the underlying structured p2p overlay and the responsibility function $isMyKey$. Please note that with the introduction of the unifying ID space we operate in an "over-overlay", which is applicable on any unifiable DHT.

DEFINITION OF DOMAINS

In order to aggregate the information of individual peers, we establish a tree structure on top of the p2p overlay, in form of an "over-overlay". The tree is built by recursively segmenting the ID space S_{ID} in intervals (which we call *Domains*) and assigning a responsible peer to each Domain, which we call *Coordinator* of the Domain. The depth of the tree is $O(\log N)$. Each level of the tree stores in union the information of all peers in the ID space, but with increasing tree depth the information is shared with more peers. A Coordinator is in charge to maintain the information of all the peers, whose IDs are in its Domain. An overview on the used terminology, is given in Figure 12. In the following introduction of the terms, we refer to this figure.

We define a *Domain* as a continuous interval in the ID space S_{ID}. Domains at the same level l in the tree do not overlap, they form a partition. Let $p \in S_{ID}$ be a peer, then the sequence of Domains

containing peer p are labeled D_p^l with l as level counter. The root of the tree is at level 0, thus the following counts:

The Domain of the root covers the whole ID space, thus all information retrieved from this Domain results in a global view.

$$\forall p \in S_{ID} \ : \ D_p^0 = S_{ID} \tag{3.8}$$

A peer ID p, for which a Domain is calculated, is always part of the Domain.

$$\forall l \in \mathbb{N}, \ \forall p \in S_{ID} \ : \ p \in D_p^l \tag{3.9}$$

The same counts also for the peer for all levels in the tree.

$$\forall l \in \mathbb{N}, \ \forall p \in S_{ID} \ : \ D_p^{l+1} \subseteq D_p^l \tag{3.10}$$

Thus, the term D_p^l identifies for a peer p Domains in which the peer is part of at every level l. The recursion ends in two cases. First, with a Domain being identical to the predecessor Domain, typically by having only the ID p in the ID range:

$$\exists k \in \mathbb{N}, \ \forall l \in \mathbb{N} \ with \ l \geqslant k \ \forall p \in S_{ID} \ : \ D_p^{l+1} = D_p^l \tag{3.11}$$

In practice this case does not occur, as the second termination criteria is mostly active. The calculation of the Domains of a peer stops at the point when the scope of the peer covers the whole Domain:

$$\forall p \in S_{ID}, \ \exists k \in \mathbb{N}, \ \forall l \in \mathbb{N} \ with \ l \geqslant k \ : \ D_p^l \subseteq scope(p) \tag{3.12}$$

Each Domain is maintained by a Coordinator, the Coordinators of the various Domains, eventually, establish the tree by sending each other information updates. Next, we conceptualize how the Coordinators identify the peers to which they establish connections to in the tree topology. In the core tree, each peer p identifies, using the deterministic function, the Coordinators for the Domains D_p^l that contain the peer's ID p. A Domain of level l (e.g. $[i_a, i_b]$) is partitioned in β (Sub-)Domains of level $l+1$ (e.g. $[i_a, i_1], [i_1+1, i_2], [i_2+1, i_3], ..., [i_{\beta-1}+1, i_b]$). Please note that by this process, the Domains and not the peers build a b-tree structure. The parameter β is the branching factor of the tree.

We map Domains to peers, that become then Coordinators of the Domain, using the responsibility function *owner*. A specific ID in the Domain, called *Domain Key*, determines the corresponding Coordinator by the responsibility function. We use a mapping function K to map Domains to their Keys, K has to fulfill Eq. 3.13.

Let K be the function mapping a Domain (subset of S_{ID}) to an ID (Domain Key) in S_{ID}, let $\wp(S_{ID})$ be the power set (set of all subsets) of S_{ID}, and let K_p^l be the key of the Domain D_p^l, then following holds:

$$\begin{aligned} &K : \ \wp(S_{ID}) \to S_{ID} \\ &\forall p \in S_{ID}, \ \forall l \in \mathbb{N} \ : \ K_p^l := K(D_p^l) \in D_p^l \end{aligned} \tag{3.13}$$

In order to compute the Domain Key K_p^l at level l a peer p first calculates the ID range of the Domain D_p^l by using the following two formulae. Here, the variable β denotes the branching factor of the SkyEye.KOM tree and ID_p the ID of the peer p:

$$min(D_p^l) = \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l} \tag{3.14}$$

$$max(D_p^l) = \frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} \tag{3.15}$$

Afterwards, the Domain Key K_p^l can be calculated. For the ease of representation, we present a simple function for K, taking K_p^l as the middle value of a Domain D_p^l. We calculate the IDs K_p^l, for which a peer p tests its responsibility, for every level l:

$$\begin{aligned} K_p^l &:= min(D_p^l) + \frac{max(D_p^l) - min(D_p^l)}{2} \\ &:= \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l} + \frac{\frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} - \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l}}{2} = \frac{\frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} + \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l}}{2} \end{aligned} \tag{3.16}$$

Now, we can define for every level l in the tree the Coordinator C_p^l of a specific Domain D_p^l. Let D_p^l be a Domain, then its Coordinator C_p^l is a peer $p \in P$ and defined as

$$C_p^l := owner(K_p^l) \tag{3.17}$$

This means, the Coordinator of the Domain D_p^l, which is in the l^{th} level and contains the ID p, is defined as the peer which is responsible for the key K_p^l in the unified ID space and the mapped ID space of the underlying DHT. Please note that we use the index p in C_p^l only to identify the Domain D_p^l for which the Coordinator C_p^l is responsible.

DEFINITION OF COORDINATORS AND PARENT COORDINATORS

Every peer p in the network identifies a single (Parent-)Coordinator C_p to which it periodically send its monitoring information called *update*. Each peer p in the network may be a Coordinator of a Domain D_p^l and receive updates. These updates are then periodically sent in the network to the Parent-Coordinator C_p^{l-1} one level higher.

Now, we discuss how a peer identifies its position in the tree and its parent node in the tree, which is its Parent-Coordinator. We refer to the Figures 13a and 13b. In order to identify its Parent-Coordinator C_p to which p has to send its individual peer information, peer p calculates the Keys K_p^* of the Domains D_p^* it is in. For instance peer $0.88 \in S_{OID}$ calculates the Domain Keys including 0.5, 0.75 and 0.875. It checks whether it is responsible for one of these Domain keys. In Figure 13a, this is the case for peer 0.88 and Domain Key 0.875. A peer p may be Coordinator for several Domains on different levels in the tree. We observe this effect at peer 0.74, which is responsible for the Domain Keys 0.75 and 0.8125. This comes from the fact that DHT responsibility area obtained with $scope(p)$ is an interval, which may contain some of the Keys K_p^* of the Domains D_p^* the peer p is in. Here, we use the postulated DHT-function $isMyKey(keyK)$, so that a peer can determine, whether it is responsible for an ID in S_{ID} or not. The first (and thus largest) Domain it is responsible for defines the position of the peer in the tree. Thus, the peer with the ID 0.74 is Coordinator of the Domains corresponding to the Domain Key 0.75 and 0.8125, but finds it position in the tree in the upper Domain (i.e. corresponding to the Domain Key 0.75).

The peer p identifies the level of its largest Domain and with that the Parent-Coordinator C_p one level higher, which is its parent node in the tree. The largest Domain a peer is responsible for is at level $level_p^{min}$, which is calculated as follows:

$$level_p^{min} := min(i \in \mathbb{N} \text{ with } K_p^i \in scope(p)) \tag{3.18}$$

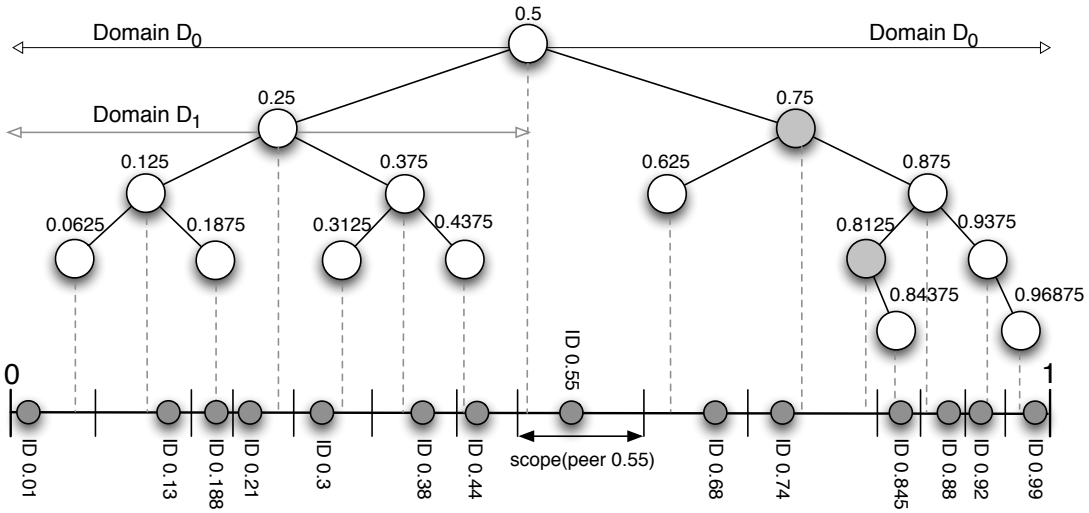
The Parent-Coordinator C_p of peer p is then

$$C_p := C_p^{level_p^{min}-1} = owner(K_p^{level_p^{min}-1}) \tag{3.19}$$

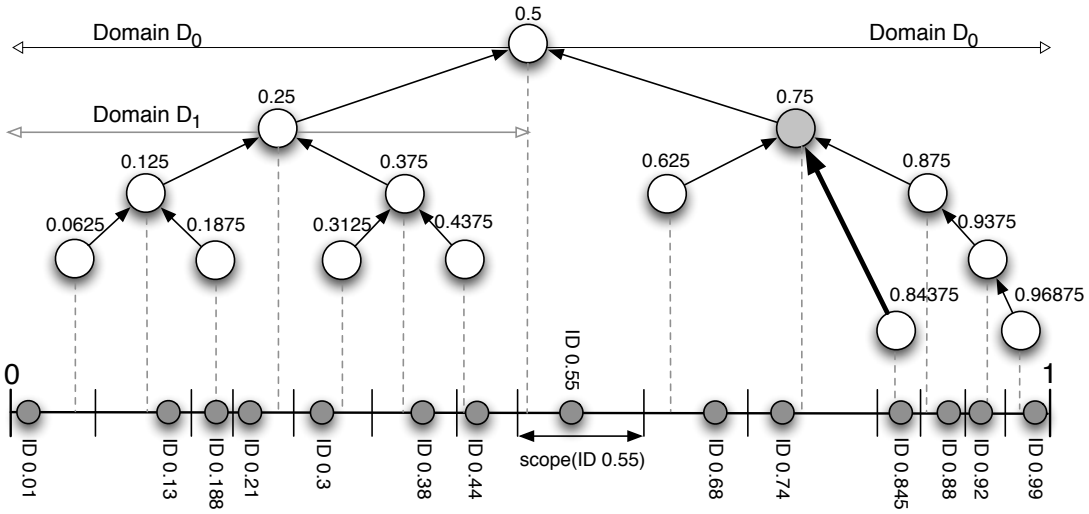
This Parent-Coordinator C_p is responsible for a Domain, which is one step larger and has a Domain Key that lies not in the scope of peer p. To this Parent-Coordinator, peer p sends periodical update messages containing monitoring information. In Figure 13b, we depict an example topology in SkyEye.KOM. We also give the actual information flow that considers the multiple roles a peer takes in the tree. Thus, peer 0.74 sends update messages to the peer responsible for the Domain Key 0.5 and acts as a Coordinator for the Domains corresponding to the Domain Keys 0.74 and 0.8125, thus as Parent-Coordinator of the corresponding child nodes.

JOINING AND LEAVING THE TREE TOPOLOGY

To join the SkyEye.KOM tree topology, peers calculate their positions in the tree using Formulae 3.16 and 3.17 as well as their Parent-Coordinator. In the next step, they send an update message to this Parent-Coordinator which is replied with an acknowledgment message. These update messages are used to inform the Parent-Coordinators about their *Sub-Coordinators*, i.e the child nodes in the tree. As the Parent-Coordinator and Sub-Coordinators are known to a peer through this simple step, the whole topology is connected. Please note, here, we rely on the assumption that the underlying DHT works properly. The maintenance of the tree topology is done with periodically sending update messages. No further steps are needed, even in the case of failing Parent-Coordinators or Sub-Coordinators.



(a) Example SkyEye.KOM Tree Topology



(b) Flow of Monitoring Information in the Tree Topology

Figure 13: Distance-based Routing in Structured P2P Overlays

The failing of a Parent-Coordinator C_*^l is detected by its Sub-Coordinators, which are Coordinators C_*^{l+1} of lower levels. They detect a missing acknowledgment message. As soon as a peer p identifies that the Parent-Coordinator C_p^l failed, no further actions need to be done. The peer uses the second postulated function `void route(key K, message M, nodehandle hint)` with the parameters $K = K_p^l$, M being the update message and optionally the previously retrieved ID of the new Coordinator as a `hint`. Alternatively, it starts a lookup for the peer now being responsible for the Domain Key (K_p^l). For this, it uses the third postulated function `Nodehandle getNodehandle(key K)`, with the parameter $K = K_p^l$. The identified peer is then the new Parent-Coordinator, which can be addressed by the forth postulated function `send(message M, nodehandle P)`. Eventually, the link to the Parent-Coordinator is updated. However, the information carried by the failed Coordinator is lost. Although the information is lost, the new Coordinator is updated when the Sub-Coordinators send their updates to their new Parent-Coordinator.

In the case of the failure of a Sub-Coordinator, nothing else needs to be done, as the new Sub-Coordinator determines its new Parent-Coordinator during the calculation of its position in the tree. No further maintenance is needed. Please note, we assume that peers fail and do not leave gracefully and that (strong) churn exist. Having only 2 message types at the over-overlay for establishing and

maintaining the tree topology, we avoid a very stateful protocol which might lead to failures, deadlocks and instability.

MODEL OF THE MONITORING TREE TOPOLOGY

The characteristics of the monitoring tree topology have a great influence on the performance of the monitoring solution and the occurring costs. Thus a detailed model of the tree topology helps in better understanding the performance and costs of our monitoring approach. If there are N participating peers in the network, we are interested in:

- Where does the $(N+1)$th peer join the tree?
- How is the distribution of nodes in the tree?
- How many nodes are placed on level i on average?
- How many children does a node on level i have on average?
- How many messages does a peer have to send for monitoring purposes?

While tree structures, especially b-trees, have been discussed in literature for a long time, they do not address the characteristics of the monitoring tree in SkyEye.KOM and the dynamism of churn in detail. The main motivation for creating an analytical model for SkyEye.KOM is that b-trees do not consider the join-behavior of peers in the monitoring tree. B-trees have typically optimal height and the node positions are predetermined. In the creation of the monitoring tree topology, the tree is created based on the ID distribution of the peers and thus the node degrees are varying.

Our first focus in the analytical model is to determine the number of peers on a level in the tree. The approach to solve this problem is to consider the probability of a node existing on level i. The variables in Table 1 have been used in the equations describing the topology of the monitoring tree.

$N \in \mathbb{N}$	Number of nodes
$\beta \in \mathbb{N}$	Branching factor
$i \in \mathbb{N}$	Tree level
$F_i(N) \in \mathbb{R}$	Expected free places on level i, if there are N nodes in the tree
$P_i(N) \in \mathbb{R}$	Probability that a node appears on level i, if there are N nodes in the tree
$E_i(N) \in \mathbb{R}$	Expected number of nodes on level i, if there are N nodes in the tree
$M_i(N) \in \mathbb{R}$	Maximum places on level i
$F(N) \in \mathbb{R}$	Entire free places in the tree, if there are N nodes in the tree

Table 1: Variables used in the Model to Describe the Monitoring Topology

The expected number of peers per level, $E_i(N)$, is calculated using following variables:

- $F_i(N)$ - the number of free places on level i
- $F(N)$ - the number of free places in a tree with N peers
- $P_i(N-1)$ - the probability of a new peer added to level i when joining a tree with $N-1$ peers
- $E_{i-1}(N)$ - the number of peers on level $i-1$

The term *place* describes an "open position" in the tree, i.e. an inner node with less than β child nodes, with β being the branching factor of the tree. The entire free places of a tree can be calculated as follows. When a peer joins the tree, it consumes one free place, but also brings β new free places to the tree. This aspect is visualized in Figure 14.

$$F(N) = F(N-1) + \beta - 1 \tag{3.20}$$

For the calculation of the peer population per level, we consider a scenario of a peer joining a tree with $N-1$ peers. Regarding to the free places on level i after the joining, three cases can be distinguished:

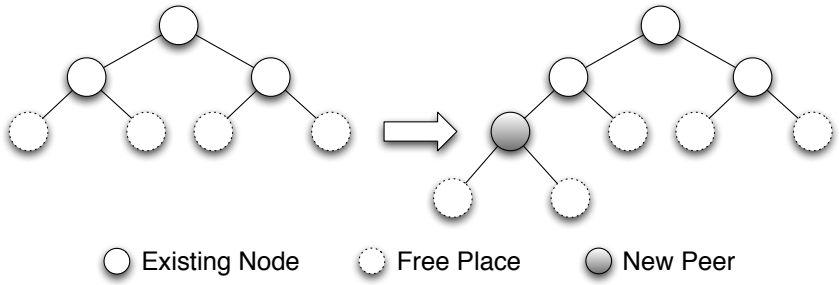


Figure 14: Effect on Free Places after the Join of a new Peer

Case 1: A new peer joins the tree at level $i-1$ providing β free places on level i. The probability for this case is termed $P_{i-1}(N-1)$. The expected number of peers at level i after the joining of the N^{th} peer resulting from this case is:

$$P_{i-1}(N-1) \cdot (\beta + F_i(N-1)) \tag{3.21}$$

Case 2: A new peer joins the tree on level i and consumes one free place. The probability for this case is termed $P_i(N-1)$. The expected number of peers at level i after the joining of the N^{th} peer resulting from this case is:

$$P_i(N-1) \cdot (F_i(N-1) - 1) \tag{3.22}$$

Case 3: A new node is neither added on level $i-1$ nor on level i. This has no influence on the free places on level i. The probability for this case is:

$$\frac{F(N-1) - F_i(N-1) - F_{i-1}(N-1)}{F(N-1)} \tag{3.23}$$

The expected number of peers at level i after the joining of the N^{th} peer resulting from this case is:

$$(1 - P_i(N-1) - P_{i-1}(N-1)) \cdot F_i(N-1) \tag{3.24}$$

The number of free places on level i is the sum of Eq. 3.21, Eq. 3.22 and Eq. 3.24.

$$\begin{aligned} F_i(N) = {} & P_{i-1}(N-1) \cdot (F_i(N-1) + \beta) \\ & + P_i(N-1) \cdot (F_i(N-1) - 1) \\ & + (1 - P_i(N-1) - P_{i-1}(N-1)) \cdot (F_i(N-1)) \end{aligned} \tag{3.25}$$

Next, we calculate $P_i(N-1)$, the probability of a new peer coming to level i when joining a tree with $N-1$ peers. In order to do so, the ID ranges corresponding to the capacities on various levels have to be calculated. Knowing the ratio of the total ID space corresponding to an level i, results in the desired probability function $P_i(N-1)$.

The Domain size corresponding to a Coordinator in the tree is directly related to its level, as depicted in Figure 15 with $\beta = 2$. Each node in the tree represents an ID range of the monitoring tree. Beginning from the root, the Domains, being ID ranges, are successively divided in β Sub-Domains. A Sub-Domains at level i has an ID range of $(1/\beta)^i$. Following equation counts:

$$P_i(N-1) = \frac{1}{\beta^i} \cdot F_i(N-1) \tag{3.26}$$

Knowing the available capacities and filling probabilities in the tree, next the expected number of peers per level $E_i(N)$ is modeled. $E_i(N)$ is a recursive function, as the probability of a node being appended on level i depends on where the $N-1$ previous nodes are situated. As the probability of the N^{th} node being appended on level i is already given by Eq. 3.26, $E_i(N)$ can be written as:

$$E_i(N) = E_i(N-1) + P_i(N-1) \tag{3.27}$$

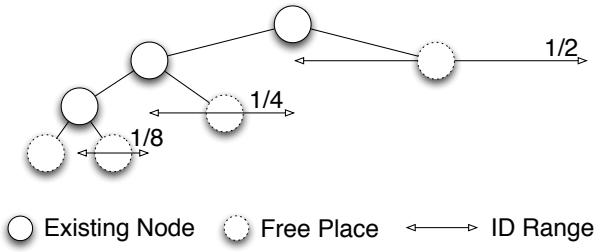


Figure 15: Probability that a Node is appended on Level i

After calculating the number of peers on level i, the amount of free places on this level must be derived. This is done by subtracting the actual number of nodes from the maximum possible places on level i. Next, we calculate the expected maximum number of places in a level, depending on the number of peers in the tree. Obviously the maximum number of places on level 0 is 1, that of the root node. In general, the maximum capacity $M_i(N)$ on level i depends on expected number of nodes on level $i-1$:

$$M_i(N) = \beta \cdot E_{i-1}(N) = \beta \cdot E_{i-1}(N-1) + P_i(N-1) \tag{3.28}$$

Thus, the remaining free places on level i can be calculated as:

$$F_i(N) = M_i(N) - E_i(N) = \beta \cdot E_{i-1}(N) - E_i(N) \tag{3.29}$$

For the completeness of the model, we also add the formula for the expected number of peers on level i which can now be derived:

$$\begin{aligned} E_i(N) &= E_i(N-1) + P_i(N-1) \quad = E_i(N-1) + \frac{1}{\beta^i} \cdot F_i(N-1) \\ &= E_i(N-1) + \frac{1}{\beta^i} \cdot \beta \cdot (E_{i-1}(N-1) - E_i(N-1)) \\ &= E_i(N-1) + \frac{\beta \cdot E_{i-1}(N-1) - E_i(N-1)}{\beta^i} \end{aligned} \tag{3.30}$$

Thus, we modeled the number of peers per level ($E_i(N)$), the free places in the tree ($F_i(N)$), the maximum theoretical number of places per level ($M_i(N)$) and the tree depth, which is related to $E_i(N)$. With this, the tree characteristics are described and a deeper understanding in the creation of the tree is gained.

3.4 MONITORING SYSTEM-SPECIFIC INFORMATION

After introducing the topology of the SkyEye.KOM tree, we describe the protocols and mechanisms used to gather, aggregate and disseminate system-specific monitoring information. The protocol considering peer-specific information is discussed in the next section.

The tree topology we created, interconnects all peers and allows for dependable ID-based sending of messages to the Parent-Coordinator. This function is used to coordinate the information flow from the leaves of the tree towards the root. A global view is created at the root and from there propagated level-wise down the tree to all peers. In this section, we discuss the nature and aggregation of system-specific monitoring information and how this information is gathered and disseminated.

3.4.1 *Metrics used for System-Statistics*

System-specific monitoring information gives insights into the quality characteristics of a running p2p system. They contain a snapshot view of the whole system in a statistical representation. System-specific characteristics are described in a set of metrics, which are monitored and measured permanently on each peer. Please note that we use the term *metric* both for the classification and type of a measured value (e.g. hop count) and also for the actual value (e.g. 5).

In order to assemble the local observations of all peers to a common global view, we use the tree topology of SkyEye.KOM. This subsection introduces the metrics, which are measured at each node, collected within the SkyEye.KOM tree and used later on to create the system statistics. Table 2 displays the metrics which are used in the prototype and the simulations. Please note that this set is extendible for a wide range of other metrics also from other functional layers in a p2p system. The listed elements in the table refer to the measurements of a single SkyEye.KOM node. The categories on the metrics help to see the range of applicability of the solution, and they do not impose a strict limitation. Further categories may be related to such as characteristics, contribution and reputation levels of peers, storage statistics of replication layers and many more.

A	S	P	Statistic - (A: Analytic Model, S: Simulation View, P: Prototype)	Unit
General Statistics - 5 values for all statistics: Count, Min, Max, Sum, Sum of Squares				
	X	X	Time	#
X	X	X	Number of peers	#
X	X	X	Online time of the peers	s
X	X		Information age at root, freshness	s
X	X	X	Peer level	#
Lookup Duration and Hops - 5 values each				
X	X		Delay in the overlay	s
X	X	X	Number of hops in the overlay	#
X	X		Number of forwarded messages	1/s
Overhead per Message Type - 20 values each: sent and received messages and traffic				
X	X	X	Complete overhead	1/s,KB/s
X	X	X	Complete overhead per level	1/s,KB/s
X	X	X	SkyEye.KOM overhead	1/s,KB/s
	X		Overlay overhead	1/s,KB/s
	X		Join and leave overhead	1/s,KB/s
	X		Lookup overhead	1/s,KB/s
General Overhead - 10 values each (global) and for each level: up- and download				
	X	X	Ratio of bandwidth used for unzipped data	%
		X	Ratio of bandwidth used for zipped data	%
Model-specific - 1 value each				
X			Number of Sub-Coordinators, free places, leafs, non-leafs	#
X			Probability to join a level	%
Simulation-specific - 15 values				
	X		Lookup operations: completed, succeeded, failed	1/s
Prototype-specific - 5 values each, 15 values for the memory-related metrics				
		X	CPU utilization	%
		X	Disk space	%
		X	Heap memory: maximum, initial, used	%
		X	Non heap memory: maximum, initial, used	%

Table 2: List of Monitored General Statistics

For the metrics mentioned in Table 2, we give a brief summary of their potential usage. Statistics on the number and online time of the peers in a p2p system in combination the traffic patterns of join-and leave-messages reveal the churn behavior and distribution of the peers.

The metrics related to the behavior of SkyEye.KOM are covered in the topology-related and traffic-related category. This helps the p2p system provider to both monitor its functional layers (e.g. p2p overlay) as well as the monitoring layer itself. Knowing the characteristics of the tree and the age of the monitored information is a relevant criterion for monitoring solutions. Another important aspect of

A	S	P	Statistic - (A: Analytic Model, S: Simulation View, P: Prototype)	Unit
Reference Signal - 20 values each: T=1m,3m,10m,30m				
	X		Monitored ZigZag Signal	#
	X		Monitored Sine Signal	#
Overhead per Message Type - 20 values each: sent and received messages and traffic				
X	X	X	Metric Update overhead	1/s,KB/s
X	X	X	Metric Update ACK overhead	1/s,KB/s

Table 3: List of Statistics on System Monitoring

monitoring are the costs, these are observed as well. The complete traffic of a p2p system is divided into in and out traffic generated by the overlay and in and out traffic generated by SkyEye.KOM. Additionally, there exist metrics concerning different types of overlay or SkyEye.KOM messages.

Some of the metrics, especially the metrics related to the resources at the peers, are interesting to see the potential of the p2p system. This potential is harnessed by the function of capacity-based peer search, which we introduce in the upcoming chapter.

These metrics are not gathered as just single snapshots of the current status of the p2p system. They are gathered in a *statistical representation* considering the *average* over all peers, *minimum* and *maximum* values, the *sum* of the values and the *standard deviation* which help the user to completely analyze the behavior of the p2p system. As these statistics are calculated over a set of measurement results of individual peers, SkyEye.KOM has to compile the corresponding individual statistics to a global statistic. Here, aggregation is used as an operator to create statistics over a larger Domain out of statistics over many smaller Domains. Please note that not all aspects of the statistics for all metrics result in reasonable information. For example, the number of peers is reported individually by each peer as one. The sum of this values result in the total number of peers in the network. The statistical information regarding the minimum, maximum and standard deviation on these individual "one" values are useless. Nevertheless, the same aggregation strategy is performed on all metrics regardless the content and semantics of the metric. As a result, signaling overhead and complexity for marking special metrics are omitted. The process of aggregation is discussed in the next subsection.

3.4.2 *Aggregation of the Metrics*

This subsection describes the statistical presentation of the monitoring information and the diverse aggregation functions, which are used to aggregate metrics in SkyEye.KOM. First, we introduce the definition of aggregation functions and then the functions considered to operate on the set of measured data are discussed and presented.

An aggregation function in the context of computer science is a function that returns a single value from a set of input values. Let $Values$ be a continuous set of values (e.g the set of `double`), then an aggregation function f is a specific function (e.g. sum) that matches a non-empty set of values to a single value:

$$f : \wp(Values)/\{\} \rightarrow Values \tag{3.31}$$

A typical aggregation function is the sum, minimum and cardinality of a set. In order to have usable aggregation functions for the monitoring purpose, the aggregation functions must also fulfill the *hierarchical computation property* as defined in [YD04a]. Let v_* be a set of values of the same metric with $|v_*| = n$, then the following counts for any index i, j with $1 < i < j < n$:

$$f(v_1, \ldots, v_n) = f(\ f(v_1, v_2, v_3, \ldots, v_i), f(v_{i+1}, v_{i+2}, \ldots, v_j), \ldots, f(v_{n-1}, \ldots, v_n)\ ) \tag{3.32}$$

Thus, the hierarchical computation property requires that the aggregation function is commutative and associative. With this property, the order of the parameters in the aggregation function as well as the aggregation of aggregated results do not have influence on the final aggregation result.

$$\begin{aligned} &f(v_1, v_2) = f(v_2, v_1) \\ &f(v_1, v_2, v_3) = f(f(v_1, v_2), v_3) = f(v_1, f(v_2, v_3)) \end{aligned} \tag{3.33}$$

Function	Description
count	Counts the number of metric values
min	Calculates the minimal value of a metric
max	Calculates the maximal value of a metric
sum	Sums the values of a metric
sum_squares	Sums the squares of the values of a metric
mean	Calculates the average of a metric
variance	Calculates the variance of a metric
standard_deviation	Calculates the sample standard deviation of a metric

Table 4: Description of the Aggregation Functions in SkyEye.KOM

Table 4 depicts the list of statistical information which are retrieved from the measurements of the individual peers. The first four aggregation functions (*min*, *max*, *sum*, *count*) compute directly a single numeric value based on several numeric input values. The mean, variance and standard deviation cannot be calculated directly, they are combined from other aggregation functions, one of them being the sum of squares. An overview of the used aggregation functions is given in Table 5.

These aggregation functions fulfill the hierarchical computation property, i.e. they are commutative and associative. This is easy to show for *min*, *max*, *sum* and *count*. The average or mean value we calculate by

$$\mathrm{mean}(a_1, \ldots, a_n) = \mathrm{sum}(a_1, \ldots, a_n) / \mathrm{count}(a_1, \ldots, a_n) \tag{3.34}$$

For the standard deviation, however, in order to comply with the hierarchical computation property, we do not choose the well-known function

$$\mathrm{standard\_deviation}(a_1, \ldots, a_n) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (a_i - \overline{a})^2} \tag{3.35}$$

with $\overline{a} = \mathrm{mean}(a_1, \ldots, a_n)$ as the arithmetic mean of all values and $n = \mathrm{count}(a_1, \ldots, a_n)$ as the quality of values. Instead, as we do not know $\overline{a}$ in advance, we use another formula which eliminates the term $\sum_{i=1}^{n}(a_i - \overline{a})^2$. It uses the *sum* as well as the *sum of squares* aggregation functions, which both

Aggregation Functions
$\mathrm{count}(a_1, \ldots, a_n) = n$
$\min(a_1, \ldots, a_n) = \min(a_1, \min(a_2, \ldots \min(a_{n-1}, a_n) \ldots))$
$\max(a_1, \ldots, a_n) = \max(a_1, \max(a_2, \ldots \max(a_{n-1}, a_n) \ldots))$
$\mathrm{sum}(a_1, \ldots, a_n) = \sum_{i=1}^{n} a_i$
$\mathrm{sum\_squares}(a_1, \ldots, a_n) = \sum_{i=1}^{n} a_i^2$
$\mathrm{mean}(a_1, \ldots, a_n) = \frac{\mathrm{sum}(a_1, \ldots, a_n)}{\mathrm{count}(a_1, \ldots, a_n)}$
$\mathrm{variance}(a_1 \ldots a_n) = \frac{\mathrm{sum\_squares}(a_1, \ldots, a_n)}{\mathrm{count}(a_1, \ldots, a_n)} - \left(\frac{\mathrm{sum}(a_1, \ldots, a_n)}{\mathrm{count}(a_1, \ldots, a_n)}\right)^2$
$\mathrm{standard\_deviation}(a_1, \ldots, a_n) = \sqrt{\mathrm{variance}(a_1, \ldots, a_n)}$

Table 5: Formulae of the Aggregation Functions

fulfill the hierarchical computation property. As a result for the sample standard deviation, we get the following formula

$$\begin{aligned} \text{standard\_deviation}(a_1,\ldots,a_n) &= \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} a_i^2\right) - \left(\frac{1}{n}\sum_{i=1}^{n} a_i\right)^2} \\ &= \sqrt{\frac{\text{sum\_squares}(a_1,\ldots,a_n)}{\text{count}(a_1,\ldots,a_n)} - \left(\frac{\text{sum}(a_1,\ldots,a_n)}{\text{count}(a_1,\ldots,a_n)}\right)^2} \end{aligned} \tag{3.36}$$

Each of the statistical aspects discussed in Table 5 has a corresponding aggregation function f_{type}. The type depicts the specific statistical aspect (e.g. the average). All of the values in the monitoring tree are numerical (`double`), thus we do not introduce variable data types.

In order to collect the information of all peers in one statistical representation, information from lower levels in the tree have to be aggregated and sent upwards in the tree. The design of the corresponding protocol for gathering a global view on system-specific information is described in the following.

3.4.3 *Protocol Design Decisions*

After having observed the set of metrics and the aggregation functions operating on them, now we present the information flow for gathering the aggregated metrics into one spot and disseminating the global view to all peers. As a basis for the information flow, the SkyEye.KOM tree topology is used. However, there are various design decisions on the usage of this topology. We first discuss the design decisions made and describe then the protocol for gathering and disseminating system statistics.

Monitoring Scope
For the monitoring of the system status, one could either monitor the whole p2p system or focus on a (small) representative subset of the peers and interpret their status. Although some solutions exist focusing on the *sampling* of a p2p system, a global view on the system status should contain the status of all peers. With this, load imbalances can be identified and valid minimum and maximum values can be deducted from the p2p system.

Reactive vs. Proactive
Gathering information on the p2p system can be done proactively and reactively. A proactive solution gathers the information continuously over time and responds quickly to queries, while needing constant effort for the preparation of the information. Reactive solutions, on the other hand, gather the required information each time a query is started, which is beneficial in scenarios with rare queries by only a few peers. Obtaining a result to a reactive query takes significantly more time than a proactive query, as the required information first has to be generated through a process which involves all peers in the network. SkyEye.KOM uses a proactive approach as the monitoring information on the system state is interesting for all peers all the time. Thus, the overhead for preparing the information proactively for all peer is less than the overhead for alternative reactive query resolutions.

Push vs. Pull
Building an infrastructure for monitoring a large-scale p2p overlay requires the exchange of messages in a coordinated manner. Information can be pulled and pushed. While pulling allows for better control on when to gather the information, it also requires one to maintain the information on the set of peers from which the information is pulled. In a push-based approach, contact with only one peer (to whom the information is pushed) has to be maintained. This approach automatically integrates new peers in the information architecture and releases failing peers. In order to reduce complexity and to save overhead costs, SkyEye.KOM uses a push-based approach.

Heterogeneous vs. Equal Roles
An information-gathering protocol can have homogeneous and heterogeneous roles and tasks for the peers involved. Homogeneous roles facilitate the protocol for gathering information, as no further

complexity is introduced through side-protocols for various roles. Heterogeneous roles, on the other hand, are useful in order to cope with heterogeneous load levels in the information-gathering architecture. However, the load on all peers for gathering system statistics is very similar due to the aggregation, thus heterogeneous roles are not needed. Using a push-based approach in a tree topology, each peer receives monitoring information from its child nodes, aggregates them and pushes the aggregated monitoring information to its father node. The incoming traffic load on all peers is limited by the number of child nodes, which is limited by the branching factor of the tree. The outgoing traffic for all peers, except the root, is the same, as the monitoring information is aggregated and sent only to one node (the corresponding Parent-Coordinator). This results in very similar load for all peers, which does not state the requirement for heterogeneous roles.

Combined Approaches for Information Gathering and Dissemination
For the gathering and provision of the global view on the system statistics, either a set of protocols or a combined approach may be used. While a differentiated set of protocols allows for task-specific optimization, it also introduces complexity and is only needed if the distinguished tasks are heterogeneous. In our case, for the gathering and dissemination of a system-specific global view, both task are very similar as they both require to submit a aggregated view of the system. In addition, both the gathering and dissemination messages in the protocol are exchanged by the same pair of peers, but in an opposite direction. We use this effect to save overhead, by applying the messages for dissemination of the monitoring as acknowledgment messages for the information gathering messages. Thus, we follow in the protocol a combined approach for information gathering and dissemination.

3.4.4 *Protocol for Monitoring the System-specific Information*

In this subsection we describe the protocol of SkyEye.KOM for gathering and disseminating the systems statistics in structured p2p systems.

GATHERING SYSTEM STATISTICS FOR A GLOBAL VIEW

In order to gather the system statistic on all peers in the network coordinately, all individual peer observations have to aggregated. As a first step, each peer p uses its sensor component to derive its individual local measurement regarding all metrics. Here, we use the assumed dependency $(\mathrm{Monitor}^{system}_{requirement}\ 1)$ for the provision of these metrics. Each metric has its own $name$, e.g. "relative upload bandwidth consumption", and the statistical aspects of it correspond to a specific $type$, e.g. average. The value $V^{p}_{i,type,name}$ describes the individual measurement value of the peer p, located at level i in the tree, regarding the metric $name$ and the statistic aspect $type$.

All peers are Coordinators for a Domain with possibly further peers in it. If no Sub-Coordinators exist in that Domain (i.e. the corresponding peer is the only peer in the Domain), it periodically sends its update information to its Parent-Coordinator. This update information consists only of the single measurement of the single peer in the Domain. With this, it implements the function $(\mathrm{Monitor}^{system}_{function}\ 1)$ for injecting the local measurements of single peers. The update intervals, $t_{metricPeriod}$ or UI, of the peers do not have to be synchronized. However, a similar value is recommended. The leaf nodes in the monitoring tree topology have no further tasks in the process of gathering monitoring information.

If a Coordinator has Sub-Coordinators in its Domain, it receives periodical update messages from them, termed *metric updates*. These metric information from the Sub-Domains are aggregated at the Coordinator and also aggregated with the local measurements of the Coordinator, implementing function $(\mathrm{Monitor}^{system}_{function}\ 1)$. Thus, the Coordinator of a Domain aggregates all information of lower Domains and creates a metric update, that describes the system status corresponding to its own Domain.

The Coordinator at level i receives metric updates from its β Sub-Coordinators at level $i-1$, it uses the aggregate functions f_{type} to compute the aggregate value of the β received values, including its own measured value, as follows:

$$V'_{i,type,name} = f_{type}(V^{1}_{i-1,type,name}, \ldots, V^{\beta}_{i-1,type,name}, V^{own}_{i,type,name}) \tag{3.37}$$

Now, $V'_{i,type,name}$ contains the aggregate value of the Domain at level i, for which the calculating Coordinator is responsible.

The aggregation of all metrics according to all aggregation functions (types) results in a statistical representation of the system status in the corresponding Domain. It is sent as metric update in the next update interval to the Coordinator one level higher, and there, the same procedure is applied. The flow of the information is depicted in Figure 16. To conclude, during the information flow towards the root, every Coordinator in the tree aggregates the received data with its own measurements and propagates it to the Parent-Coordinator. This is done until the root is reached. At the root of the tree, the monitoring information regarding the system status is complete and can be used. On lower levels, statistics on subsets of the tree are available.

DISSEMINATION OF THE GLOBAL VIEW TO ALL PEERS

For the dissemination of information from the top to the bottom of the tree, a pushed-based and proactive approach is used in SkyEye.KOM. The dissemination of the information implements the function **(**$\text{Monitor}^{\text{system}}_{\text{function}}$ **2)** as stated in the section on functional requirements for the monitoring mechanism. The goal of a proactive dissemination is to keep all peers in the network always informed about the status of the p2p system regardless of their individual actual interest. Although this may seem inefficient, it saves signaling overhead for the indication of peers who are interested in the global view.

To reach all the peers, the established monitoring tree topology is used. Whenever a Coordinator or the root receives and processes a metric update from lower levels of the tree, it acknowledges this update. The acknowledgment message has a twofold purpose: informing the Sub-Coordinator about the validity of its Parent-Coordinator and providing the Sub-Coordinator with a fresh view on the system status. The acknowledgment message contains the global system statistics received from one level higher in the tree. Thus, starting from the root, all lower Coordinators in the tree receive the acknowledgment message with the global system statistics and propagate it further down. Eventually, every peer receives the global statistics on the p2p system, with the degree of freshness depending on their position in the tree. By this, even new peers in the tree are directly informed about the p2p system's status within their first update interval

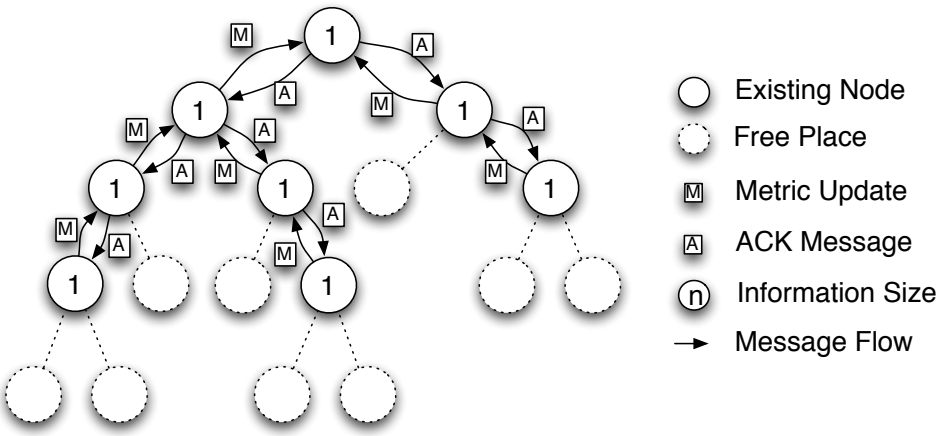


Figure 16: Gathering and Disseminating System Statistics

To understand the whole process of creating the system statistics, we depict the protocol of the periodically repeated metric updates in Figure 17:

1. **Receiving statistics of lower Domains, sending ACKs with global view**: Periodically, Parent-Coordinators receive metric update messages containing the system statistic regarding the Domains they manage. The Sub-Coordinator-specific metric sets are stored with the contact information of the Sub-Coordinators and a timestamp in a table. Every metric update received from a Sub-Coordinator is answered with an acknowledgment message, containing the global system statistics received from one level above (see step 5).

   When an update message is going to be created, every peer regularly inspects the timestamp of every Sub-Coordinator entry. If the period between the update and the actual time exceeds a

given threshold called $t_{oldMetrics}$, the entry is removed. This updating of the entries prevents the aggregation of old metrics from Sub-Coordinators, which may already have a new Coordinator or just went offline.

The value for the threshold $t_{oldMetrics}$ is related to the metric update interval $t_{metricPeriod}$ (alias UI) and expressed as a multiple of the metric update interval. Several simulations with different multiplication factors have shown that a good value for the multiplication factor of $t_{metricPeriod}$ ranges between 1.5 and 2.

2. **Measuring of own statistics**: Every node measures its own metrics as depicted in Table 2

3. **Aggregation of statistics for own Domain**: The own metrics of a peer (calculated in step 2) are aggregated with the received metrics of valid Sub-Coordinators (calculated in step 3). Due to the update of the table in step 1, a node only aggregates metrics of updates received in the last update period. The aggregation results in a metric update message, ready to be sent to the Parent-Coordinator in the tree.

4. **Push of statistics to Parent-Coordinator**: As next step (i.e. step 4), every peer calculates its position in the SkyEye.KOM tree as well as its Parent-Coordinator. After this position-fixing in the tree, the metric update message is sent to the Parent-Coordinator of the corresponding peer.

5. **Acknowledging received statistics with global statistics**: The parent node in the tree (the Parent-Coordinator of the peer) acknowledges the metric update with its view on the global system statistics. This global view on the p2p system is the result of the monitoring protocol. It is stored locally and used as input for the acknowledgments sent to Sub-Coordinators (in step 5).

DISCUSSION OF THE FEATURES

Having discussed the metric set, the aggregation functions and the metric update protocol, we briefly discuss the effect of churn and failures on the presented approach.

We summarize the effects of churn on the SkyEye.KOM topology, which have already been discussed in Subsection 3.3. In the case of a failure of a Coordinator, a new Coordinator is picked based on the ID of the orphaned Domain Key. Within the next update period, the new Coordinator receives a new set of metric updates from its Sub-Coordinators and is able to create the complete view on the Domain within this single update period. Please note that this is done automatically as the second postulated function, `void route(key K, message M, nodehandle hint)`, always routes the update message M to the correct Parent-Coordinator based on the Domain Key K.

In the case of a failing Sub-Coordinator the corresponding Parent-Coordinator does not receive updates from this Sub-Coordinator anymore. Its entry in the table of Sub-Coordinators is deleted after a timeout of $t_{oldMetrics}$. This threshold is related to the metric update period $t_{metricPeriod}$, the multiplication factor is between 1.5 and 2. A small factor results in quick deletions of monitoring information of left Sub-Coordinator.

A Coordinator may receive two metric update messages from the same Sub-Coordinator during one update period, due to unsynchronized update intervals. In that case, it always overwrites the previously stored with the fresher monitoring information.

The periodic sending of metric update messages maintains the tree topology. Every peer periodically calculates its position in the tree, as well as the position of its Parent-Coordinator, before it sends its metric updates. Subsequently, the structure of the tree is always kept up-to-date and references to dead or old Parent-Coordinators are updated. This step also helps to identify new Parent-Coordinators. New peers on the other hand can quickly identify their Parent-Coordinator and receive with the first acknowledgment message the global view on the system statistics.

Due to the joining and leaving of peers, the responsibility ranges of the peers in the ID space may shift as well. In the case that a peer is relocated on the tree between two updates, the previously received information typically does not fit the new position in the tree. This case leads to a biased monitoring view as the monitoring information of a relocated peer may be present in the old Parent-Peer (for one update interval) and in the new Parent-Coordinator as well. Additionally, the statistics on the Domain view of the relocated Coordinator have to be considered as well. These statistics were valid for the

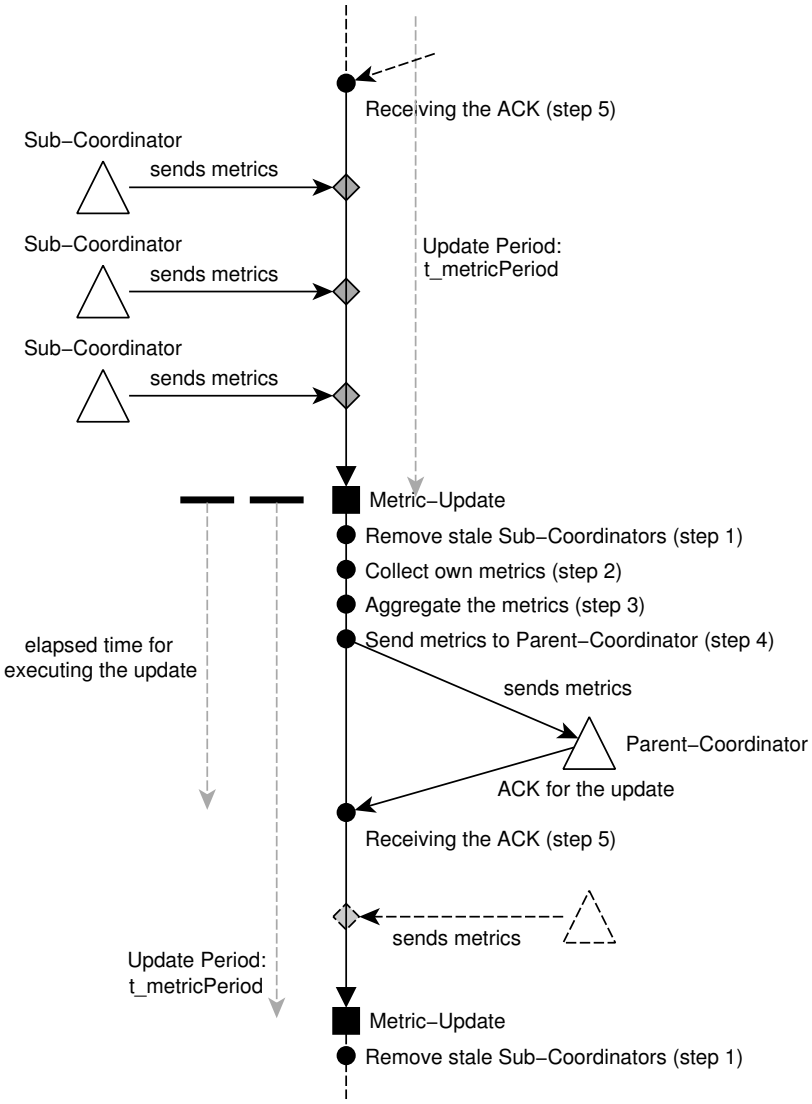


Figure 17: Protocol for Gathering System-specific Information in the SkyEye.KOM Tree

previous Domain the Coordinator was responsible for. These statistics could be reset, kept or adapted to the new level of the Coordinator. However, the monitoring results are biased by the relocation of the peers to new positions in the tree due to the joining and leaving of the peers. We address this issue in the following section by presenting smoothing approaches for the biased monitoring view.

In summation, we create a monitoring tree topology that adapts its structure in a fast and efficient way to the underlying overlay. It copes easily with churn and implements an efficient yet very fault-tolerant topology maintenance approach. We only use messages for gathering and disseminating monitoring information for the maintenance of the tree topology. In general, the update period $t_{metricPeriod}$ (alias UI) has great impact both on the costs as well as the performance of the monitoring solution, as it is responsible for triggering the metric updates and thus the tree-maintenance.

SMOOTHING OF THE MONITORED METRICS

In order to eliminate monitoring outliers and obviously biased results indicated by occasional jumps in the metric values, we apply various monitoring smoothing mechanisms. A smoothing function defines an upper bound on the changes of the monitored metrics (i.e. it keeps the derivation function in a decent interval). Smoothing functions require a history of the observations, in order to analyze the trend and slope.

We applied two smoothing functions: the median over an observation history and exponential smoothing. Let H be the history of observations with m_H the current measure, then we calculate following smoothed views. The median $M_i(H)$ is a parametrized function with an odd value i denoting the number of considered previous measurements, thus $i = |H|$. The median over a sample set gives us the value in the sample set that is in the middle of the order of values. Outliers are eliminated with the median successfully.

In exponential smoothing, a history of observations is considered as well, but with decreasing weights for older measurements. A smoothed value s_H for $|H|$ observations is recursively calculated using a current measurement m_H and the smoothed value of previous observations with α as smoothing factor:

$$s_H = \alpha m_H + (1 - \alpha) s_{H-1} \tag{3.38}$$

Here, outliers are considered, but jumps in the observation are smoothed to trends. By avoiding outliers, the refinement mechanisms also introduce an additional delay in the information propagation, thus the information is less fresh. The trade-off between freshness and having fewer outliers is parametrizable with the history size of previous observations.

SYNCHRONIZED INFORMATION RETRIEVAL AND DISSEMINATION

The core topology of SkyEye.KOM does not state requirements on the update times of individual nodes but assumes that they are similar or equal. With this, the propagation of the information in the tree towards the root is "stored" in average for the half of an update interval on each level before it is forwarded one level higher. This leads to an information age of $O(UI \cdot \log_\beta(N))$ at the root and additional $O(UI \cdot \log_\beta(N))$ for the dissemination of the aggregated global view. In order to synchronize the gathering and dissemination of the monitoring information, we present an extension to the core monitoring protocol for system-specific information.

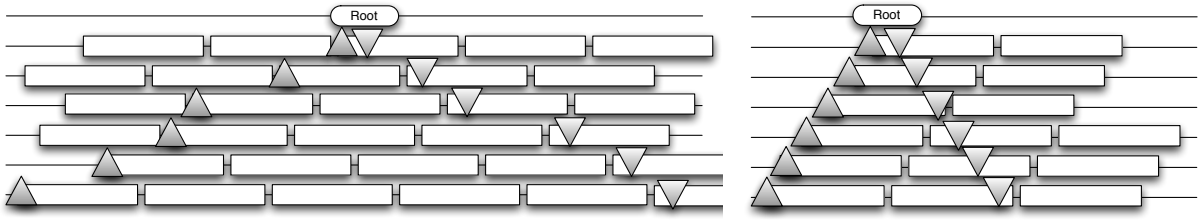
For the dissemination of the information, we propose an additional fast flooding in the tree with the current global view initiated by the root. Every time the root creates a global view based on the received monitoring information, it sends an ACK message marked for flooding to its Sub-Coordinators. Every Coordinator receiving such an ACK message marked for flooding extracts the global view for local storage and instantly forwards the message to its Sub-Coordinators. With this approach, we reduce the time for information dissemination from $O(UI \cdot \log_\beta(N))$ to $O(UI + \log_\beta(N))$ assuming a maximum delay of 1 second for forwarding the ACK message. However, this special ACK message increases the traffic overhead of SkyEye.KOM. Each peer receives one additional ACK message and has to forward it to as many as it has Sub-Coordinators. With regards to the small size of the ACKs and global view, these costs are tolerable.

For the acceleration of the gathering of the information, we propose the synchronization of the launches of the update messages, as depicted in Figure 18. This approach makes sure that updates are received just in time, for example, 1-2 seconds before the proceeding update is launched one level higher in the tree. With this, the time for gathering and aggregating the global view in the root of the tree is reduced from $O(UI \cdot \log_\beta(N))$ to $O(UI + \log_\beta(N))$, assuming an offset of 1 second. In order to communicate the offsets between Coordinators and Parent-Coordinators, the ACK messages for flooding are used that are propagated from the root. As they are instantly spread in the tree, they may be used as a relaxed synchronization clock. The ACKs for flooding contain initially the update interval UI in a field termed *launch offset*.The launch offset, LO, describes the time when the next update message shall be launched. With every step in the flooding this value is decreased by 2 seconds and propagated further in the flood. Each peer receiving the ACK marked for flooding synchronizes its update launch to this value and adapts the value in the ACK before transmitting it to its Sub-Coordinators.

To give an example, we assume that the transfer of an update message or ACK from one Coordinator to its Sub-Coordinator needs 0.5 seconds. Further, we assume the update interval $UI = 60s$ and set the launch offset $LO = UI = 60s$ in the root. The root sends an ACK for flooding upon creating a new global view to its Sub-Coordinators with the launch offset of $LO = 58s$. The Coordinators at level 1 set their next update to start in 58 seconds, adapt the field $LO = 56s$ in the ACK and forward the ACK to the Coordinators at level 2. These peers synchronize their update events to start with an offset of $LO = 56s$ and forward the ACK with $LO = 54s$. In short, every level l in the tree is advised to launch

the update event at the time $LO = 60 - 2l$ after receiving the flooded ACK. Further, we assumed a delay of 0.5 seconds per transmission resulting in a skewed and slightly biased synchronization. However, the procedure results in a setup that a peer at level l initiates an update message with an offset of $60 - 2l + 0.5l = 60 - 1.5l$ seconds after initiating the flood from the root. The resulting updates will be received one level higher 0.5 seconds later, thus at $60 - l$ seconds, leaving 1 second for aggregation and sending the update to the Parent-Coordinator.

With the proposed extension, all update actions are synchronized in a fashion that leaves only 1 second of delay on each tree level and results thus in an information freshness at the root that is $O(UI + \log_\beta(N))$. In combination with the synchronized dissemination of the monitoring information, SkyEye.KOM aggregates and disseminates the global view nearly optimal as no unnecessary message is sent, every peer is contacted only once for gathering and disseminating in the tree and the delay of unused monitoring information is reduced from $UI/2$ in average to 1 second.



(a) Unsynchronized Update Intervals (b) Synchronized Update Intervals

Figure 18: Unsynchronized and Synchronized Update Gathering and Dissemination

3.4.5 *Model of the System Monitoring Protocol*

Having discussed the protocol for gathering and disseminating a global view on the system statistics, next, we model the unbiased approach to give more insight into the interdependencies in the protocol.

First, we list the relevant variables used in this section in Table 6. With these variables in mind, we present the model for the performance of the monitoring approach and the costs resulting from it.

PERFORMANCE OF MONITORING SYSTEM-SPECIFIC INFORMATION

Precision is the main performance metric of the monitoring solution, but it requires both a monitored and a reference signal. A reference signal is always measurable in a controlled environment, such as the simulator. However, for the analytical evaluation we refer to the freshness of the information received by the root as an indicator for the precision. In the following, we model the freshness of the monitoring information at the root regarding one single peer and all peers in average, both with and without churn.

Freshness of the information, and thus the precision, depends on the update interval and height of the monitoring tree. The term for the age of the message at the root, sent by a peer p from level i can be given as:

$$Age(i, UI) = i \cdot UI \tag{3.39}$$

Formula 3.39 implies that the message containing the status of a peer p is propagated one step further towards the root in every update interval in the worst case. In the best case, the update of a Sub-Coordinator arrives just on time at the Coordinator before an update message to the Parent-Coordinator one level higher is sent. Thus, in the best case, the monitoring information is rapidly gathered and passed directly on through. The root has the global view on all peers, but this view is based on information of different freshness. The average age of messages at the root can be calculated with the help of the expected number of nodes on a level i, which is given by the equation 3.40.

The average age of messages at the root is in the worst case:

$$AvAge(E_i(N), i, UI) = \frac{\sum_{i=1}^{N} E_i(N) \cdot Age(i, UI)}{N} \tag{3.40}$$

$N \in \mathbb{N}$	Number of nodes in the monitoring tree
$i \in \mathbb{N}$	Tree level
$UI \in \mathbb{R}$	Update interval in seconds
$E_i(N) \in \mathbb{R}$	Expected number of nodes on level i when there are N nodes in the monitoring tree
$Age(i, UI) \in \mathbb{R}$	Age of information that the root receives, sent by a peer from level i
$AvAge(E_i(N), i, UI) \in \mathbb{R}$	Average age of messages that the root receives, which comprises information of different freshness
$\gamma \in \mathbb{R}$	Churn rate per update interval
$E_{churn,i}(E_i(N), \gamma) \in \mathbb{R}$	Expected number of failed peers on level i per update interval in a tree with N peers
$AvAge_{churn}(E_i(N), i, UI, \gamma) \in \mathbb{R}$	Average age of messages at the root when churn of peers is considered
$size_m \in \mathbb{R}$	Size of a metric update or an ACK
$NrMetricUpdate(N) \in \mathbb{N}$	Number of metric updates in an update interval
$NrMetricACK(N) \in \mathbb{N}$	Number of ACKs on metric updates in an update interval
$NrMetricTotal(N) \in \mathbb{N}$	Number of sent metric updates and ACKs in an update interval
$TrafficMetricUpdate(N, size_m) \in \mathbb{R}$	Traffic caused by metric updates in an update interval
$TrafficMetricACK(N, size_m) \in \mathbb{R}$	Traffic caused by ACKs on metric updates in an update interval
$TrafficMetricTotal(N, size_m) \in \mathbb{R}$	Traffic caused by sent metric updates and ACKs in an update interval
$AvMessage(N) \in \mathbb{R}$	Average number of messages (metric updates and ACKs)
$AvTraffic(N, size_m) \in \mathbb{R}$	Average traffic
$AvMessage_i(N) \in \mathbb{R}$	Average number of messages sent by a peer on level i (metric updates and ACKs)
$AvTraffic_i(N, size_m) \in \mathbb{R}$	Average traffic of a peer on level i

Table 6: Variables used to Describe the Monitoring of System Statistics

The model up to this point considers the joining and graceful leaving of peer. In both cases, the local tree structure of an affected peer (i.e. its Parent-Coordinator and the Sub-Coordinators) is updated and the model is valid. Next, we discuss the case of failing peers and their effects on the age of information in the tree.

A failed Coordinator is replaced at the time its successor detects its new position in the tree. This happens when the new peer in that position wants to send an update message. However, at this time, it does not have any information about its new Sub-Coordinators, so it sends its current (and therefore old) status. Here, we assume a worst case scenario, without smoothing of the monitoring information. The Sub-Coordinators of the relocated Parent-Coordinator detect their new Parent-Coordinator ID before sending their update messages. However, the maximum time difference for the new Parent-Coordinator to be informed is one update interval UI. Thus the messages of the Sub-Coordinators of a failing peer are delayed by one update interval UI, the messages from them have an average age of $i \cdot UI + UI$ instead of $i \cdot UI$.

Assuming that the churn rate in the system per update interval is γ, following equation is defined regarding the age of information in a system under churn. Firstly, on each level, we have a ratio γ of failed peers for which we take $i \cdot UI + UI$ as the age of the messages. Secondly, when building the average value, we divide the sum of ages by $N \cdot (1 - \gamma)$, with N denoting the number of peers when no churn is considered. So the number of the actual online peers which send information is $N \cdot (1 - \gamma)$.

As we know how many peers are on a level i without churn, we calculate the number of expected peers per level i with churn ($E_{churn,i}(E_i(N), \gamma)$) and the average age of information at the root ($AvAge_{churn}(E_i(N), i, UI, \gamma)$) as follows:

$$E_{churn,i}(E_i(N), \gamma) = E_i(N) \cdot \gamma \tag{3.41}$$

$$AvAge_{churn}(E_i(N), i, UI, \gamma) = \frac{\sum_{i=1}^{N}((E_i(N) - E_{Churn,i}(E_i(N), \gamma)) \cdot i \cdot UI + E_{Churn,i}(E_i(N), \gamma) \cdot (i \cdot (UI + UI)))}{N \cdot (1 - \gamma)} \tag{3.42}$$

The derived formulae give us insight on the influence the parameters have on the freshness of the monitoring information.

COSTS FOR MONITORING SYSTEM-SPECIFIC INFORMATION

In order to set freshness in relation to generated costs, we also modeled the cost-related metrics, like the total network-traffic in terms of the number of exchanged messages and bytes. First, we determine the number of messages generated. On the basis of this value, we calculate the traffic of the monitoring network.

We consider the number of messages in an update period. In SkyEye.KOM, all peers generate one metric update per update interval. The Parent-Coordinator gathers the information of all its Sub-Coordinators and together with its own information; it generates one single metric update. The number of generated messages equals the number of the peers minus the root. Every metric update message is answered with an ACK containing the global system view, which was aggregated at higher levels. The number of sent ACKs also equals the number of nodes minus one.

$$\begin{aligned} NrMetricUpdate(N) &= (N-1) \\ NrMetricACK(N) &= (N-1) \\ NrMetricTotal(N) &= (N-1) \cdot 2 \end{aligned} \tag{3.43}$$

Traffic caused by the monitoring network through sending metric updates and ACKs can be simply attained by multiplying the number of messages with the size of a message ($size_m$). Due to the aggregation of statistical data, the messages are small and independent of the corresponding monitored Domain size. Additionally, the metric update messages and the corresponding ACKs have the same size, as they both contain one instance of system statistics. Consequently, all messages for gathering and disseminating system statistics have the same size ($size_m$), independent of the sending peer and the type of the message. We give following equations:

$$\begin{aligned} TrafficMetricUpdate(N, size_m) &= (N-1) \cdot size_m \\ TrafficMetricACK(N, size_m) &= (N-1) \cdot size_m \\ TrafficMetricTotal(N, size_m) &= (N-1) \cdot 2 \cdot size_m \end{aligned} \tag{3.44}$$

The average number of messages and the average traffic occurring is given in the following equations:

$$\begin{aligned} AvMessage(N) &= \frac{(N-1) \cdot 2}{N} \\ AvTraffic(N, size_m) &= \frac{(N-1) \cdot 2}{N} \cdot size_m \end{aligned} \tag{3.45}$$

In average, each peer sends and receives two messages during one update interval, a metric update and an ACK. With these in total four messages, the system statistics are gathered and disseminated among the peers and the tree structure is maintained. This means that each leaf sends and receives only one message, whereas every inner node sends $\beta + 1$ messages (β ACKs, 1 update) and receives as much as well (1 ACK, β updates).

Next, we describe the average number and sizes of transmitted messages by the peers according to their levels (i). The number of messages to be transmitted per peer per level depends on the number of Sub-Coordinators, that is, the number of peers one level below. As every peer in the tree, except the root, sends one metric update per update interval, differing traffic per tree level is caused by the number of

ACKs to be sent. The number of ACKs sent from level i equals the number of nodes on level $i+1$. The average number and size of messages per peer on level i, per metric update and ACK, equals:

$$\begin{aligned} AvMessage_i(N) &= 1 + \frac{E_{i+1}(N)}{E_i(N)} \\ AvTraffic_i(N, size_m) &= (1 + \frac{E_{i+1}(N)}{E_i(N)}) \cdot size_m \end{aligned} \tag{3.46}$$

The number of received messages can be determined analogously. A peer on level i receives $E_{i+1}(N)/E_i(N)$ metric updates and 1 ACK per update period. In a full tree, thus every non-leaf peer would send 1 metric update to its Parent-Coordinator and receive β metric updates from its Sub-Coordinators.

In conclusion, every peer has a limited node degree given by the branching factor β and has only one peer to push monitoring information to. With this limit, the load on the peers is bound. Despite the clarity of the protocol, it is very efficient and fault-tolerant.

3.5 MONITORING PEER-SPECIFIC INFORMATION

After having conveyed the creation and maintenance of the monitoring tree topology and how it is used to gather and disseminate system-specific statistics, we describe next the gathering and usage of the peer-specific information in the p2p network.

The peer-specific information is gathered for the purpose of enabling capacity-based peer search, as stated by the functional requirement **(**$Monitor^{peer}_{function}$ **2)**. This function allows users and peers to search for a specific number of peers (n) fulfilling a set of requirements, such as 5 peers having at least 400 MB of free storage and a upload bandwidth of at least 200 KB/s. The individual quantifiable characteristics of a peer we describe with the term *attribute*. The attributes of a set of peers cannot be aggregated without essential information loss; the size of the attribute information is proportional to the number of monitored peers.

Attributes describe the capabilities of an individual peer. We depict the list of considered attributes in Table 7. They are listed in a list format, termed *attribute-entry*. The main resources of a peer, CPU power, memory, storage space, and its bandwidth capacities, are listed. A weighted product of these values results in the *peerQuality* value, which allows for a simple comparison on the potential of peers and especially to apply an order on them. Besides the attributes, the attribute-entry contains also the peerID.

The information flow is similar to the one for gathering system-specific information: the attribute lists of the peers are sent to Parent-Coordinators, which concatenates them and propagates them further up in the SkyEye.KOM tree. Queries contain a field identifying the requester, defining the number of requested peers and a list of requirements for peer attributes. Peers address their queries to their Parent-Coordinator. The Parent-Coordinator checks locally whether it has information about n peers fulfilling the desired requirements. Then, it either replies with n peers fulfilling the criteria or it redirects the query one level higher in the tree. If no Coordinator in the tree can respond to the query, the root of the tree responds with a list of peers fulfilling the criteria (less than n). In order to avoid stressing peers with unbearable load, we introduce the concept of *Support Peers*, which are picked according to their capacities to support the duties of an overloaded Coordinator.

In the following section, we describe the set of attributes, how they are gathered in the network and how the queries are stated and processed. After this, we discuss the problem of load allocation and how the concept of Support Peers solves this problem.

3.5.1 *Protocol Design Decisions*

In order to enable capacity-based peer search, the individual attribute entries of the peers have to be gathered and prepared to match corresponding queries. The design decisions taken in the creation of the protocol for peer-specific information gathering and capacity-based peer search are introduced next.

S	P	Attribute - (S: Simulation View, P: Prototype)	Unit
General Attributes			
X	X	PeerID - ID of the corresponding peer	ID
X	X	Parent PeerID - ID of the Parent-Coordinator	ID
X	X	Current tree level of the node	#
X	X	Lifetime of the peer	#
Offered Capacities			
X	X	Offered download capacity	KB/s
X	X	Offered upload capacity	KB/s
X	X	Offered CPU cycles	#
X	X	Offered main memory	MB
X	X	Offered storage space	MB
Evaluation-specific Attributes			
X		GNP-Location Coordinates	5 Doubles
	X	The operating system of the peer	String
	X	The system architecture of the node	String
	X	The current version of the node's Java VM	String
	X	The lower bound of the node's responsibility range	ID
	X	The upper bound of the node's responsibility range	ID
Attributes relevant for the Tree Extension with Support Peers			
X	X	Maximum capacity for receiving attribute entries	#
X	X	Maximum capacity for sending attribute entries	#
X	X	Threshold for attribute entries to pick a Support Peer	#
X	X	Maximum capacity for receiving attribute entries of the Support Peer	#
X	X	Maximum capacity for sending attribute entries of the Support Peer	#

Table 7: List of Monitored Attributes/Capacities

Monitoring Scope

The monitoring of the available capacities and peers' statuses in the p2p system may either consider all peers in the p2p network or allow a limited view on a certain subset in the network. Although monitoring the whole p2p network is interesting from a statistical point of view, for the provision of capacity-based peer search it does not provide any further benefits. The problem statement we aim to resolve is finding suitable peers matching to a query. Whether these peers are picked from a large or small set of monitored peers is irrelevant, assuming that the suitable peers are in the potential set. Thus, we relax the requirement and allow a limited view on the peers in the network, both to address the load limits of individual peers and due to the relaxed requirements of the problem statement. However, we ensure that through the shrinking of the observed set, only "bad" peers are removed from the set of potential peers, while good peers are kept.

Reactive vs. Proactive Information Gathering

The information about peers can be gathered proactively or reactively. A proactive solution gathers the information continuously over time and responds quickly to queries. This results in constant traffic overhead for monitoring, but low traffic cost and low delay for query processing as the information is prepared. Reactive solutions, on the other hand, gather the required information every time a query is started. This option requires less traffic overhead in the case of infrequent queries in the p2p network. However, we assume that many queries are stated in a large-scale p2p network and that low response times are crucial for this functionality. Thus, a proactive option has been chosen for gathering the peer-specific attributes. In order to limit the traffic overhead for monitoring, we also use traffic limits (maximum capacities) that can be set by each peer individually.

Push vs. Pull
Peer-specific information may be pulled or pushed in the SkyEye.KOM tree towards the Parent-Coordinator. Here, the same principle applies as for monitoring system-specific information. Following a pull-based approach is difficult as it requires each Coordinator to identify every potential Sub-Coordinator in the unified ID space. Every peer is Coordinator in various Domains and, thus, a large list of potential Sub-Coordinators exist. A push-based approach is easier and more fault-tolerant as only one single Parent-Coordinator has to be identified based on the tree topology of SkyEye.KOM.

Reactive vs. Proactive Information Provision
We follow a proactive monitoring approach for the peer-specific information gathering. Thus, the information on the quality of the peers is assumed to be pre-processed and stored in the tree. A proactive approach for information provision would push this information periodically to the peers, causing large traffic overhead. However, as the peers are expected to state specific queries, most of the pushed information would be useless. Thus, this peer-specific monitoring information is used reactively for queries. This generates only traffic overhead in the case of queries being stated.

Recursive vs. Iterative Query Processing
Stating a query for capacity-based peer search can be either processed recursively or iteratively. In an iterative approach, a query is directly processed and replied by the contacted peer, which returns an intermediate result and a next peer ID to query. With this, the requesting peer has to initiate as many queries as needed to retrieve the desired number of peers searched for, while reusing intermediate results. In a recursive approach, the queried peers are forwarding the query along the tree until the desired number of peers is found. Only one result message is then sent to the requesting peer. While the iterative approach brings earlier intermediate results, it requires the query initiating peer to contact every peer along the query path independently, thus causing time and traffic overhead for connection establishment for sending intermediate results frequently. The recursive solution, on the other hand, uses pre-established links in the SkyEye.KOM tree and leads to only one result message per query with the desired set of peers more quickly. In SkyEye.KOM, the recursive approach is used to handle capacity-based peer search queries.

Heterogeneous vs. Equal Roles
For monitoring system-specific information in the p2p network, the heterogeneity of the peers has not been taken into account, as aggregation of the monitoring information leads to a similar load on all peers. For the peer-specific information, aggregation cannot be used, as the attribute-entries of each individual peer need to be kept. Thus, a larger Domain, i.e. at higher levels in the tree, also leads to a larger data set regarding the peer-specific information. The information of the Sub-Domains are then concatenated. While information is gathered, the information size increases with the size of the observed Domain, and thus capable peers are needed to carry the load. However, this is in conflict with the aim of allocating peers to Coordinator positions in the tree based on their IDs. In order to support heterogeneity in the network, we introduce the concept of Support Peers. These are capable, strong peers in the corresponding Domain that are picked by the Coordinator to take over a part of the load. Having strong peers in strongly loaded positions in the tree leads to a tree of strong peers, which is capable of carrying as much load as possible.

One Single Support Peer vs. Many Support Peers per Domain
In order to fully process all information of all peers in the network, one single Support Peer per Domain may not be enough. Several Support Peers per Domain, however, would be able to bear all occurring load. On the other hand, having more than one Support Peer transforms the tree into a mesh that requires synchronization among the Support Peers. In the worst case, at the root level several hundreds Support Peers could be active that need a timely and efficient synchronization and maintenance protocol. Obviously, there is a trade-off and a scalability issue with the idea of having multiple Support Peers per Domain. In order to have an efficient, flexible and deterministic information gathering protocol, the

protocol uses just one single Support Peer, leading to a near optimal peer to position assignment in the tree.

Having seen the design decision regarding the handling of peer-specific monitoring information, next, we present the protocol for gathering peer attributes in SkyEye.KOM and performing capacity-based peer search.

3.5.2 *Protocol for Monitoring Peer-specific Information*

The protocol for gathering and querying peer-specific information uses the SkyEye.KOM tree topology to exchange information between Coordinators and Parent-Coordinators. *Attribute updates* are used to gather the attribute information of peers in a Domain. Attribute updates contain the attribute-entries of peers of the corresponding Domain, they are periodically transmitted via the SkyEye.KOM tree towards the root. The duration of a period between two attribute updates is determined by the attribute update period interval, $t_{AttPeriod}$.

Figure 19 depicts the update protocol for monitoring peer-specific information. Please note that this protocol is the core solution, which is later extended to support the heterogeneity of the peers while allocating load to more capable Support Peers.

GATHERING PEER-SPECIFIC INFORMATION IN THE TREE

In the following, we present the protocol for gathering peer-specific information in the SkyEye.KOM tree and refer for the numbering to Figure 19.

1. **Receiving attribute-entries of lower Domains, sending ACKs**: A Coordinator may receive a report from Sub-Coordinators containing the attribute-entries of the best peers in their Domain; these are stored locally. These updates are acknowledged with a message that attests to the existence of the peer and contains information about how many entries it is able to receive ($recvAtt_{max}$) in the next attribute update. This concept of a *receiver window* is also used in TCP [KC81], which controls the information flow and limits congestion on the receiver side.

   Upon every attribute update, peers check the entries in their table of Sub-Coordinators, from which they received attribute updates. If an update is older than a given threshold termed t_{oldAtt}, the entry is removed. This refreshing of the table entries prevents the node from forwarding attribute-entries of Sub-Coordinators, which may already have a new Coordinator or have gone offline. Similar to the threshold $t_{oldMetrics}$, t_{oldAtt} can be chosen dynamically or is calculated as a multiple of the attribute update interval, $t_{AttPeriod}$. Simulations with different multipliers have shown that a good value for the multiple of $t_{AttPeriod}$ ranges between 1.5 and 2.

2. **Measuring of own attributes**: Every node measures its own attributes (see Table 7) and creates an attribute-entry including the *peerQuality*. A sensor module assumed in dependency **(**$Monitor^{peer}_{requirement}$ **1)** provides the desired attributes; it is not further discussed in the scope of the monitoring architecture.

3. **Preparation of attribute-entries for own Domain**: The attribute-entry of the peer is combined with a subset of entries, which it received from its Sub-Coordinators. These injections of the local measurements implement the function **(**$Monitor^{peer}_{function}$ **1)**. This combined information is stored locally and used for the queries to find matching peers. Depending on its network connection, this subset can range from just a small amount to all attribute-entries of the corresponding Domain. As every peer has an individual maximum sending capacity, $sendAtt_{max}$, a maximum number of entries can only be sent in an attribute update. If only a part of the whole data is to be sent, the *peerQuality* of each attribute-entry is used as an order to determine the "best" peers. The entries of the "best" peers are selected to be sent with the next attribute update.

4. **Push of attribute-entries to Parent-Coordinator**: An attribute update is pushed to the Parent-Coordinator, which was calculated during the last metric update. Here, the established and maintained SkyEye.KOM tree topology is used, while saving communication overhead. The attribute update is sent to the Parent-Coordinator, respecting its maximum download capacities

$recvAtt_{max}$. Thus, the upload limit of the sending Coordinator and the download limit of the receiving Parent-Coordinator are not exceeded. Concerning the tasks of the root of the SkyEye.KOM tree, the root does not need to do anything, since no Parent-Coordinator is available and no further evaluation of the attribute-entries is required.

5. **Acknowledging received attribute-entries**: As the attribute update is received by the Parent-Coordinator, an ACK is sent. The ACK contains a simple validation of the existence of the Parent-Coordinator, it is very small. As the corresponding ACK is received, the current attribute update terminates. Otherwise, the Parent-Coordinator is recalculated and a retransmission is executed.

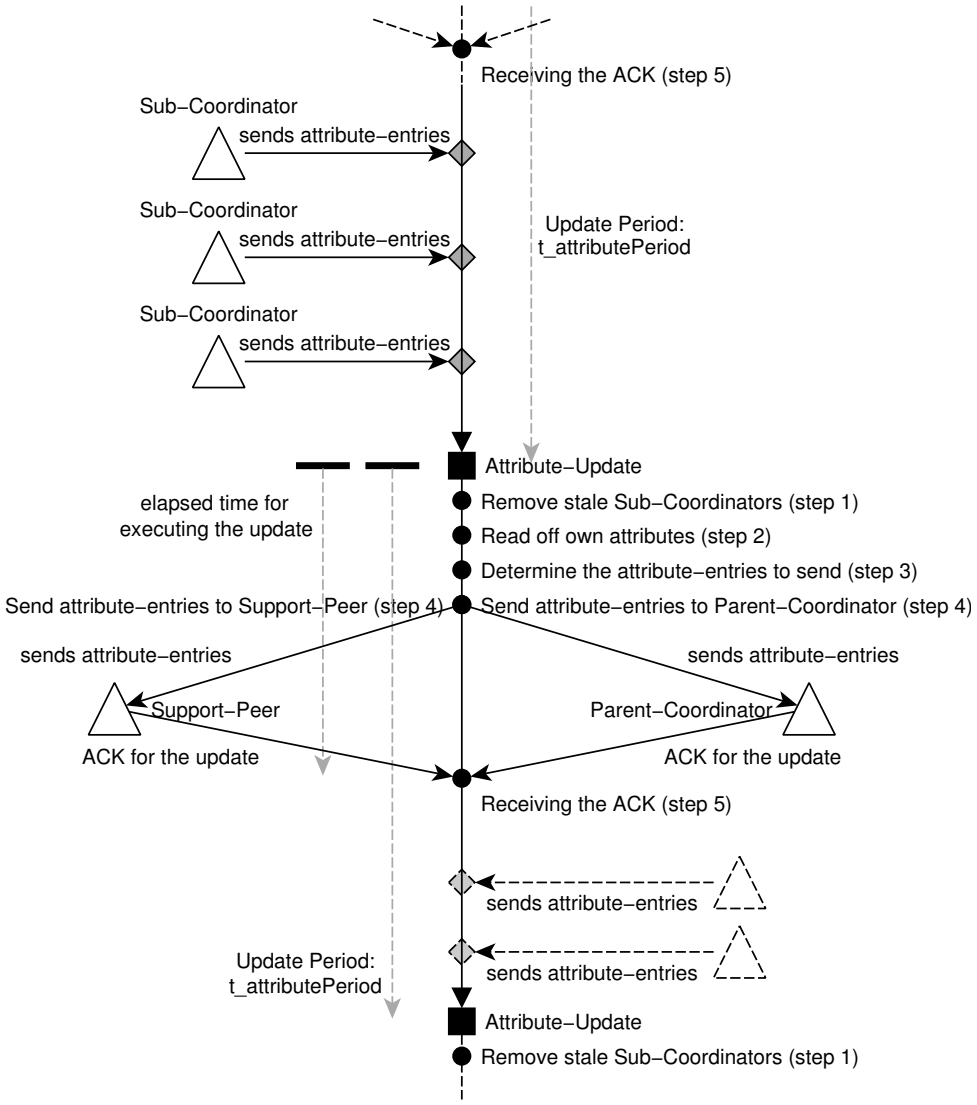


Figure 19: Protocol for Gathering Peer-specific Information in the SkyEye.KOM Tree

In contrast to the metric updates with aggregatable statistics, the peer-specific attribute entries are concatenated and thus grow in their size on their way to the root. However, the load is increasing on the peers in higher positions on the tree. Figure 20 visualizes the number of received attribute entries, which depends on the level in the tree and grows in size while approaching the root. As a negative example for this circumstance, the root, its children and grandchildren may consist of weak peers, leading to the loss of attribute information in these levels. In order to avoid the congestion of network resources and to support individual peer load limits, we introduce the concept of Support Peers that takes effect if the individual load limit of a peer is exceeded.

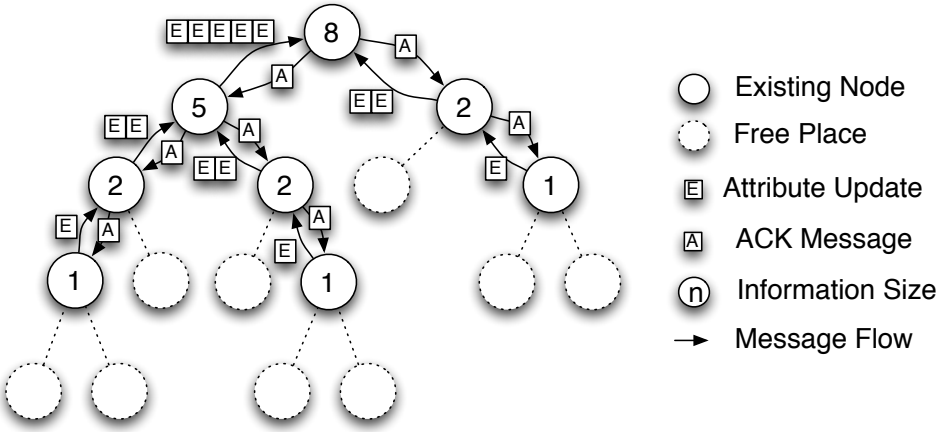


Figure 20: Load Level during Attribute Gathering

SUPPORTING THE HETEROGENEITY OF THE PEERS

Every peer in the SkyEye.KOM tree reserves a fraction of its bandwidth for the attribute updates, whereby the peers avoid the whole bandwidth being utilized for the transmission of attributes. A peer may measure its maximum available bandwidth according to [LFV08, RF08]. The two fractions indicating the bandwidth contribution for attribute updates regarding the up- and download capacities of a peer p are termed $AttUpFrac_p$ and $AttDownFrac_p$ The reserved bandwidth is termed $sendAtt_{max}$ and $recvAtt_{max}$, correspondingly. Please note that this amount may vary over time. This fraction of bandwidth is translated into the amount of attribute-entries that the peer is willing to handle. The exact function is not decisive, as it only indicates an estimation of the number of attribute-entries to process. The following formula is a possible example of how to calculate the translation from the reserved fraction of the download-bandwidth to the maximum amount of entries that a peer can receive:

$$recvAtt_{max} = \frac{AttDownFrac_p \cdot MAX\_Network_{Down_p} \cdot t_{AttPeriod}}{\overline{AttEntrySize}} \tag{3.47}$$

Correspondingly, this formula relates to the maximum number of entries a peer can send in an attribute update period:

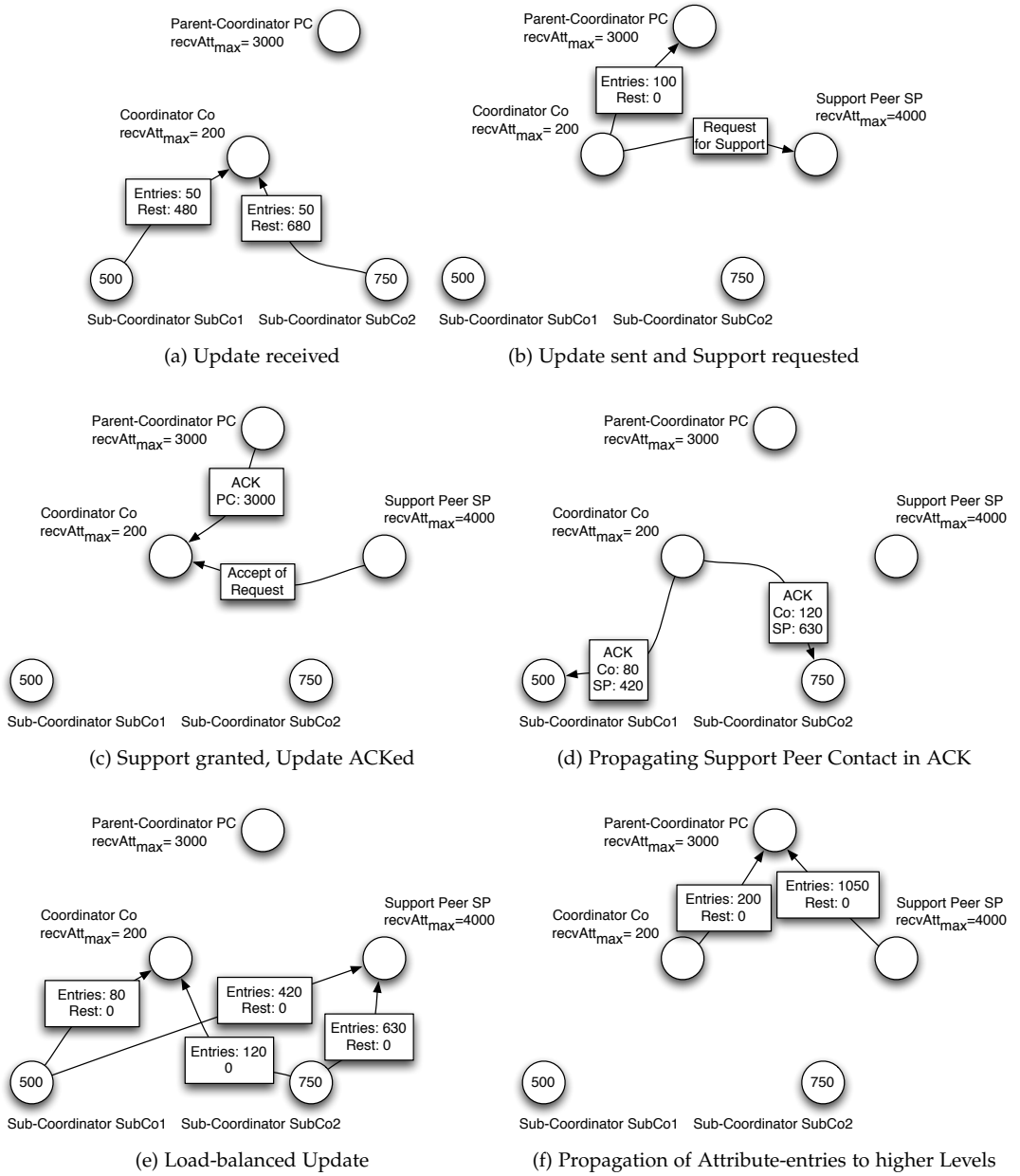
$$sendAtt_{max} = \frac{AttUpFrac_p \cdot MAX\_Network_{Up_p} \cdot t_{AttPeriod}}{\overline{AttEntrySize}} \tag{3.48}$$

These contribution limits are an essential consideration in the attribute update protocol in order to support the maximum load limits of the peers. If the amount of attribute-entries that a Coordinator actually manages is smaller than or equal to $sendAtt_{max}$ and $recvAtt_{max}$, the Coordinator can send and receive all attribute-entries. The terms $recvAtt_{free}$ and $sendAtt_{free}$ describe the amount of attribute-entries, a Coordinator can receive or send before reaching its limits $recvAtt_{max}$ and $sendAtt_{max}$.

In order to comply the sending limit ($sendAtt_{max}$), Coordinators which cannot send all of the attribute-entries, i.e. exceed $recvAtt_{max}$ and $sendAtt_{max}$, send just a subset attribute entries to their Parent-Coordinators. The subset consists of the best peers according to the *peerQuality* metric. For controlling the amount of received attribute-entries, a receiver window approach is used, similar to TCP [KC81]. The first attribute update to a new Coordinator only contains a few attribute entries. The corresponding acknowledgment message contains the receiving window size ($recvAtt_{max}$) of the Parent-Coordinator. Further attribute updates consider this information to limit the maximum size of attribute updates.

This approach solves the requirement of supporting the individual peer load limitations and avoiding peer overload. However, attribute-entries that are not propagated up in the tree are lost for higher levels. As a next step, we introduce the concept of *Support Peers* which address the issue that weak peers may be overloaded in higher positions in the tree and strong peers may be "underloaded" in lower positions

in the tree. The main idea is that weak Coordinators ask strong peers in their Domain to take over as much load as possible, without being themselves overloaded. Through this, both the weak and strong peers are utilized more efficiently and a large fraction of peer-specific information is gathered in the Domain whilst complying with the maximum peer load limits. In Figure 21, we present the adapted attribute gathering protocol.



(a) Update received

(b) Update sent and Support requested

(c) Support granted, Update ACKed

(d) Propagating Support Peer Contact in ACK

(e) Load-balanced Update

(f) Propagation of Attribute-entries to higher Levels

Figure 21: Propagation of Attribute Updates in the Tree including Support Peers

Besides the maximum load limits $recvAtt_{max}$ and $sendAtt_{max}$, we further introduce a threshold bandwidth capacity ($recvAtt_{thresh}$) that triggers the invocation of Support Peers. The threshold $recvAtt_{thresh}$ is slightly below $recvAtt_{max}$ (e.g. $recvAtt_{thresh} = 0.8 \cdot recvAtt_{max}$). In the case that attribute-entries sent by the Sub-Coordinators exceed this threshold, a Support Peer is activated before the acknowledgment is sent. To enable the transmission of the left attribute-entries, the Coordinator searches for an appropriate Support Peer that it can introduce as a new receiver to the Sub-Coordinators. We depict the algorithm of picking a Support Peer and including it in the information gathering process in Figure 21. The algorithm is as follows:

- Among its stored attribute-entries of the monitored peers, the Coordinator chooses the peer with the *second* highest *peerQuality*. Here, the Coordinators also consider the $recvAtt_{free}$ and $sendAtt_{free}$ values of the peers to be chosen. The chosen peer is a candidate for the Support Peer position and a support request is sent to this peer. The request lists the desired support capacities in terms of upload and download bandwidth, the ID of the requesting Coordinator, its Parent-Coordinator ID and a time period for which the reservation is made
- The nominee must accept this request, the Support Peer selection algorithm makes sure that the candidate Support Peer is not already a Support Peer of another Domain.
- The Support Peer accepts the request and reserves the desired amount of up- and download bandwidth. It adapts its attribute-entry to match the reservation and introduces a second Parent-Coordinator regarding the attribute updates. From now on until the reservation lasts, the Support Peer sends the attribute updates received in its role as Support Peer to the new Parent-Coordinator.
- An acknowledgment message is created, containing the receiver window $recvAtt_{max}$ of the Coordinator and the receiver window of the new Support Peer, as well as its peer ID. Additionally, a time period is denoted, which advises the Sub-Coordinator to send for this period additional messages to the new Support Peer.
- The Sub-Coordinator sends the attribute-entries exceeding the receiver window of its Parent-Coordinator to the Support Peer. The quality of the monitored peers in the attribute updates has the same distribution in both attribute update messages.
- If the period of a Support Peer is over, the Coordinator might be able to handle the actual amount of data, but if not, a new Support Peer is assigned.

Please note that a Coordinator accepting the request to act as Support Peer is represented two times in the SkyEye.KOM tree and has to play two roles. Besides the tasks of a Coordinator, it additionally accomplishes the functions of a Support Peer. For this reason, it must determine which of the two nodes in the tree is addressed when a message arrives. Then, the received data is separately processed and may not be exchanged. Finally, the corresponding Parent-Coordinator in the tree has to be found for the information propagation.

Every Coordinator receives information from its Sub-Coordinators, among others, about the best peers in the corresponding Domains. Thus, from these two best peers the *second* best can be used as the Support Peer and the best peer is passed to the Parent-Coordinator. With this approach, at the top of the tree, the best peer is available as the Support Peer and at every level the second best peer of the Domain. Consequently, the peer capacities in the network are utilized very efficiently, as the most suitable peers are placed at positions in the tree where most of the load occurs. The peer-to-position assignment is nearly optimal. However, as some of the Support Peers might not be used at higher positions, their potential is wasted on lower levels, and thus in these cases the assignment is not optimal.

EFFECTS OF CHURN ON COORDINATORS AND SUPPORT PEERS

When a Coordinator distributes the load to its Support Peer, the Sub-Coordinators send one part to the Coordinator and one to the Support Peer. As the Sub-Coordinators may have Support Peers as well, a constellation appears in which at maximum $2 \cdot \beta$ peers send attribute updates to two peers one level higher in the tree. The protocol is depicted in Figure 21e and Figure 21f. During the existing of this constellation between a Coordinator, a Support Peer, a Parent-Coordinator and the Sub-Coordinators four incidents can occur.

1. A new peer takes over the Parent-Coordinator's position: it is correctly addressed by both the Coordinator and the Support Peer as they send attribute updates to the corresponding Parent-Coordinator being responsible for the Domain Key.
2. The Coordinator goes offline: a new peer takes over the Coordinator position. It decides on its own whether it can handle the incoming attribute updates or whether it needs a Support Peer as well. In its acknowledgment message it informs the Sub-Coordinators about the existence of a Support

Peer. Since it is possible that the new Coordinator can handle the amount of attribute-entries, the current Support Peer of the old Coordinator is not addressed any longer and can cease its deployment.

3. A Support Peer goes offline: the corresponding Coordinator has to be informed by its Sub-Coordinators. As a consequence of this incident, it must choose a new Support Peer and provide the new one to its Sub-Coordinators. If the Coordinator can handle the actual amount of data by itself, the need for a new Support Peer expires.

4. All Sub-Coordinators of a Coordinator go offline or if a Support Peer receives no updates for a longer period: the supporting node can cease its deployment.

As a result of the protocol considered now, the attribute information of the peers are gathered, the load limits of the peers are not violated and, using Support Peers, the available bandwidth capacities are efficiently utilized. The gathering of the capacity information of all the peers in the network is done by the most capable peers in a tree structure, resulting in a tree height of $O(\log_\beta(N))$ and information age at the root of $O(\log_\beta(N) \cdot t_{AttPeriod})$. Based on this gathered information, we are now able to perform capacity-based peer search queries.

3.5.3 *Query Processing Protocol for Capacity-based Peer Search*

Having gathered the information on peer capacities in the whole p2p network using the SkyEye.KOM tree, we now introduce the notation for the queries and how they are processed. The queries implement the function **(**$Monitor^{peer}_{function}$ **2)**, enabling capacity-based peer search. A query contains information about the number of desired peers and constraints regarding their capabilities. A peer could state a query, such as seeking five peers with an upload-bandwidth faster than 6,000 KB/sec and a storage space larger than 80 GB. As a result, the requesting peer would receive a list of five peer IDs.

Every query consists of a unique query ID, the ID of the requesting peer, the number of desired peers and a clause containing one or more conjunct conditions. A condition names the attribute, its type, a value and an operator, which compares the capacity of a peer with the defined value. The language of the query allows for the usage of $<$, $>$ or $=$ as operator. An example condition within the clause is: *MAX_NETWORK_UP > 1000 Double*. The amount of requested peers must fulfill the conditions of the clause, such as *6 of FREE_DISK_SPACE > 512000 Integer; TIME_ONLINE < 3600 Integer*. A query is solved when enough peers are found that satisfy the clause, as depicted in the following protocol.

The query is resolved as depicted in Figure 22 in the case that a peer cannot resolve the query locally:

1. A peer originates a query with a unique *query ID* and sends it to its Parent-Coordinator.

2. The receiving peer checks, whether it can solve the complete query and return the results to the originator if all requested peers were found. Otherwise, it adds the peer IDs of suitable matches to the query and forwards the message to the Support Peer in its Domain or to the Parent-Coordinator if it has no Support Peer or if it is the Support Peer itself. A Support Peer forwards the query to the designated Parent-Coordinator.

3. Step No. 2 is repeated until the query either reaches a peer that completes the number of desired peers fulfilling the requested conditions or in the case the query is unsolvable it reaches the root. In the case that the query is solvable, the result list is sent back to the querying peer. In the latter case, the root sends the incomplete result list back to the querying peer.

4. In order to not assign the same resources of the peers to various query initiating peers, the attribute-entries of each peer in the result list are adapted, assuming that these resources will be used or reserved immediately.

Queries may address different quality levels of peers. Easily solved queries are answered at lower levels in the tree, thus avoiding the creation of additional load on the higher levels. Only hard to solve queries, seeking for very capable peers, traverse up the tree. Queries traverse the tree and pass at maximum two peers per level, thus resulting in a query hop count complexity of $O(t_{AttPeriod} \cdot \log_\beta(N))$.

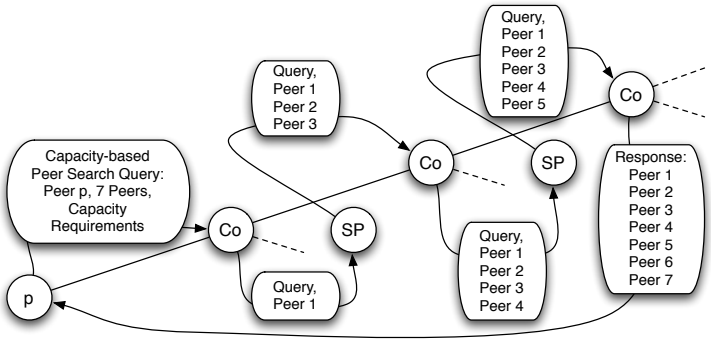


Figure 22: Recursive Query-Solving in the SkyEye.KOM Tree

A	S	P	Statistic - (A: Analytic Model, S: Simulation View, P: Prototype)	Unit
General Statistics - 5 values for all statistics: Count, Min, Max, Sum, Sum of Squares				
X	X	X	Number of Support Peers	#
X	X	X	Number of stored attribute entries in tree	#
X	X	X	Number of stored attribute entries at root	#
X	X	X	Number of stored attribute entries per level	#
X	X	X	Peer quality of Coordinators	#
X	X	X	Peer quality of Support Peers	#
Capacity-based Peer Search - 5 values each				
X	X		Delay for resolving queries	s
X	X	X	Number of hops for resolving	#
X	X	X	Number of initiated queries	1/s
X	X	X	Level of initiated queries	#
X	X	X	Level of solved queries	#
Overhead per Message Type - 20 values each: sent and received messages and traffic				
X	X	X	Attribute update overhead	1/s,KB/s
X	X	X	Query overhead	1/s,KB/s

Table 8: List of Statistics on Capacity-based Peer Search

3.5.4 *Model of the Peer Monitoring Protocol*

In this section, we present the model for the protocol for gathering and querying peer-specific information (i.e. attributes). In order to do so, first the concept of Support Peers is modeled, as it affects the monitoring topology. Then, we discuss the costs of gathering peer attributes and the performance of the capacity-based peer search. An overview of the used variables is given in Table 9.

EFFECTS OF THE SUPPORT PEERS ON THE MONITORING TOPOLOGY

The deterministic assignment of Domains to Coordinators on the basis of the ID space does not take into account the capabilities of a peer. The Coordinator decides to select a Support Peer when the number of attributes in the attribute updates and thus the consumed bandwidth exceeds a certain threshold. We calculate the probability that a Support Peer is needed based on the load and capacities of a Coordinator. The number of Support Peers is then used to model the induced delay and traffic costs.

First, we depict the expected distribution of capabilities among the peers, or in other words, the number of attributes they are willing to handle. We set this distribution in relation to the occurring load by the attribute updates. Thus, we model the probability of Coordinators that cannot handle the load and pick a Support Peer.

$N \in \mathbb{N}$	Number of nodes in the monitoring tree
$E_i(N) \in \mathbb{R}$	Expected number of nodes on level i when there are N nodes in the monitoring tree
$\sigma_i \in \mathbb{R}$	Portion of peers which cannot handle the amount of traffic on level i, used to determine the need for support peers
$E_{i_{Support}}(N, \sigma_i) \in \mathbb{R}$	Expected number of nodes on level i including support peers
$size_{attr} \in \mathbb{R}$	Size of an attribute entry
$TrafficAttrReceived_i(N, size_{attr}) \in \mathbb{R}$	Traffic received by peers on level i, used to determine the need for support peers
$AvTrafficAttrReceived_i(N, size_{attr}) \in \mathbb{R}$	Average traffic received by a peer on level i, used to determine the need for support peers
$NoAttr_i(N, \sigma_i) \in \mathbb{R}$	Number of attribute updates in an update interval of peers on level i
$NoAttrACK_i(N, \sigma_i) \in \mathbb{R}$	Number of ACKs on attribute updates in an update interval of peers on level i
$NoAttrTotal(N, \sigma_i) \in \mathbb{R}$	Number of sent attribute updates and ACKs in an update interval
$AvMessageAttr(N, \sigma_i) \in \mathbb{R}$	Average number of messages (attribute updates and ACKs)
$AvAttrUpdate_i(N, \sigma_i) \in \mathbb{R}$	Average number of attribute update sent by a peer on level i
$AvAttrACK_i(N, \sigma_i) \in \mathbb{R}$	Average number of ACKs on attribute updates sent by a peer on level i
$TrafficAttr_i(N, size_{attr}) \in \mathbb{R}$	Traffic caused by attribute updates sent by peers on level i in an update interval
$TrafficAttrACK_i(N, \sigma_i, size_{attrACK}) \in \mathbb{R}$	Traffic caused by ACKs sent by peers on level i in an update interval
$AvTrafficAttr_i(N, \sigma_i, size_{attr}, size_{attrACK}) \in \mathbb{R}$	Average traffic of a peer on level i caused by sent attribute updates and ACKs in an update interval
$TrafficAttrTotal(N, \sigma_i, size_{attr}) \in \mathbb{R}$	Traffic caused by sent attribute updates and ACKs in an update interval
$AvTrafficAttr(N, \sigma_i, size_{attr}) \in \mathbb{R}$	Average traffic

Table 9: Variables used to Describe the Monitoring of Peer Attributes

The distribution of capabilities of peers is expected to be log-distributed. We consider discrete capability classes cl_k, representing peers being able to cope with α^k entries. The probability of a peer having the capability to process $cl_k = \alpha^k$ attributes is expected to be:

$$ProbCapability_{cl_k} = 1/\beta^{\log_\alpha(cl_k)} \tag{3.49}$$

To give an example: Half of the peers belong to the class of smallest capability class (2), a quarter of peers belongs to the next class (4) etc. and only a very small fraction comprises high quality. The average capability $AvgCapability$ of a Coordinator is:

$$AvgCapability(N) = \log_\alpha(N) \tag{3.50}$$

In order to calculate the load per Coordinator, we model the number of attributes per level sent per attribute update message. The maximum amount of attributes to be handled by a peer on level i equals the number of peers in its sub-tree. The number of peers in a Domain of a Coordinator at level i is:

$$PeersInDomain_i(N) = \sum_{k=i+1}^{N} E_k(N)/E_i(N) \tag{3.51}$$

The expected number of peers with given capabilities α^k in an attribute-entry set on level i is represented as a table with following entries:

$$CapabilityDistribution_i(N, k) = PeersInDomain_i(N) \cdot ProbCapability(\alpha^k) \tag{3.52}$$

Capability class	2	4	8	16	32	64	128
Number of peers	16000	8000	4000	2000	1000	500	250
Percentage(%)	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078
Capability class	256	512	1024	2048	4096	8192	16384
Number of peers	125	62.5	31.25	15.63	7.81	3.91	1.95
Percentage(%)	0.0078	0.0039	0.0020	0.0010	0.0005	0.0002	0.0001

Table 10: Capacity Distribution Model of 32000 Peers

The table lists for all levels and all capacity categories the expected number of peers in a tree with N peers. In order to make the concept clear, we give an example for the capacity distribution in Table 10 for $N = 32000$ peers. The maximum number of received attribute entries is equal to the number of peers in the domain minus the Coordinator.

$$MaxAttrRecv_i(N) = PeersInDomain_i(N) - 1 \tag{3.53}$$

Let $size_{attr}$ be the size of an attribute entry, then the size of an attribute update message received by all peers on level i is in average:

$$TrafficMaxAttrRecv_i(N, size_{attr}) = MaxAttrRecv_i(N) \cdot size_{attr} \tag{3.54}$$

The ration σ_i of peers on level i not being able to cope with the load and needing a Support Peer is:

$$\sigma_i(N) = 1 - ProbCapability(MaxAttrRecv_i(N)) \tag{3.55}$$

For each level i, we calculate the percentage of peers, σ_i, which cannot handle the corresponding amount of attributes. This value equals the probability that Support Peers are required. The expected number of Support Peers per level in a tree with N peers is:

$$\begin{aligned} E_{i_{Support}}(N) &= ((1-\sigma_i)\cdot 1 + \sigma_i \cdot 2) \cdot E_i(N) \\ &= (1+\sigma_i) \cdot E_i(N) \end{aligned} \tag{3.56}$$

In the following, we investigate the effects of Support Peers on the performance and the costs for gathering peer-specific information.

MESSAGE OVERHEAD FOR MONITORING PEER-SPECIFIC INFORMATION

Using Support Peers increases the costs on average as more capable peers are utilized on higher positions in the tree. In comparison to the monitoring of system-specific information, the attribute lists gathered are also larger. The following equation gives the number of attribute update messages sent by peers which are situated on level i in one update interval.

$$\begin{aligned} NrAttrSent_i(N, \sigma_i) &= E_{i_{Support}}(N) \cdot (1+\sigma_{i-1}) \\ &= (1+\sigma_i) \cdot E_i(N) \cdot (1+\sigma_{i-1}) \end{aligned} \tag{3.57}$$

The first term describes the amount of nodes on level i including Support Peers. The second term implies that messages are not only sent to the Parent-Coordinator but also to Support Peers on level $i-1$. The number of sent ACKs from level i is similar, as each update message is acknowledged:

$$\begin{aligned} NrAttrSentACK_i(N, \sigma_i) &= E_{i_{Support}}(N) \cdot (1+\sigma_{i+1}) \\ &= (1+\sigma_i) \cdot E_i(N) \cdot (1+\sigma_{i+1}) \end{aligned} \tag{3.58}$$

The first term again is the number of peers which are positioned on level i and the second term describes the number of children and the Support Peers of these children which expect to receive an ACK message. When dividing the expressions Eq. 3.57 and Eq. 3.58 by $E_i(N)$ we obtain the average number of attribute update and ACKs sent by a peer on level i:

$$\begin{aligned} AvAttrSentUpdate_i(N, \sigma_i) &= (1+\sigma_i) \cdot (1+\sigma_{i-1}) \\ AvAttrSentACK_i(N, \sigma_i) &= (1+\sigma_i) \cdot (1+\sigma_{i+1}) \end{aligned} \tag{3.59}$$

Summing up the number of attribute update messages and ACKs on all levels, the total amount of sent messages is obtained:

$$\begin{aligned} NrAttrSent(N, \sigma_i) &= \sum_{i=1}^{N} E_{i_{Support}}(N) \cdot (1 + \sigma_{i-1}) \\ NrAttrSentACK(N, \sigma_i) &= \sum_{i=0}^{N} E_{i_{Support}}(N) \cdot (1 + \sigma_{i+1}) \end{aligned} \tag{3.60}$$

TRAFFIC COSTS FOR MONITORING PEER-SPECIFIC INFORMATION

Having discussed the number of messages sent in the tree, in the following, we also focus on their size. Next, we regard the traffic generated by sending the attribute information towards the root. As mentioned, when a Coordinator receives a message containing attributes from his children, it appends its attributes to the existing ones and sends them to its Parent-Coordinator. In order to model the traffic overhead, we need to model the size of the messages and, thus, the number of attribute entries a Coordinator/Support Peer team is able to bear per level. A Support Peer has always the second highest capability out of all peers in the corresponding Domain; thus the highest capability level has to be identified whereby more than 2 peers exist in the Domain of the Coordinator/Support Peer team. Equation 3.52 introduced the variable $CapabilityDistribution_i(N, k)$, giving the expected number of peers with a capability class of α^k in a Domain located at level i, in a network with N peers. Thus, the expected capability of a Support Peer at level i in a network with N peers is with given α and i:

$$\begin{aligned} CapSP_i(N) = &\alpha^k \text{ with } k \in \mathbb{N} \text{ and} \\ &CapabilityDistribution_i(N, k) > 2 \text{ and } CapabilityDistribution_i(N, k+1) < 2 \end{aligned} \tag{3.61}$$

In total, a Coordinator / Support Peer team at level i can bear the following amount of attribute entries:

$$CapCoordSP_i(N) = AvgCapability(N) + CapSP_i(N) \tag{3.62}$$

However, the actual received load by the Coordinator / Support team, $LoadCoordSP_i(N)$, is the maximum out of the number of attribute-entries $MaxAttrRecv_i(N)$ for the corresponding Domain and the capability limit of the Coordinator / Support Peer team, $CapCoordSP_i(N)$:

$$LoadCoordSP_i(N) = max(CapCoordSP_i(N)), MaxAttrRecv_i(N)) \tag{3.63}$$

Let $size_{attr}$ be the size of a single attribute entry and $size_{ACK}$ the size of an acknowledgment message. Then the traffic load per Coordinator / Support Peer team on level i is the load of them multiplied with the size of a single attribute entry.

$$TrafficRecvCoordSP_i(N, size_{attr}) = LoadCoordSP_i(N) \cdot size_{attr} \tag{3.64}$$

This can be differentiated for Coordinators and Support Peers:

$$\begin{aligned} &TrafficRecvCoord_i(N, size_{attr}) = \\ &AvgCapability(N)/CapCoordSP_i(N) \cdot TrafficRecvCoordSP_i(N, size_{attr}) \\ &TrafficRecvSP_i(N, size_{attr}) = \\ &(1 - AvgCapability(N)/CapCoordSP_i(N)) \cdot TrafficRecvCoordSP_i(N, size_{attr}) \end{aligned} \tag{3.65}$$

The traffic for sending attribute updates and acknowledgments for received attribute updates per Coordinator / Support Peer team on level i is modeled as follows:

$$\begin{aligned} &TrafficSentCoordSP_i(N, size_{attr}) = \\ &\quad min(LoadCoordSP_{i-1}(N), LoadCoordSP_i(N)) \cdot size_{attr} \\ &\quad + size_{ACK} \cdot (1 + \sigma_i(N)) \cdot \frac{E_{i+1}(N)}{E_i(N)} \end{aligned} \tag{3.66}$$

Until now, we have modeled the costs generated by proactive gathering of peer-specific information.

MONITORING SCOPE AND PERFORMANCE OF INDIVIDUAL QUERIES

In this part, we model the performance of gathering peer-specific information in the p2p system and using this information for capacity-based peer search. The main metrics relevant for the performance of the approach are:

- The amount of attributes-entries stored in average per level ($LoadCoordSP_i(N)$)
- The quality of the peers listed in the attributes-entries per level ($CapabilityDistribution_i(N,k)$)
- The query resolving peer distribution
- The distribution of hops needed for resolving a query

The first two metrics determine the amount of information available and the quality of the peers listed. These metrics describe what and how many peers are offered for the capacity-based peer search. The queries requesting a specific amount of peers with a given quality will be matched to these. The performance of the capacity-based peer search is expressed in the latter two metrics.

The freshness of the information is the same as in the monitoring of system-specific information. Although, due to the Support Peers concept additional nodes have been introduced that have to be passed towards the root, only the number of tree levels and the length of the update intervals (UI) are crucial. Updates are sent in parallel to the Coordinator and Support Peer.

The capability of the Coordinators and Support Peers limit the amount of stored attribute-entries. However, the actual number of peers in the Domain may be larger. In that case, peers with a higher (peer) quality are kept. In our model, we store and process attribute entries of peers of better quality classes until the contingent of $CapCoordSP_i(N)$ is reached.

The amount of attribute entries stored in a Coordinator / Support Peer team on level i was calculated in Eq. 3.62 in the variable $LoadCoordSP_i(N)$. Please note that for the sake of clarity, we only consider one attribute for the model, the capabilities of the peers, and omit other possible attributes, like CPU power, free memory and bandwidth capabilities. Thus, we define for modeling reasons the $QualityDistribution$ of the peers in terms of quality classes as:

$$QualityDistribution_i(N,k) = CapabilityDistribution_i(N,k) \tag{3.67}$$

In Table 11, we show the situation that a Coordinator and its Support Peer can store 125 attribute entries out of 2000 incoming attribute entries:

Capability class	16384	8192	4096	2048	1024	512	256	128	64	32	2-16
Stored attribute entries	0.12	0.24	0.49	0.98	1.95	3.91	7.81	15.63	31.25	62.5	0

Table 11: Example Table of stored Attribute Entries: 125 out of 2000

Table 11 is an example of a table that depicts both the quality and quantity of the attributes a peer offers. We use tables for each level in the monitoring tree to model the query solving. A query for k peers with a specific quality level (e.g. $q = 256$) is solved on level i, if an average Coordinator / Support Peer team at level i together stores more than k peers corresponding to the desired quality level (e.g. $q = 256$).

Following matrix calculates the query solving levels for queries for k peers with a quality of q:

$$\begin{aligned} QuerySolvLevel(N,k,q) = i \; with \; & QualityDistribution_i(N,k) > q \\ & and \; QualityDistribution_{i+1}(N,k) < q \end{aligned} \tag{3.68}$$

The number of hops needed for resolving a query for k peers with a quality of q is then the difference of the levels of the query originating and solving peer. With γ being the level of the query initiating peer, we calculate the number of hops needed for resolving a query:

$$QueryHops_\gamma(N,k,q) = max(0, \gamma - QuerySolvLevel(N,k,q)) \tag{3.69}$$

DISTRIBUTION OF QUERY RESOLVING PEERS

Up to this point, we just considered individual queries and solutions. In order to get the distribution of where queries are generated and solved for the whole monitoring tree, we need to take the peer distribution per level into account.

The distribution for query origination is equal to the peer distribution in the tree. Thus, it is independent of the query parameters:

$$\mathrm{QueryOrig}_\gamma(N,k,q) = E_\gamma(N) \tag{3.70}$$

The corresponding query origination distribution is as follows:

$$\mathrm{QueryOrigProb}_\gamma(N,k,q) = \mathrm{QueryOrig}_\gamma(N,k,q)/N \tag{3.71}$$

The number of queries solved on level i is derived from the distribution of query origination and the query resolving hops needed:

$$\mathrm{QuerySolvCount}_i(N,k,q) = \mathrm{QueryOrig}_\gamma(N,k,q) - \mathrm{QueryHops}_\gamma(N,k,q) \tag{3.72}$$

The distribution of query solving is then

$$\mathrm{QuerySolvProb}_i(N,k,q) = \mathrm{QuerySolvCount}_i(N,k,q)/N \tag{3.73}$$

Here, we must take into account that the number of originated queries differs from level to level, because it depends on the number of peers on a level. Thus, we do not count 1 for a query solved on a level i which is originated from level k, but add instead the probability that the query is originated from this level k. In summing up these probabilities and dividing the result by the number of queries solved on this level, respectively, by the number of same hop counts detected, we obtain statistics about the position of query solving peers and hop counts needed.

3.6 RELATED WORK

Various papers have addressed peer and system information monitoring for p2p networks. For the classification of related work, we follow the classification points introduced in the Sections 3.2.1, 3.4.3 and 3.5.1 on design decisions.

Centralized vs. Distributed

Network and distributed system management has been investigated thoroughly in literature in previous decades. The simple network management protocol [CFSD90] has been published as RFC in 1988 and continuously improved. It presents an approach for the centralized monitoring of network devices. Traditionally, the quality of service provided by the network and transport layer were in focus (e.g. [vBH97, KCH02]), as well as the general quality and status of nodes in the network. Since this topic is highly interesting from a commercial perspective, a wide range of hardware manufacturers established system-on-a-chip or operation system independent solutions for monitoring internal capabilities.

Intel's Active Management Technology (AMT) [DvdGH+06] features, among others, device discovery, feature tracking, system shut down and restart, and utilization metrics, even when the host's Operation System (OS) is inactive. AMT's architecture is based on the concept of a platform container. It is an OS independent computing platform that resides on a dedicated microcontroller in the chipset, or a plug-in card. Accordingly, this platform container has an isolated execution process (including processor and memory), which enables the system to operate in pre-power, pre-OS states, and is capable of assisting the OS when it is present, and taking over when it is not.

Intel's AMT technology is already broadly integrated into desktop PCs with Intel Core 2 processor and is also available in laptop PCs with Centrino processor. IBM System z10 Active Resource Monitoring (ARM) technology is targeted towards the goal of maintaining continuous operation on IBM servers. Therefore, ARM implements an integrated, automatic resource monitoring software application to track resource, performance, and operational problems. Moreover, the system initiates recovery actions in case of failures. In contrast to Intel's AMT approach, IBM's ARM focuses on the high-performance and

commercial server market, which is the basis for grid-computing and large enterprise networks. The centralized approaches are optimized for a commercial market with a small- to mid-sized network. For large-scale networks with no commercial provider, like p2p systems, they are not suitable. A monitoring solution in a p2p system must be distributed as well and hosted by all participating peers.

Integrated vs. New Layer
As a next classification step for related work, we look at overlay-integrated and overlay-independent solutions. DASIS [AAGW04] is a module extending the routing table of the used overlay to store additional routing-specific information; no further monitoring data structure is proposed. Assuming prefix-based routing, peers gossip information in order to become experts for all of their prefix-zones. DASIS strongly depends on the details of the used overlay and can only be used for small portions of information. P2P-Diet [IKT04] extends hybrid unstructured p2p overlays with the functionality of ad-hoc and continuous search for specific objects (and peers). Using a publish/subscribe approach tightly coupled to the used super-peer overlay, peers are notified when other peers of their interest appear. P2P-Diet provides network monitoring and capacity-based peer search, causing significant overhead by broadcasting information updates and extensive (minimal spanning tree) maintaining operations. Maintenance costs for the tree in SkyEye.KOM are low, as we use a deterministic function in SkyEye.KOM to identify the node positions in the tree. HilbertChord [SSN+08] is a modified Chord overlay with a two dimensional ID space that is mapped through Hilbert space filling curves onto the one dimensional ID space of Chord. The two dimensions of HilbertChord acts as key and value entries for the capabilities of the depicted peers. Lookups to a specific two-dimensional ID in the overlay are routed to a peer, providing the desired capacities. This approach is both inflexible regarding the portability to, for instance, other overlays as well as the flexibility regarding quickly changing capacity values. Another drawback lies in the fact that for every attribute to announce, a new dimension is needed in the overlay. The metering and monitoring project of JXTA [TAA+04] aimed at providing a monitoring tool for JXTA networks but has been abandoned. Astrolabe [vRBV03] was published in 2003 as a distributed (structured) monitoring solution, and although not in specific for p2p systems, many concepts of it can be adapted to the p2p scenario. In Astrolabe, nodes join several so-called zones, which correspond to the nodes' hierarchical host name. Creating a topology according to the hierarchical zones results in a tree of depth $O(\log |IDspace|)$; the tree in SkyEye.KOM is $O(\log |N|)$ deep. Information is disseminated between zones and in the zones via gossiping. Although Astrolabe fulfills the desired functionality, it comes with its own overlay and is inefficient due to the depth of its tree, which contains various empty zones. Willow [vB04] extends the idea of Astrolabe to a DHT overlay, integrating various functionality of p2p layers. The solution is more efficient, but not overlay-independent. Overlay-independent solutions allow for a wider use of the monitoring solution.

Monitoring Scope
For the monitoring of the system status we aimed with SkyEye.KOM at a global view. Authors like Cyrus Hall et al. [HC09] aim to providing access to a representative, unbiased sample set in p2p networks. These peers may be queried for their status and result in an interpolated global view. The sample of a biased set, as in [BGMGM03] picks the peers in the network with a probability related to their node degree, resulting in biased estimates. The authors of [NG09] aim at consistent monitoring data among the peers. Several approaches discuss, how to derive an unbiased sample of peers, for example with random walks [MMKG06], newscast [TJ09], gossiping [KDG03] and roaming agents [DR05]. The interpolated view of the samples is, however, rarely accurate as we have found out in our evaluations of SkyEye.KOM, while comparing the monitoring results of a Coordinator on level 2 and 3 to the results of the root. The Grid Box Hierarchy protocol [GvRB01] creates a tree structure but leaves the decision on participation in the global view to the peers. This is due to the fact, that receiving the global view in [GvRB01] induces a message overhead of $O(\log N)$ on the corresponding peer, in SkyEye.KOM the message overhead is $O(1 + \beta)$, with $\beta = 4$ for example.

Monitoring Topology
In SkyEye.KOM, we follow a tree-based approach; however, other topology types could be deployed as well. T-MAN [JMB09] is a proactive gossip-based overlay topology management system where each

peer exchanges periodically its knowledge with neighbors. Information spreads slowly in the system and is hard to update. SDIMS [YD04b] allows information aggregation and attribute-based search for peers. In contrast to SkyEye.KOM, SDIMS maps each attribute to a key in the key space and aggregates all the information about the attribute at the corresponding peer in the DHT. The SDIMS architecture is balancing the load by having multiple aggregation trees and thus distributed load on individual peers. In the key-specific aggregation tree, optimized attribute-specific update and query algorithms can be used. Although SDIMS is highly optimized for traffic efficient updating and querying of single attributes, the split of the aggregation tree also cut off the relationship between the attributes. A complex query consisting of the retrieval of multiple attribute values requires multiple steps, which causes more time and message overhead than in a solution with a combined aggregation path. In the case of monitoring the system status, a single tree is very suitable as all of the desired information is available in one place and the information is gathered and distributed efficiently and in a coordinated fashion. Loops and redundant communication are avoided.

Push vs. Pull
SOMO [ZSZ03] is a metadata overlay for the resource management in p2p DHTs. SOMO builds a tree in a top down fashion on the peers in the ID space, identifying nodes in the tree using a stateless function. In SOMO the information is pulled up towards the root of the SOMO tree, aggregated and pushed back. A peer calls the growth procedure to split the current responsible region into sub-regions and find for each of the regions a responsible peer. This is periodically triggered by the root of the tree and requires that peers being responsible for a region to have periodically look for unattached peers in their region. This results in an increased overhead, which is solved in SkyEye.KOM by a push-based approach. A prune procedure is also called periodically by the root to consolidate regions.

Reactive vs. Proactive
CONE [BVV03] is a structured over-overlay like SOMO as well. In CONE, a tree is established using the natural order of the peer IDs; peers with higher IDs are parent nodes of peers with smaller IDs. The tree is used to aggregate peer information in a reactive manner. An information update triggers various update steps up the tree. Due to the reactive behavior of CONE, the overhead generated through updates is significant. In SkyEye.KOM updates are transmitted proactively, in peer-specific intervals, leaving time for messages to arrive and to be processed in a group. Furthermore, besides information aggregation, SkyEye.KOM allows attribute-based search for peers. The authors of [LL04] and [DNF05] propose a tree-based approach with a similar monitoring topology as SkyEye.KOM, but do not discuss how it is further used, whether for proactive or reactive monitoring. Another reactive approach is presented in [IFN02]; the proposed solution discusses resource location in a grid environment using p2p techniques, which addresses a different problem statement.

3.7 CONCLUSIONS

SkyEye.KOM implements a monitoring layer for structured p2p systems by introducing an additional (unified) key space, which makes it independent from the specific p2p overlay used, while using the general KBR API for structured p2p overlays. This new ID space is recursively partitioned in ID intervals called *Domains*. For each Domain, a characteristic ID is calculated using a deterministic function that maps the Domain to a single ID in it, called *Domain Key*. The peer responsible for the Domain, termed *Coordinator* of the Domain, is identified by being responsible for the Domain Key in the DHT. The recursively partitioned Domains and the corresponding Coordinators build a tree structure. Peers identify their position in the tree based on their ID and periodically send information messages to the Coordinator one level above them in the tree. These messages, termed updates, contain both information on the individual peer capacity and aggregatable statistics information. Coordinators periodically pass the aggregated statistics and the list of peer capacities one level higher in the tree in a push-based manner. The aggregated statistics received by a Coordinator describe the statistics of the peers in the corresponding Domain. Aggregation is used to keep the size of the status information small and independent of the size of the Domain. Acknowledgment messages are used to disseminate the global statistical view retrieved from higher levels in the tree as well as maintaining the tree topology. The core

tree is also used to gather unaggregated, peer-specific attribute information which are appended in each Coordinator. Peers can send queries regarding a set of peers with specific capacities to their Coordinators which forward the query up the tree, until one Coordinator has information on the required set of peers. In order to relieve the load for supporting capacity-based peer search, Coordinators may choose more capable *Support Peers* from their Domain and dispatch a part of the update and query load to them. Having some Coordinators dispatching their demanding duties to Support Peers, results in an easy to maintain support tree with capable peers. In conclusion, a tree structure is build on top of a DHT to gather and disseminate global system statistics and to support capacity-based peer search.

4 EVALUATION OF THE MONITORING SOLUTION

Omnia aliena sunt, tempus tantum nostrum est - everything is extrinsic, only time is ours.
- Lucius Annaeus Seneca
We are just statistics, born to consume resources.
- Horace

In this chapter we evaluate SkyEye.KOM thoroughly. For the evaluation of the monitoring solution, SkyEye.KOM, we first introduce the evaluation methodology, including the evaluation metrics and goals. They describe the aspects of the solution that are investigated and the effects that are of interest for a monitoring solution. As evaluation methods we use simulations, analytical modeling and evaluation through a prototype which we deployed in a testbed. All three of these evaluation approaches have their corresponding strengths and weaknesses, which in combination give an exhaustive view of the quality of the solution.

After the introduction of the evaluation goals, methodology and metrics, we present the evaluation results of SkyEye.KOM. We discuss the creation and characteristics of the monitoring tree, before investigating the monitoring quality regarding the system- and peer-specific information according to each evaluation methodology.

We show in the evaluation that SkyEye.KOM is predictable in its behavior and fully understood. The monitoring solution comes with very low costs, precise monitoring results for the system statics, controllable peer load and efficient query resolving for the capacity-based peer search.

4.1 EVALUATION OVERVIEW

The goal of the evaluation is a systematical analysis of the behavior of SkyEye.KOM. We evaluate SkyEye.KOM with an analytical model, with large-scale simulations and with a detailed prototype deployed in a testbed. The analytical model for SkyEye.KOM is presented throughout in Chapter 3. The analytical model describes the expected characteristics of the monitoring tree, the generated costs and the query performance. Although the analytical approach helps to investigate the interdependencies between parameters and metrics in the behavior of the solution, the model does not capture the effects of dynamism in the network.

Simulation studies have been performed in order to investigate the solution in more detail. The simulator PeerfactSim.KOM [KKM+07] has been extended with the structured overlay Chord [SMK+01] and a reference DHT that uses a centralized index, to create a distributed hash table (CDHT). Both overlays comply with the KBR-interface presented in [DZD+03]. Specifically, for the main evaluation of SkyEye.KOM, we use the reference DHT, implementing the desired functionality of DHTs, in order to not bias the results by the characteristics of Chord, Kademlia or Pastry. We extended the KBR interface with the second function from the requirements section, *Boolean isMyKey(key)*. The simulator allows to use the system-wide simulator view as reference information for the monitored global view. Consequently, we can compare the performance of the monitoring solution and measure all desired metrics describing the solution. The simulations also help to investigate the effect of variation on the parameters used in the solution, such as the update interval or the branching factor.

The simulations use measurement-based models for the peer capacities and the underlying transport layer, which have been validated in the corresponding literature. However, for a complete understanding of the quality and trade-offs in the solution, the simulations need to be validated through a prototype. Thus, we implemented SkyEye.KOM prototypically and evaluated it in a testbed. The testbed results show the applicability of the approach and give deeper insights in the performance of the solution.

Through a comparison of the evaluation results obtained through the analytical model, simulations and testbed analysis, SkyEye.KOM has been thoroughly evaluated.

4.1.1 *Goal*

The goal of the evaluation is to systematically analyze the characteristics of SkyEye.KOM in relation to the desired functionality. The evaluation is threefold organized according to the evaluation methods, with each investigating the characteristics of the monitoring tree, the metrics related to the monitoring of system-specific information and the metrics regarding the monitoring of peer-specific information.

In the monitoring tree, we evaluate the following points:

- Peer positioning: what is the peer level distribution, how deep is the tree, is it formed as expected?
- Coping with churn: how robust is the tree topology under churn?
- Freshness: what is the age of the information gathered at the root and distributed to the peers?
- Cost: what are the costs in the overlay for establishing and maintaining the tree structure?

The evaluation strongly focuses on the characteristics of the tree, especially the peer level distribution. The positions of the peers in the tree have a strong influence on the eventual age of the information that reaches the root. Having a model for the tree structure that is validated through simulations, allows us to predict the behavior of the tree and the interdependencies of the parameters on the performance.

The monitoring of system-specific information aims at showing the status of the running system. Thus, the main evaluation criterion is the precision of the monitoring results. Additionally, the second essential criterion is that the monitoring solution is meant to be an extension of current p2p systems and thus very cost effective. We evaluate the following points regarding the monitoring of the system-specific information:

- Precision: what is the monitoring error in comparison to the actual status of the system?
- Costs: what is the traffic overhead to obtain the aforementioned monitoring performance?

The scope of the monitoring view for system-specific information is covered with the number of peers included in the tree, thus the tree characteristics. The evaluation focuses on the monitoring performance of selected system status metrics and the overall monitoring costs. In terms of reference metrics that are monitored, we focus on traditional metrics used for evaluation, such as the number of peers in the network, their bandwidth consumption and the messaging overhead in the overlay. These metrics cover a wide set of metric types, ranging from global values, peer-specific values and network-specific values. In addition, we introduce two reference signals: Sine and ZigZag with variable intervals. For these signals, we synchronize the peers in the simulator to induce the same signal at the same time at all peers. With this, it is easy to evaluate, how precise the information is monitored, whether it is distorted and how fresh the monitored information is. An overview on the set of metrics monitored in the p2p system is outlined in Table 2, 3 and 8.

The monitoring of peer-specific information aims to prepare the information about individual peer capacities, in order to provide capacity-based peer search on them. The main criterion is to gather the information of as many peers as possible, without overloading the Coordinators and Support Peers in the monitoring tree. Furthermore, this information must be fresh and the peer capacities still available when used in a query response. One main evaluation aspect of the capacity-based peer search is therefore the quality and quantity of the peer attributes that are monitored and usable for the capacity-based peer search. As a second step, the query protocol is deeply investigated and the query originator and resolver positions in the tree are analyzed. The effects of varying complexities of queries is investigated and the query performance and response quality are evaluated. The costs for the gathering and querying of peer-specific information are measured and the effects of the Support Peers are shown. The following questions are addressed:

- Monitoring scope: how many peers are monitored, what is the quality of information?
- Query protocol: where and how fast are queries resolved depending on their complexity?
- Costs: what is the traffic overhead on the Coordinators and Support Peers?

In the following subsection, we introduce the metrics used to evaluate the monitoring solution SkyEye.KOM.

4.1.2 *Metrics*

In order to evaluate the solution in terms of tree characteristics, monitoring performance and costs of both, system-specific and peer-specific information, we use the monitored metrics presented in Tables 2, 3 and 8. Please note, these metrics are both measured by the evaluation environment and used by the monitoring solution to capture the status of the p2p system. In the tables, we present the metrics used, give a short description and also note by which evaluation method it has been measured. As an evaluation method we used simulations (s), an analytical model (a) and a prototype (p).

In Table 2, we list the metrics regarding the monitoring topology. The metrics depict the structure and behavior of the monitoring tree under various numbers and behaviors of peers in the network. The number of the various peer types in the tree topology and the characteristics of the tree itself have a high influence on the monitoring performance.

In Table 2, in addition, the metrics regarding the traffic-related costs are listed. The number of messages sent and received, the message sizes and the corresponding bandwidth consumption are listed according to the various message types. The following message types are considered:

- Overlay messages: lookup, join/leave, other (maintenance)
- System-monitoring messages: metric update, metric ACK
- Peer-monitoring messages: attribute update, attribute ACK
- Capacity-based peer search messages: attribute query, attribute response

The traffic costs are differentiated according to the message types, thus helping to investigate the costs for the individual operations. We only consider traffic costs, computational costs for aggregation and query calculations are not essential and can be solved in O(1) and O(log n), correspondingly.

The performance of the system monitoring solution is measured in terms of the closeness of the monitored metric values in comparison to the actual metric values. Table 3 depicts the main metrics used for evaluation of SkyEye.KOM regarding the monitoring of system-specific information, namely the reference signals Sine and ZigZag.

In Table 8, we present the metrics used to evaluate the monitoring capabilities of SkyEye.KOM regarding peer-specific information. First, the capacities of the Coordinators and Support-Peers are analyzed and the monitoring results regarding some peer resources are shown. Based on the capacities of the Coordinators and Support Peers, the completeness of the monitoring view is investigated. The scope of this view describes how many peers of the corresponding Domain are covered. Lastly, queries are performed on this knowledge set as well as the query processing and query resolving characteristics are elaborated.

The metrics presented in the evaluation describe the characteristics and behavior of the monitoring solution in total. The performance and cost-related metrics are also suitable for comparing SkyEye.KOM to other monitoring solutions. Having introduced the metrics for evaluation, the evaluation techniques and setups are subsequently presented.

4.2 EVALUATION TECHNIQUES AND SETUP

In this section, we briefly introduce the evaluation techniques and implementation details, as well as presenting the various aspects of the evaluation setup and the varied parameters. We evaluated SkyEye.KOM through three methods: an analytical model, large-scale simulations and a prototype implementation in a testbed. All methods come with strengths and limitations in their evaluation statements. However, through a combined and comparative view, the characteristics and behavior of SkyEye.KOM are thoroughly analyzed.

4.2.1 *Analytical Model Details*

The goal of the evaluation through the analytical model is to get a deep understanding in the parameters influencing the performance of SkyEye.KOM. The analytical model describes theoretically the

interdependencies between the parameters in the systems and the metrics. Here, we stick to the metrics investigated in the simulative and prototypical evaluation as a reference. We describe the characteristics of the monitoring tree, followed by the performance of the monitoring solution in terms of information age and monitoring scope. Additionally, we calculate the costs in terms of traffic generated and the load distribution in the network.

The main limitation is that the modeled view on the monitoring tree is static. All actions performed in the monitoring tree assume that no peers join or fail in the meantime and that all the information in the network is consistent. Churn in the network is modeled in the fact, that for every peer count the expected monitoring tree can be modeled. The limitations of the solely theoretical approach, the following limitations have to be considered. The number of peers in the network define the expected tree. In this tree, costs and performance are modeled. The information in the peers is considered to be consistent. No message retransmissions are needed, resulting in a slightly lower traffic load. No violation of timeout intervals for Sub-Coordinators is modeled, resulting in a slightly lower information age. The underlying overlay and its routing behavior are not modeled; it is assumed to provide the KBR-functionality. The monitoring values are not smoothed. Lastly, the model assumes that the peer IDs are evenly distributed.

The parameters of SkyEye.KOM in the analytical model are also functional parameters, which allows their effects to be investigated in more detail. Regarding the evaluation setup, an analytical model does not comprise a time line of events. We describe the characteristics of the monitoring tree, which depend on the number of peers in the network and the branching factor of the tree. Based on this reference tree, we evaluate the performance of the solution and the costs.

4.2.2 *Simulation Details*

The goal of the simulative evaluation is the analysis of the solution in a large-scale scenario with several thousands of peers. This scenario cannot be done in a testbed due to the limited capacities of the testbeds of today's research community. The largest testbed, the PlanetLab [Pla] consists of 1085 nodes, out of which only a subset is typically online, providing a platform for a wide set of concurring applications. A simulator allows for the simulation of several tens of thousands of nodes and thus the capturing of the effects of large networks.

We used PeerfactSim.KOM [KKM+07] as the simulator, which has been developed at the Multimedia Communications Lab at the Technische Univeristät Darmstadt. The simulator is used in the department and within several national and European projects. Various publications at prestigious p2p conferences have used PeerfactSim.KOM to evaluate their results, for example, in [KLS07], [KLKP08] and [GSR+09]. The Java-based simulator is event-based, consisting of several functional layers. The event-based nature of the simulator ensures that events are processed sequentially and may initiate new events in the future.

Functional Layers in PeerfactSim.KOM
The functional layers of PeerfactSim.KOM are depicted in Figure 23 and described in the following:

- User: Defining the strategies and actions performed on the application layer
- Application: Defines the application and its characteristics, such as file sharing
- Service: Monitoring, management and further service functions
- P2P overlay: Structured overlays providing a KBR interface
- Transport: Defines the bandwidth management, according to UDP or TCP
- Network: Defines the delay for peer to peer transmissions and peer churn
- Simulation event queue: managing and scheduling events in the simulation

SkyEye.KOM has been implemented in PeerfactSim.KOM in the service layer and adapted to operate on the KBR-interface. As KBR-compatible structured overlays, we used Chord [SMK+01] and a centralized DHT (CDHT). Both overlays are depicted in Figure 24, they provide the KBR interface

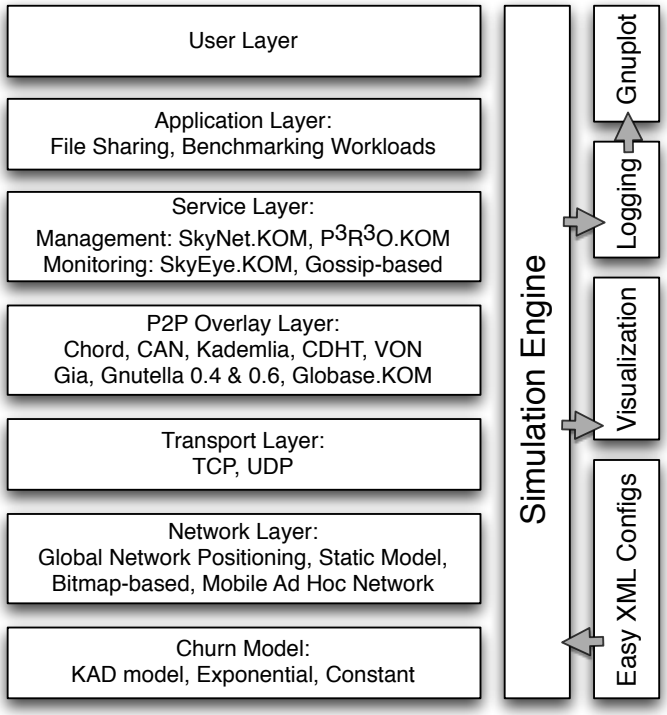


Figure 23: Overview on the Functional Layers of PeerfactSim.KOM

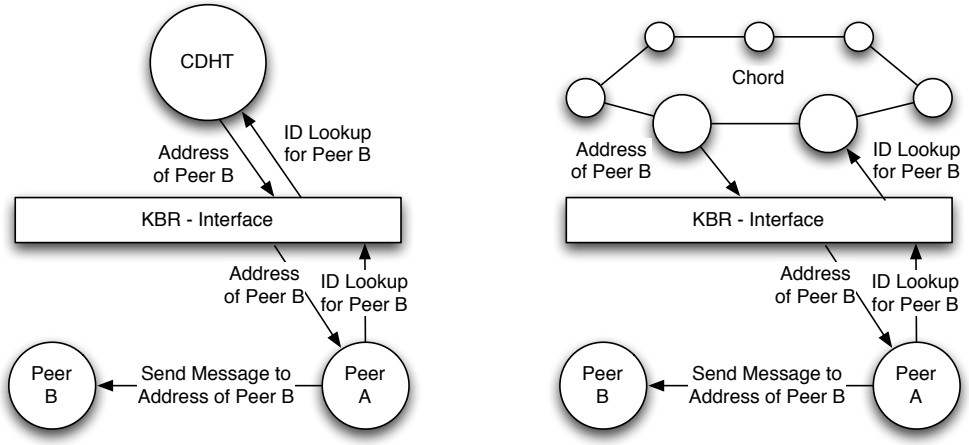
which is required by SkyEye.KOM. For a fully operational Chord implementation, we used the protocol description in [SMLN+03] as a reference. The centralized DHT (CDHT) is similar to the Napster topology and has a server connecting all peers. Instead of storing ID tags of shared music files, the server stores the peer IDs and responsibility ranges of the connected peers. Thus, a DHT is implemented with a centralized index. This structured overlay implements an ideal DHT, as all messages are routed successfully and any query regarding the responsibility ranges of peers are resolved correctly. The overlay acts as a reference overlay in our evaluation to distinguish the effects of the overlay from the monitoring solution.

In the application layer, no application was used, in order to focus on the evaluation of SkyEye.KOM. The transport layer implements UDP [Pos80] for small messages and TCP [KC81] for large messages. Message loss is assumed and implemented, but is also addressed with TCP and application level retransmissions. The impact of the message loss is negligible.

The network model used in the simulations is important for the evaluation results. It is both responsible for the churn behavior in the network and the delay for message transmissions in the simulated network. The churn behavior of the peers was deducted from KAD measurements as published in [SENB07]. They have been obtained through measurements done on the KAD network over seven weeks, capturing the online behavior of the peers. As a delay model, we used Global Network Positioning, as described in [NZ02]. Through measurements including 150,000 nodes in the Internet, a multi-dimensional map has been created, that allows for simple calculations of delays between two peers. Using the measurement-based churn and delay model for the network layer creates, the basis for validated simulation results.

Simulation Setup and Time Line

Next, we describe the setup for the simulations and the time line of events. The list of the parameters and their settings for the simulations are as follows. In the simulation, we use for the branching factor β the values 2,4,8 and 16, as metric update interval we use 15s, 30s, 1m, 2m and 4m and regarding the attribute update interval, we chose 3 minutes. In the simulations the network is scaled to 10,000 peers in order to investigate the scalability behavior of the solution.

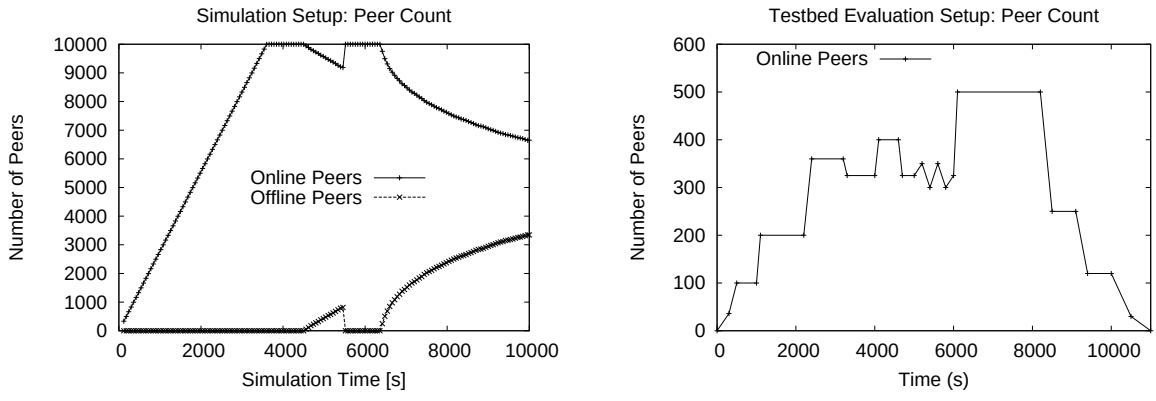


(a) KBR-Lookup in Ideal DHT (b) KBR-Lookup in Chord

Figure 24: Lookups in KBR-compliant Structured P2P Overlays

Due to the controlled environment, the effects of all parameters can be analyzed. We vary the network size, the churn pattern, the branching factor β of the tree, as well as the update interval, UI. As the simulations describe a bridge from testbed evaluations to analytical models, we also simulate with the settings of the model and the prototype testbed. With this, the results of all three evaluation methods can be compared.

The time line of events is determined by the churn behavior of the peers. The main churn model is based on KAD measurements, as described in [SENB07]. We evaluated our approach both with 1000 and 10000 peers over 10000 seconds. The time line of peer count for the setup with 10000 peers is depicted in Figure 25a, the same shape is valid for the network with 1000 peers. In the first hour (until $t = 3600s$) the peers join, and then, after a stabilization phase, we induce from $t = 4500 - 5700s$ exponential churn resulting in 10 % of the peers leaving the system, after the churn phase, the number of peers recovers to 10000 and stabilizes. At $t = 6500s$, we induce KAD churn based on measurements in the KAD network, which were presented in [SENB07]. The number of peers drops in the next 3500s to 70% of the initial network size.



(a) KBR-Lookup in Ideal DHT (b) KBR-Lookup in Chord

Figure 25: Time Line of Peer Count in Evaluation Setups

4.2.3 *Testbed Details*

The goal of the testbed evaluation is the validation of the simulation and analytical results as well as a deeper insight into the costs occurring at the level of the peers. Through local measurements, the CPU, memory and bandwidth consumption can be investigated. In addition, we observe the impact of churn in a real deployment.

Functional Layers in the Prototype
The prototype consists of two functional layers: SkyEye.KOM and the structured p2p overlay FreePastry [Ric]. FreePastry, along with OpenChord [LK06], BambooDHT [Sea04] and JXTA [TAA+04], is one of the open source overlay implementations available. It implements Pastry, as described in [RD01], and is widely used in the research community. FreePastry has been chosen as an overlay for the prototype, as it provides the KBR-functionality and no further changes were necessary.

On top of FreePastry, SkyEye.KOM was implemented with all modules described in Figure 11 in Section 3.2. Additionally, a logging component was added, which stores all received metric and attribute updates in a file. Having a log of the monitoring information per peer, means testbed results can be evaluated in a post processing step. In order to have meaningful information, additional sensor were added to measure the hardware utilization and to count the traffic load.

Testbed Setup and Time Line
The testbed for the evaluation of the prototype consists of 37 modern PCs with a dual-core CPU and at least 2GB of RAM. On these PCs, two types of setups were combined. In the prototype, we use for the branching factor β the value 2, as metric and attribute update interval we use 5s. First, every PC had a single instance of SkyEye.KOM and FreePastry running, emulating a small p2p network. As a second step, several instances of the prototype were started on every single PC, resulting in up to 500 instances on 37 PCs. With this, we evaluated the behavior of the prototype in a mid-sized p2p network.

The time line of peer count for the testbed setup is depicted in Figure 25b. We imitated the exponential churn as well as drastic peer joins and failures. The peer fail ratio was tested for 10%, 20% and 50%. No additional workload was placed on the nodes, as no further application was implemented. However, by omitting application-specific workloads, we obtained the clear costs for the monitoring solution SkyEye.KOM.

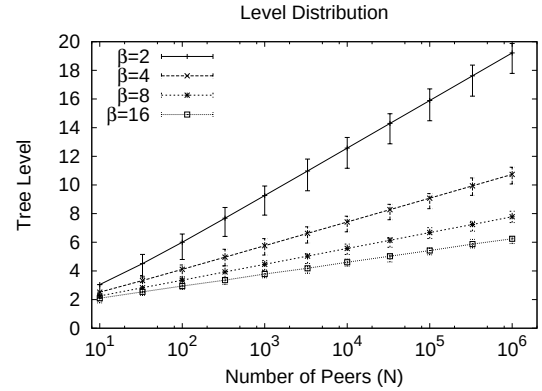
In the previous subsections, we have introduced the evaluation methodologies used and presented the setup of the analytical model, the simulations and the testbed. Having seen the metrics before, next, we present the analytical model and the evaluation results regarding the monitoring topology. The tree topology is essential for both, the performance of the solution as well as the traffic generated per peer and in total. The evaluation of the monitoring protocol for system-specific information and for peer-specific information follow after the discussion of the evaluation results regarding the monitoring topology.

4.3 ANALYTICAL EVALUATION RESULTS

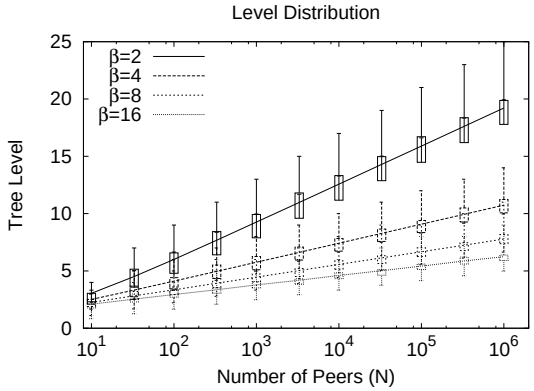
In Chapter 3, we gave a model for SkyEye.KOM leading to a basic understanding for the establishment of the monitoring tree, next, we present the evaluation results of the model. The chapter focuses only on the analytical results and present the characteristics of the monitoring tree, the expected monitoring freshness with regard of system monitoring and present a parameter study on the expected quality of the capacity-based peer search. Through this evaluation, we gain an understanding of the implications of the parameter choices in SkyEye.KOM. The section closes with an analytical comparison of our distributed monitoring approach with an centralized monitoring approach. We point of the limitation of the centralized approach in terms of scalability.

4.3.1 *Results on the Peer Distribution*

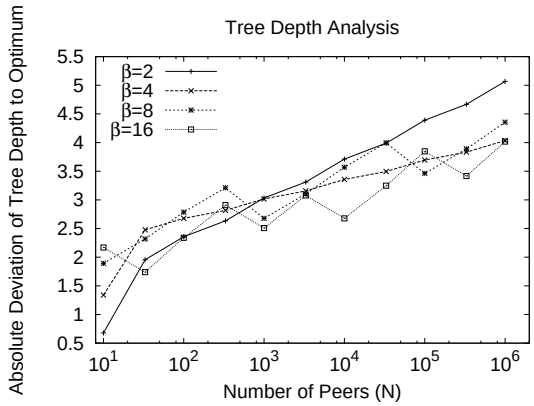
Next, we focus on the characteristics of the monitoring tree, as it has a great influence on the monitoring quality and induced costs. We depict the expected peer distribution (Figure 26a), the tree depth and



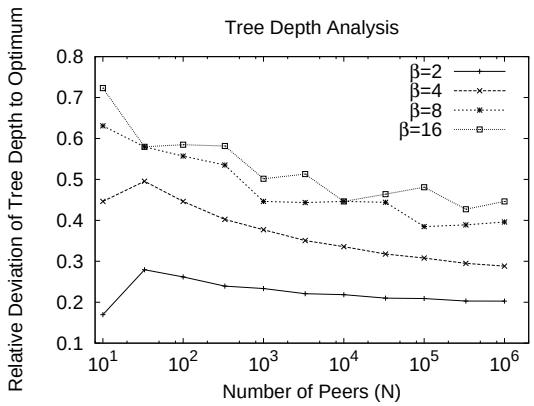
(a) Peer Level Distribution with varying N and β



(b) Tree Depth, Optimum, and Level Distribution



(c) Absolute Deviation to Optimum



(d) Relative Deviation to Optimum

Figure 26: Tree Depth and Deviation from Optimum

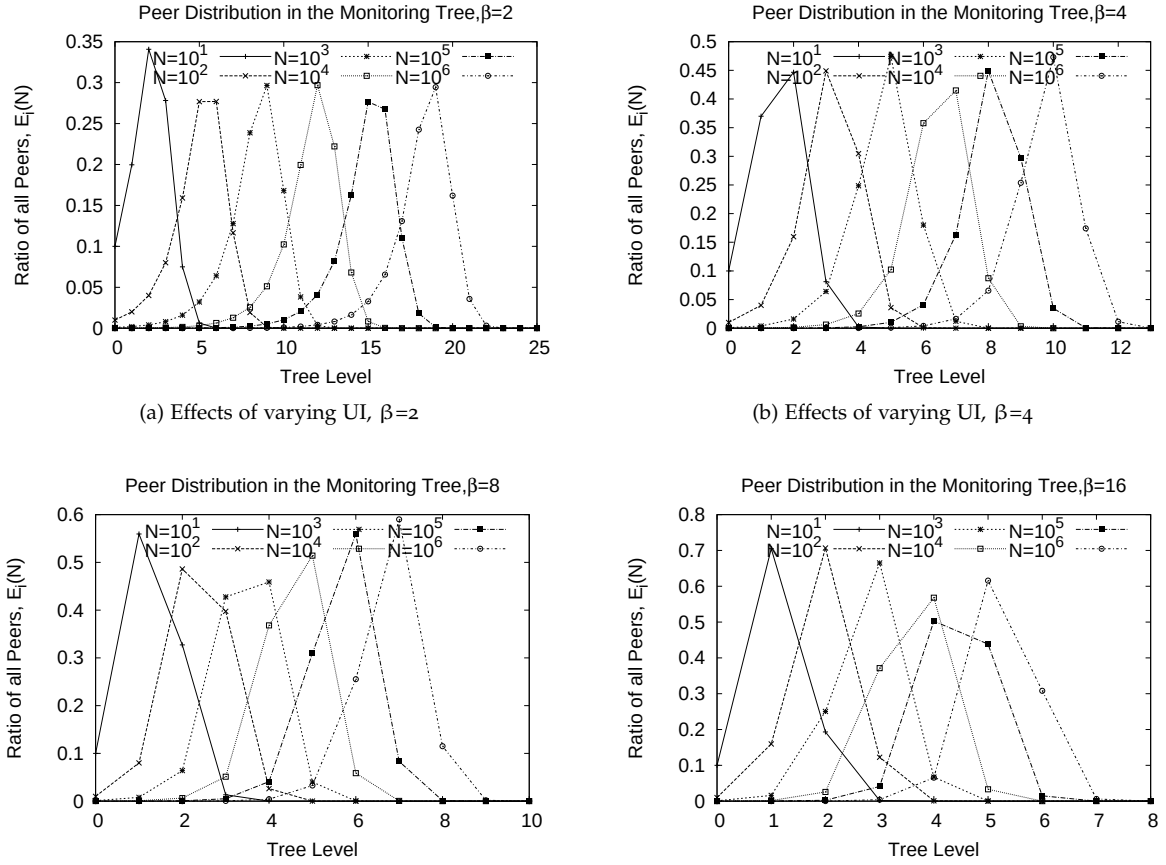
peer distribution (Figure 26b) as well as the absolute and relative deviation from the optimal tree depth (Figures 26c and 26d).

Figure 26a depicts the number of peers on the individual levels of the tree with varying number of peers. The number of peers is parametrized exponentially, in order to investigate the scalability of the solution. The figure shows that the tree scales logarithmically with a basis matching the branching factor β. The narrow standard deviation shows that most of the peers are located only on a few levels, thus, having a similar destination to the root.

Regarding the tree depth and its deviation from the optimum tree depth (i.e. $\log_\beta(N)$), the figure shows that the tree grows logarithmically and that the tree depth is close to the optimum. The imbalance of the tree that causes deviation from the optimum is a result of the non-optimal ID distribution of the peers. However, the deviation is small and the tree is not expected to degenerate.

Besides the tree depth, we have a closer look at the analytical results on the peer distribution. Figure 27 depicts the ratio of expected peers on the individual levels for varying number of nodes in the p2p network. The figure shows that the tree grows logarithmically and has an exponential distribution of peers at the levels in the upper part of the tree. The peers are close together and have similar levels, as shown through the small standard deviation in Figure 26a. With an increasing number of nodes in the network, the tree depth and peer distribution are predictable and of low tree height with an increased β.

In Figure 28, we depict the number of expected peers per level and the number of free place on the level; both add up to the maximum available peer positions in the level. For each network size, five graphs are depicted. The largest shows the potential peer positions in the tree that could be filled with nodes. The second largest shows the free places in the tree, places that may be set by new arriving peers. The third largest graph depicts the actual expected number of peers in the tree on the corresponding



(a) Effects of varying UI, β=2

(b) Effects of varying UI, β=4

(c) Effects of varying UI, β=8

(d) Effects of varying UI, β=16

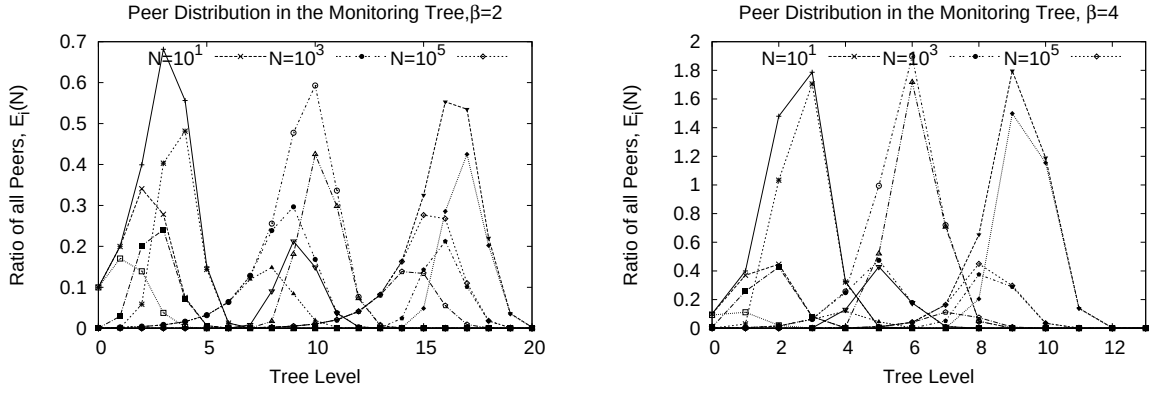
Figure 27: Effects of Parameter UI on the Peer Distribution

levels. This number is a combination of peers that have free places among the Sub-Coordinators and those that do not. These two numbers are depicted by the fourth and fifth graph.

In Figure 29, we depict the peer distribution in the tree with varying network sizes. For each branching factor β, we show the ratio of the total number of peers per level as well as the ratio of peers with and without free places among their Sub-Coordinators. The branching factor $\beta = 2$ results in a very deep tree, while the other branching factors, $\beta = 4, 8, 16$, result in trees of small height that are close to each other. As a conclusion to this observation, we state that the branching factor $\beta = 2$ should be avoided and a larger β should be used. Here, a trade-off has to be identified, as with increasing β the average peer load increases linearly while information age at the root decreases only logarithmically.

4.3.2 *Results on the Freshness of the Monitoring View*

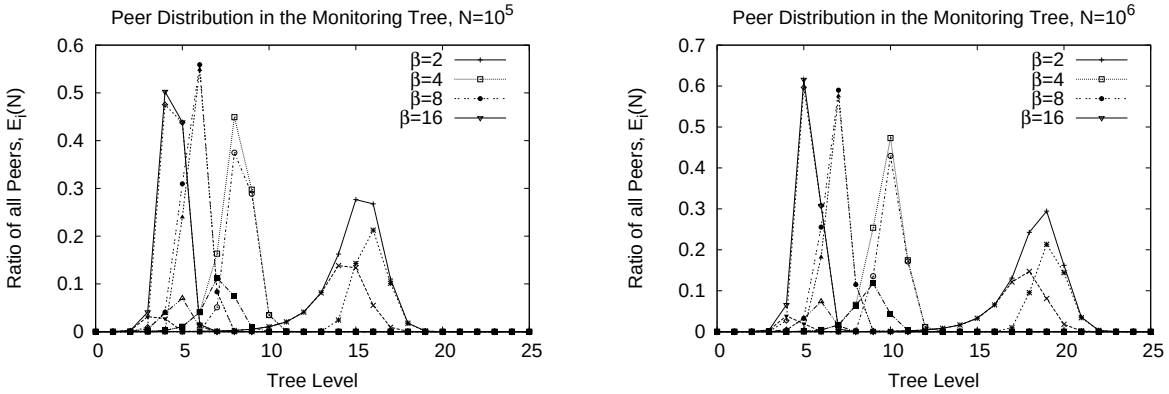
The freshness of the information in the monitoring tree is an essential indicator for the quality of the monitored results. SkyEye.KOM does not estimate or guess the accurate status of the system, but gathers and aggregates it, based on older local observations. Thus the age of these information, combined to a global view, is one main metric regarding the performance and quality of the solution. Regarding the age of the information that is gathered and aggregated at the root of the SkyEye.KOM tree, we investigate the following the effects of the parameter choices. In Figure 30 and Figure 31, we depict the average age of the information retrieved by the root under varying β and update interval UI. As anticipated, the average information age becomes smaller with higher branching factors. Varying the branching factor β determines the height of the tree, as shown in Figure 26b. Equation 3.40 explains that the height of the tree is linearly linked to the age of the retrieved information. The correlation can



(a) Detailed Peer Distribution with $\beta = 2$

(b) Detailed Peer Distribution with $\beta = 4$

Figure 28: View on Peers, Non-leafs, Leafs, free Places and potential Places in the Tree



(a) Peer Distribution $N = 10^5$, varying β

(b) Peer Distribution $N = 10^6$, varying β

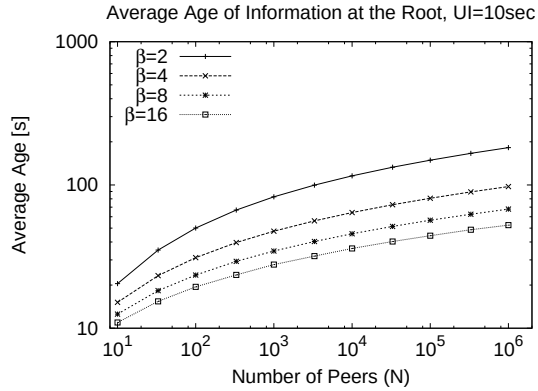
Figure 29: Effect of Parameter β on the Peer Distribution

be observed for the update intervals $UI = 15s, 30s, 60s, 120s$ in the Figures 30a, 30b, 30c and 30d. We can calculate the expected age of the information based on the given parameters and network size. Figures 31a, 31b and 31c show the effect of varying update intervals for different branching factors.

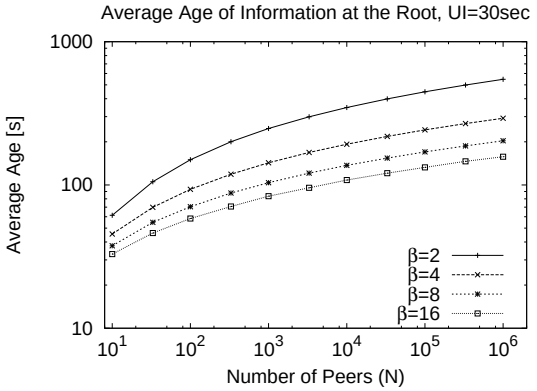
For example, with $UI = 60s$ and $\beta = 4$, we expect an information freshness of about 200 seconds. We observe that the update interval is proportional to the information age as well as the logarithmic influence of the branching factor on information age. In Figure 31d, we depict the effects of varying the ratio of the parameters UI and β. Keeping the ratio $UI/\log(\beta)$ constant results in a predictable behavior of the information age.

In order to analyze the expected message overhead, we modeled and simulated a p2p network with 5000 nodes. The resulting peer distribution is depicted in Figure 32a. Regarding the load balancing, as depicted in Figure 32b, the load is unbalanced with higher branching factors, but at a very low scale. At branching factor 8, the number of sent messages ranges between 1 and 9 with only 11% of peers sending more than 6 messages, whereas at branching factor 2 the load is evenly distributed. Every peer sends at minimum 1 and at maximum 3 messages. We determined furthermore that the amount of messages sent by the majority of peers in all graphs is approximately 1.5. However, the number of sent messages is very low considering that every update message is smaller than a typical packet size of 3KB.

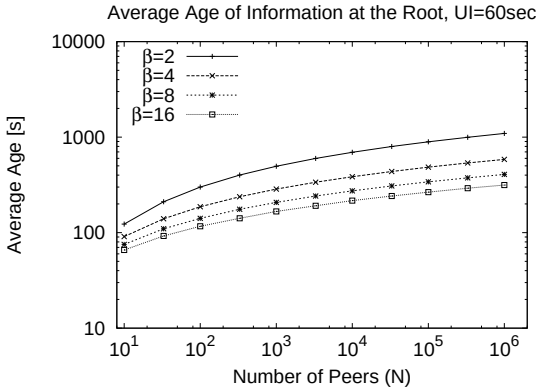
To conclude, the branching factor of 4 or 8 is desirable, as it keeps the tree height small but also keeps the load on the peers small. A branching factor of 2 results in a deep tree with significantly older information at the root, while saving only 2 messages per peer and update interval. A branching factor



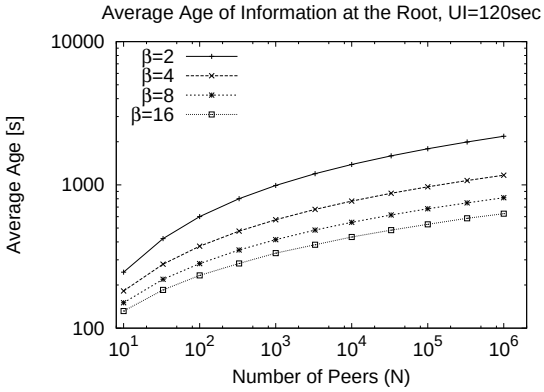
(a) Effects of varying β, UI=10s



(b) Effects of varying β, UI=30s



(c) Effects of varying β, UI=60s



(d) Effects of varying β, UI=120s

Figure 30: Effects of Parameter β on the Average Information Age

of 16 or even 32 puts more load on the peers, while not resulting in much lower trees than a branching factor of 4 or 8.

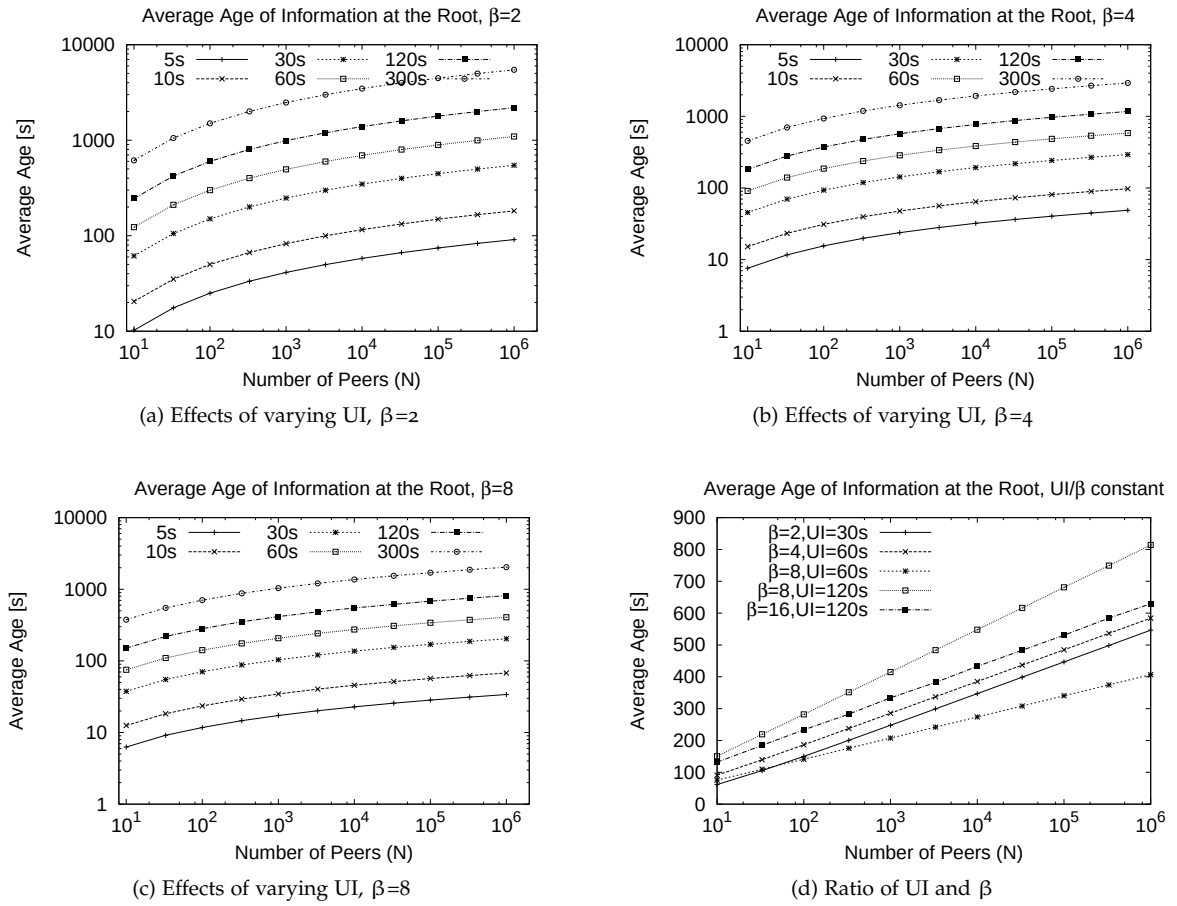
4.3.3 *Results on Monitoring and Querying Peer Capacities*

In the following, we evaluate the characteristics of resolving queries related to the function of capacity-based peer search in the model. In order to do so, we first introduce our model for the peer capacities. In Section 3.5.4, we stated the assumption that the distribution of capabilities of peers is log-distributed. In Table 12, we give an example on the basis of the quality and capacity distribution.

Quality/ capacity class	2	4	8	16	32	64	128
Number of Peers	16000	8000	4000	2000	1000	500	250
Ratio [%]	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078
Quality/ capacity class	256	512	1024	2048	4096	8192	16384
Number of Peers	125	62.5	31.25	15.63	7.81	3.91	1.95
Ratio [%]	0.0078	0.0039	0.0020	0.0010	0.0005	0.0002	0.0001

Table 12: Quality Class Distribution with N = 32000

In this case, the average quality/capacity is equal to 28, which can be obtained by multiplying the number of attributes that can be handled in a quality class with the corresponding number of peers and than averaging the result by the number of peers. The second row in the table contains the number of



(a) Effects of varying UI, β=2

(b) Effects of varying UI, β=4

(c) Effects of varying UI, β=8

(d) Ratio of UI and β

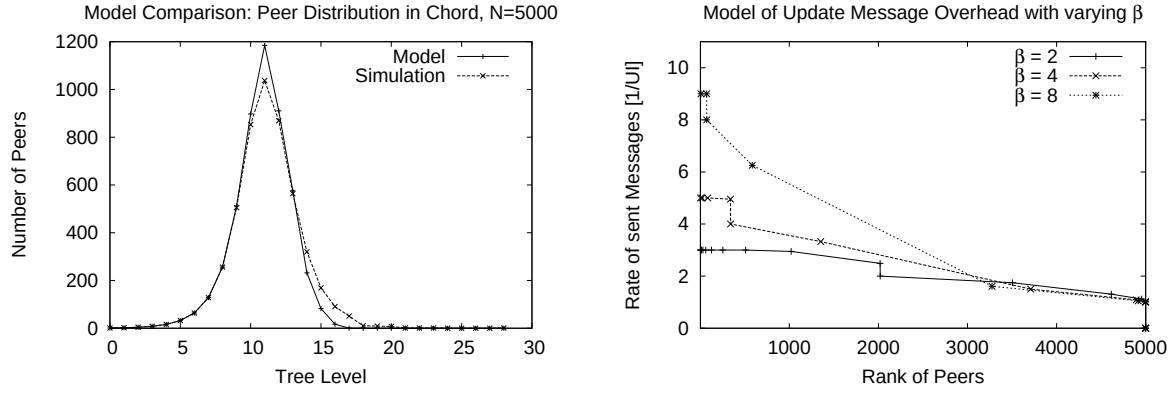
Figure 31: Effects of Parameter UI on the Average Information Age

peers in each class, relating to the entire number of peers in the network. The same distribution can also be applied to the pool mentioned above, respectively the attribute entries a peer possesses. If a peer on level i knows about 2000 peers, we obtain a quality/capacity distribution that is presented in Table 13.

Quality/ capacity class	2	4	8	16	32	64	128
Corresponding number of Peers in the Pool	1000	500	250	125	62.5	31.25	15.63
Ratio [%]	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078
Quality/ capacity class	256	512	1024	2048	4096	8192	16384
Corresponding number of Peers in the Pool	7.81	3.91	1.95	0.98	0.49	0.24	0.12
Ratio [%]	0.0078	0.0039	0.0020	0.0010	0.0005	0.0002	0.0001

Table 13: Quality Class, related Quantity and Ratio of Peers in a Pool of 2000 Peers

We interpret the results of Table 13 as follows. A peer on level i with a pool containing 2000 attribute entries knows about 1.95 peers that can handle 2048 attribute entries. The classes with better quality/capacity such as 16384, 8192 and 4096 are ignored because the number of peers known to be in these classes by this peer is smaller than 1. Thus, the best possible quality/capacity class from which this peer can appoint a Support Peer is 2048. If there are several peers on the same level i requiring a Support Peer, the number of peers represented in class 2048 may not cover the need. In this case, we take the remaining part of the Support Peers from the next best quality/capacity class, namely from class 1024. In this example, the peer and its Support Peer decide to set a load threshold to only store 200 attribute entries out of 2000. The content in the storage of those peers is depicted in Table 14.



(a) Peer Distribution

(b) Message Overhead

Figure 32: Model and Simulation of SkyEye.KOM's Message Overhead, $\beta = 2$

Quality/ capacity class	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
Stored attribute entries	0	0	0	0	62.5	31.25	15.63	7.81	3.91	1.95	0.98	0.49	0.24	0.12

Table 14: Number of Attribute Entries actually Stored

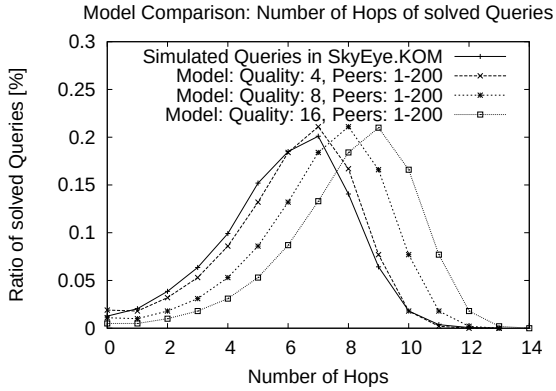
We used this quality/capacity distribution to model the queries for the capacity-based peer search. A query in our model is solved on level i if the stored information fulfills the criteria searched for. We model the level where a query is solved and how many hops are required to reach the level. The Figures 33, 34 and 35 show a comparison of the model with a simulation run with 5000 peers for validation. In the simulation, the number of peers searched for is fixed to 10 and the quality/capacity is varied between 2 and 32.

We conducted the calculations on our model with two types of queries, varying the number of searched peers and varying the query complexity (i.e. the quality class searched for). In Figure 33a and 34a, we consider the variation of the number of peers queried and depict the distribution of the number of hops for resolving the queries as well as the position of the query resolving peers. Regarding the variation of the query complexity, we depict the distribution of hops and levels of query resolving peers in Figure 33b and 34b.

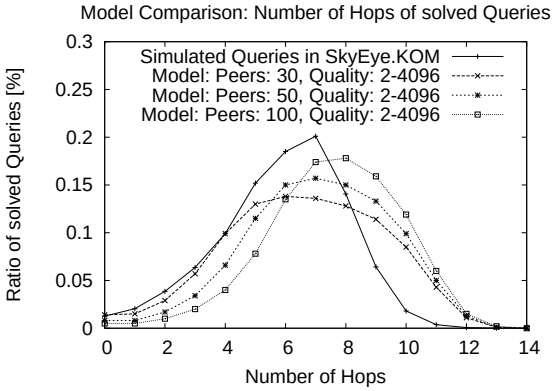
In Figure 34a, one can observe the close similarities of the shapes of the curves from the model and from the simulation. In particular, the curve from the model with required quality of 4 shows an apparent resemblance to the simulated results from SkyEye.KOM. At a level smaller than 6, about 80.6% of queries in SkyEye.KOM are solved, whereas the corresponding value in the model amounts 81.2%. In both cases, the level at which most of the queries are solved is level 4. Furthermore, the figure highlights the shift in query solving levels to higher regions with increasing complexity of the queries. The majority of queries in this setup with 5000 peers are solved at level 4 when the required quality is 4, when the required quality is equal to 8 and 16, nearly 60% of the queries are solved at level 1 to 3 respectively at levels 1 and 2.

In Figure 34b, we observe divergences between the model results and simulation results. This results from the distribution of quality/capacity classes. We chose a $\log_2$ distribution, which, for example, allocated 5000 peers to 12 quality/capacity classes. The peer distribution limits the possible ratio of solvable queries as there is not a sufficient number of peers on the corresponding levels. Thus, it defines an upper bound for the ratio of solvable queries.

Concerning the required number of hops of a solved query, Figure 33a and 33b display the distributions for the two types of queries on the model versus the simulation results of SkyEye.KOM. For both types, we see that most of the queries in the model require between 6 to 10 hops to be solved, depending on the complexity of the query, whereas in the simulation many queries are solved after seven hops. On the whole, we state that the graphs show a close similarity to the simulation results and the model is validated.

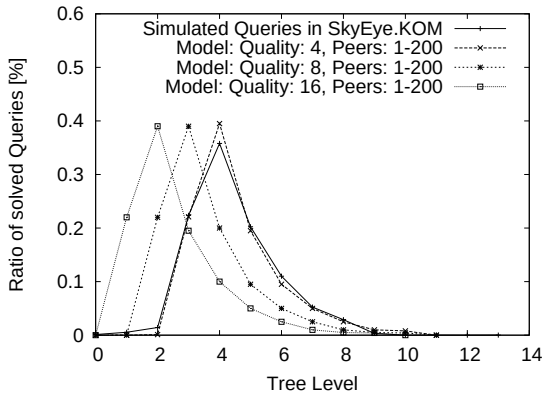


(a) Number of hops of queries with fixed quality and variable number of peers

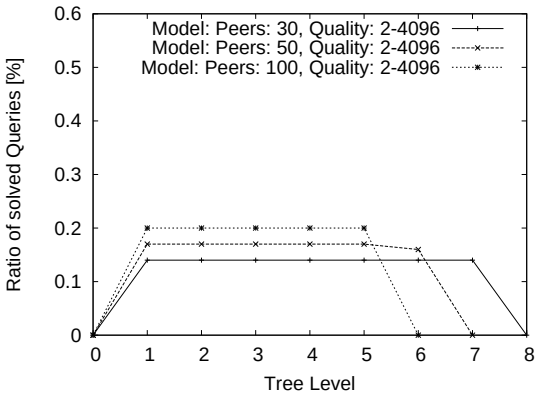


(b) Number of hops of queries with fixed number of peers and variable quality

Figure 33: Number of Hops for Query Solving in Capacity-based Peer Search



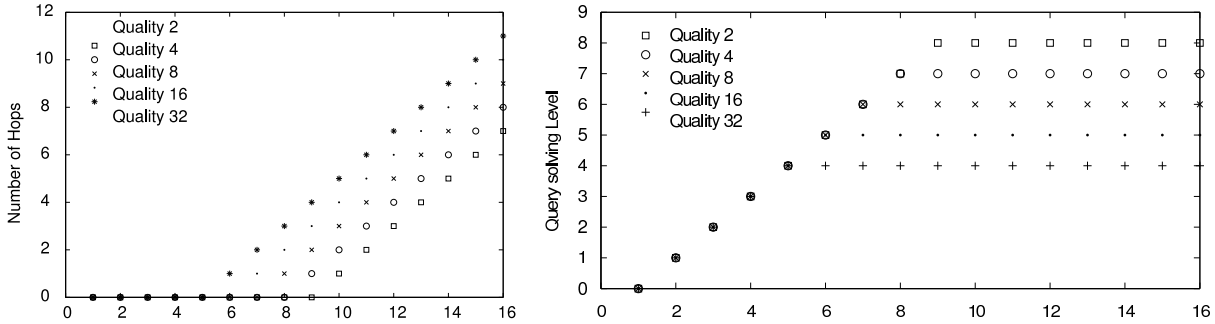
(a) Position of query solving with fixed quality and variable number of peers



(b) Position of query solving with fixed number of peers and variable quality

Figure 34: Position of Query Solving in Capacity-based Peer Search

Regarding the hop count for query resolving, we must take into account that the number of originated queries differs from level to level, because it depends on the number of peers on a level. Thus, we do not count one for a query solved on a level i which originated from level k, but use the probability of query origination on level k. In summing up these probabilities and dividing the result by the number of queries solved on this level, respectively, by the number of hop counts, we obtain statistics about the position of query solving and hop counts. Figure 35a holds the information about the number of hops required for a query to be solved, depending on the originating level of the query. Figure 35b contains the positions of the peers solving a query with regard to the complexity of the query depending on the level of query origination. To conclude our observations on the model of capacity-based peer search, we obtained a model to describe the location of query initiation and resolving based on the complexity of the query, the number of peers in the network as well as the capacity distribution among the peers. With this deep understanding, we are able to predict the behavior and characteristics of SkyEye.KOM with varying influence factors. Next, we present a general comparison on the costs related to a distributed and a centralized monitoring approach for p2p systems.



(a) Distribution of Hops for Query Resolving

(b) Distribution of Query Resolver Levels

Figure 35: Query Hop and Resolver Distribution in relation to Level of Query Initiator

4.3.4 *Modeling and Comparison to a Centralized Monitoring Approach*

In order to depict the distributed nature of the solution SkyEye.KOM and to put the evaluation results in context, we also present a brief model of a centralized monitoring approach. A centralized solution uses a server connected to all nodes that receives periodic update messages, aggregates them and disseminates the results as an ACK. Thus, the information age is optimal, while stressing the server with high load. Next, we define equations for the centralized topology regarding to freshness of information, the number of messages and traffic. We use the notation from the previous sections.

Based on the assumption that peers send messages of the same size in every update interval as in the distributed solution, the information sent by any peer reaches the server within one update interval. The information of the server itself is instantly available. The average age of messages at the server equals at maximum the length of the update interval (UI), as all the peers communicate directly with the server.

The number of messages in total only depends on the number of participating peers. Due to the particular structure of this topology, the situation at the server and other peers is different.

As the total number of messages or the entire traffic in the network only depends on the number of participating peers, we focus on the average age of messages and the average number of messages sent. We observe that the centralized architecture outperforms the tree topology regarding the age of the information and the precision. The average age of messages in the case of centralized architecture equals UI, thus $O(UI)$. In the tree topology, the age of the information grows logarithmically, with N and β as the basis; thus $O(UI \cdot \log_\beta(N))$. However, this precision comes with very high costs at the server of magnitude $O(N)$, whereas in the tree topology, the load on every peer is $O(\beta)$. The load in the tree topology is independent of N and thus the distributed solution is scalable, whereas the centralized approach is not. Regarding the load on the peers, β is a fixed value for a monitoring tree and a value of 4 or 8 is both resulting in only 3 or 7 more small messages per UI per peer in comparison to the load on the peers/clients in a centralized approach.

In the following Table 15, the discussed formulae of the monitoring tree and the centralized architecture are compared. UI stands for the update interval, N is the number of nodes, i the tree level, $E_i(N)$ the expected number of nodes on level i in a tree with N nodes and $size_m$ the size of a metric update or an ACK. For an example comparison, we choose $\beta = 8$, $N = 10^6$, $UI = 60s$ and assume a message size of $size_m = 3kb$. With the chosen parameters, the age of the information is about 6 times higher and the load is 9 times higher, but still under $1kb/s$ per peer. In the centralized approach, the server faces severe traffic (and computational) load and is a single point of failure. The comparison in Table 15 shows that the monitoring of a large scale network is possible in totally decentralized fashion with similar performance and costs for the peers as in a centralized approach.

Category		Tree	Centralized	
			Server	Clients
Average Age		$(\sum_{i=1}^{N} E_i(N) \cdot i \cdot UI)/N$	UI	
Number of Messages per Peer	Updates sent	1	0	1
	Updates received	β	N	0
	ACKs sent	β	N	0
	ACKs received	1	0	1
	Total	$2 \cdot (\beta + 1)$	$2 \cdot N$	2
Traffic per peer	Total	$2 \cdot (\beta + 1) \cdot size_m$	$2 \cdot N \cdot size_m$	$2 \cdot size_m$
Example: $\beta = 8$, $N = 10^6$, $UI = 60s$, $size_m = 3kb$.				
Average age		6min 47s	1min	
Number of messages per peer per $UI = 60s$		18	2,000,000	2
Traffic per $UI = 60s$		54kb/min	6,000,000kb/min	6kb/min
Traffic per second		0.9kb/s	100,000kb/s	0.1kb/s

Table 15: Comparison of the Centralized Approach with the Monitoring Tree

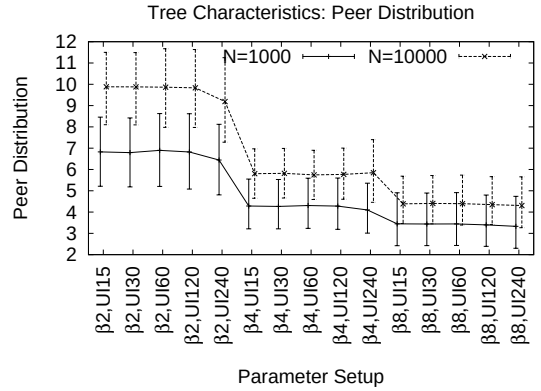
4.4 SIMULATION OF PARAMETER VARIATION AND SMOOTHING

In this section, we present the evaluation results of the simulations regarding the investigation of the influence of the parameters UI and β, the monitoring update interval and the branching factor of the SkyEye.KOM tree, as well as the parameters in the exponential- and media-based smoothing. Through the variation of the update interval and the branching factor, we analyze the effects on the tree characteristics and monitoring quality as well as the traffic and messaging overhead. The variations of the smoothing parameters give us insights on the effects on the monitoring quality, which do not influence the tree characteristics or costs.

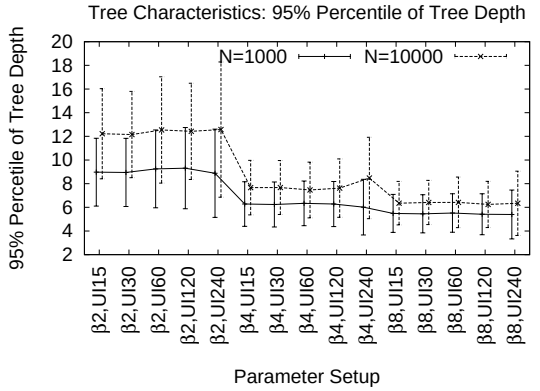
4.4.1 *Results on the Tree Characteristics and Monitoring Quality*

One main aspect of the quality of the monitoring solution SkyEye.KOM is the characteristics of the corresponding SkyEye.KOM monitoring tree. We depict in Figure 36 the peer distribution in the tree and tree depth with varying parameters UI and β. The influencing parameter for the tree characteristics is the branching factor β. The peer distribution is depicted in Figure 36a, with exponentially increasing β, the average peer position decreases linear. Considering the tree depth, Figure 36b shows the maximum tree depth of 95% of the peers over time. It neglects outliers in the tree and shows the main tree depth. This figure validates the modeled tree depth that is shown in Figure 26a. Figure 36c presents the average maximum tree depth over the simulation. It shows a stronger variance and contains random influences based on the ID distribution of the peers. One pair of peers located closely to each other in the ID space results in a deep branch of the tree in the corresponding location. Despite the random influence based on the ID distribution of the peers, the tree does not degenerate and increases the level size only by a few levels.

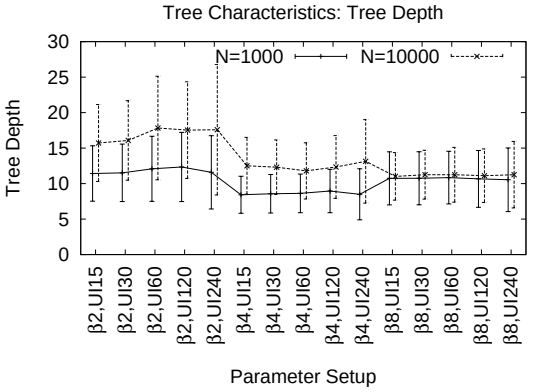
The tree depth has a great influence on the monitoring quality and the resulting freshness of the monitoring information. We present in Figure 37a the averaged freshness of the monitoring information at the root over the simulation time. We observe on the one hand that an increasing branching factor leads a lower information age, which is due to the lower depth of the monitoring tree. On the other hand, the linear increase of the update interval leads to an linear increase of the information age at the root. This effect has also been shown in the analytical results in Figures 30 and 31. However, the transmission delay of a message in the tree is only very small in comparison to the larger update intervals, thus



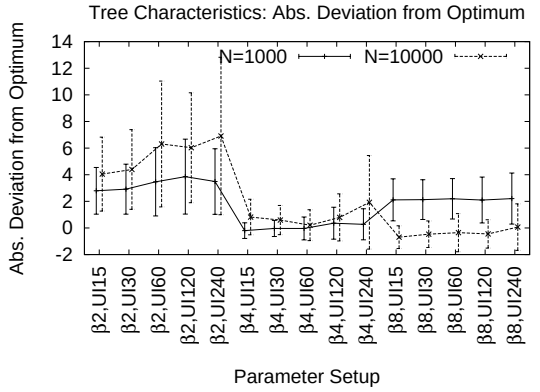
(a) Peer Distribution in the Tree



(b) 95% Percentile of Tree Depth



(c) Tree Depth considering all Peers



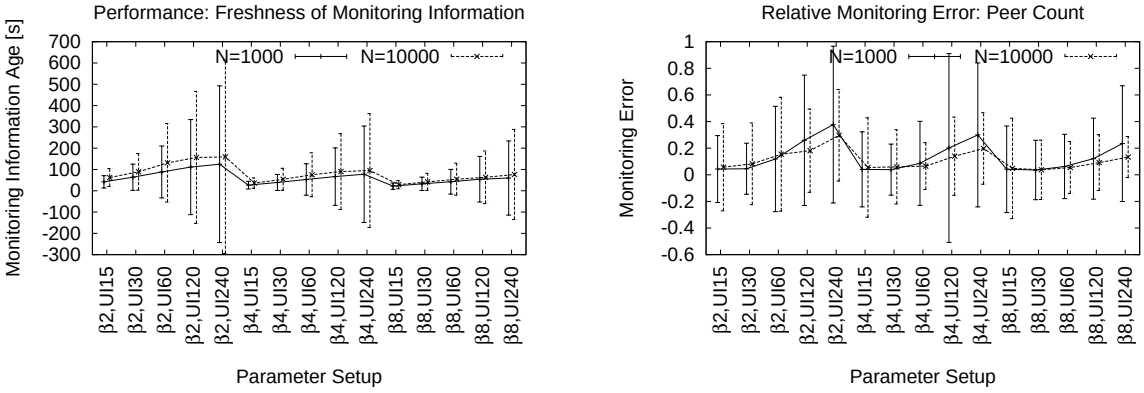
(d) Absolute Deviation of Tree Depth of Optimum

Figure 36: Effect of the Branching Factor β on the Tree Depth

the effects of the tree depth on the resulting freshness are smaller than that of the decreased update intervals.

Regarding the monitoring quality, we focus on three main metrics, the peer count as well as the reference signals Sine and ZigZag, both with an interval of $T = 30m$. The relative error in monitoring the peer count is presented in Figure 37b. It resembles the observations made considering the freshness of the information view at the root. With linearly decreasing UI and exponentially increasing β the relative error decreases linearly. This observation affirms the correlation depicted in the Figure 31d of the analytical model.

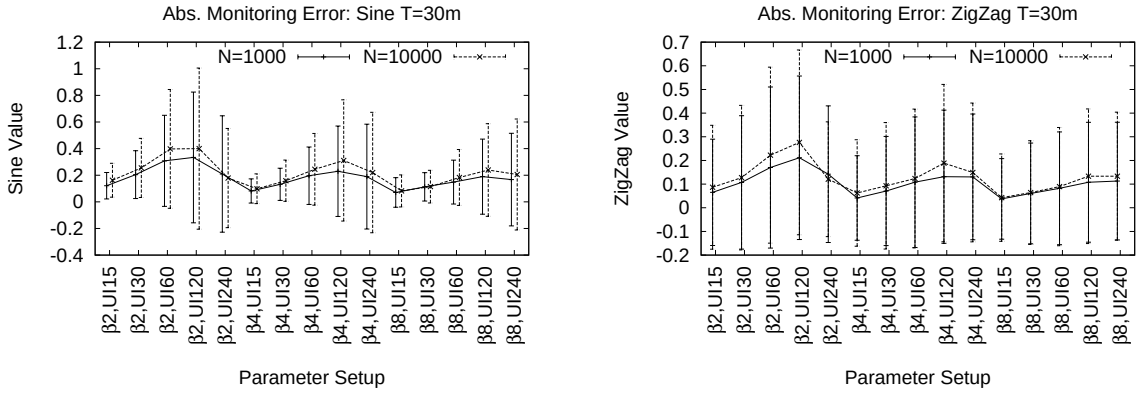
A more expressive metric for the quality of the monitoring solution are the monitored metrics of the reference signals Sine and ZigZag. These signals are used for benchmarking the solutions, as they do not depend on the topology or mechanism characteristics. We synchronously induced these reference signals at the peers in the simulator. The main question is, how well SkyEye.KOM monitors these reference signals under variations of the parameter settings for UI and β. The results are depicted in Figure 38a for the Sine signal and Figure 38b for the ZigZag signal. The figures underline the observations regarding the effects of the parameters UI and β on the freshness and quality of the monitoring view. An additional view on the quality of the monitoring is given in the evaluation of the costs. We present these metrics in the next subsection and show besides the simulation view also the monitored overhead estimates. The precision of the monitoring solution is underlined within these graphs as well.



(a) Freshness of Monitoring Information in Root

(b) Relative Monitoring Error: Peer Count

Figure 37: Effect of the Variation of the Parameters β and UI on the Freshness



(a) Relative Monitoring Error: Sine T=30m

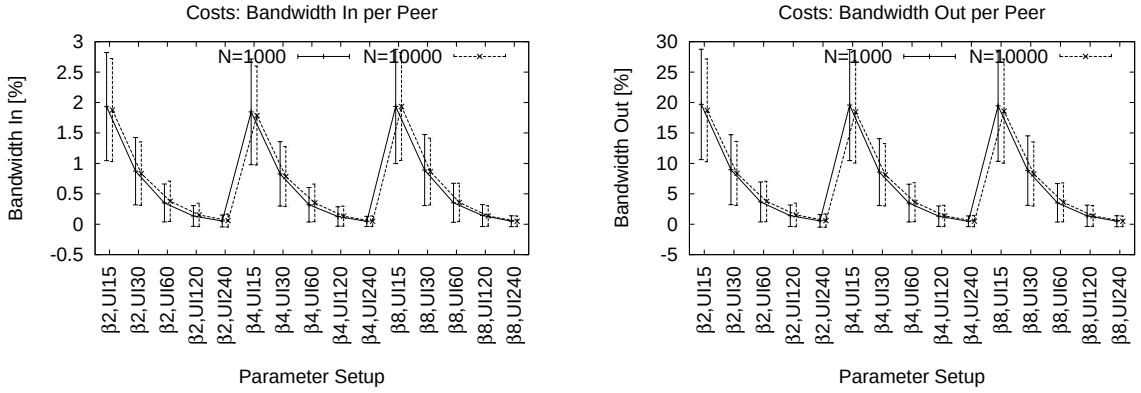
(b) Relative Monitoring Error: ZigZag T=30m

Figure 38: Effect of the Variation of the Parameters on the Reference Sine and ZigZag Signal, T=30m

4.4.2 *Results on the Message and Traffic Overhead*

Having sketched the monitoring quality of SkyEye.KOM under variation of the parameters UI and β, next, we describe the costs in terms of message and traffic overhead. In Figure 40, we show the traffic overhead induced by SkyEye.KOM for monitoring the p2p system classified according to the message types. One main observation is the influence of the parameter UI on the induced traffic. The anti-proportional correlation between the update interval and the traffic and messaging overhead is defined by the direct link of the overhead and the frequency of update transmission. This effect is characteristic for proactive monitoring approaches. A second observation is that the main impact on the traffic overhead is given by the updates related to monitoring the system status. One reason for this is that the parameter for attribute updates intervals was chosen as 180s, while the metric update interval was lower.

In the protocol design, we motivated that due to the aggregation of monitoring information regarding the system statistics their size does not grow, while the size of the attribute updates grows while approaching the root. Specifically, the update metric message sizes are not influence by the tree level nor by the number of peers in the network. For the attribute update messages, however, this is the case.However, currently the amount of monitored metrics regarding the system statistics both in the simulations as well as in the prototypical implementation is much larger that the monitored peer information. An overview on the monitored system information in the model, simulation and prototype is given in Tables 2, 3 and 8. The lists of gathered peer-specific information in the simulation and



(a) Average Down Bandwidth Consumption per Peer

(b) Average Up Bandwidth Consumption per Peer

Figure 39: Effect of the Variation of the Parameters β and UI on Bandwidth Consumption

prototype are shown in Table 7. Regarding the system-specific information, we gather in the simulation 320 data tuples (i.e. descriptors, types and names) and related to the peer-specific information 20 data tuples. This relation of metric update sizes to attribute update sizes, effects, in addition to the smaller update interval of the system-specific information, the large impact of the metric updates in the traffic overhead.

In Figure 39, we describe the ratio of bandwidth consumption per peer and the corresponding monitoring error. Please note that the peers have a ten times larger download capacity than upload capacity. The bandwidth consumption correlates with the parameter for update intervals UI and is independent of the branching factor β and N. The parameter UI affects the frequency in which updates are sent and is linearly correlated to the induced costs. Although the parameter β influences the number of incoming messages due to the influence on the number of Sub-Coordinators, in average over all peers the effects are the same as every peer sends only one update message per update interval, only one update message is received in average per peer. The sames counts for the metric ACK messages.

Regarding the frequency of sending messages, we present the overview according to the message types in Figure 41. Roughly one quarter of the messages is related to the metric updates (Figure 41c), one quarter to the metric ACKs (Figure 41d) and half of the messages are related to overlay lookup messages as seen in Figure 41f. In specific, every metric update message is answered with exactly one metric ACK message. In each update period one metric update is sent in average over all peers and one metric ACK is received. In order to send these messages, we use the assumed Functions 3 and 4 of the KBR interface. Function 3 (*nodehandle getNodehandle(key K)*) performs a lookup in the KBR-compliant p2p overlay and thus generates both for the metric update and the metric ACK an overlay message. Once the lookup is successful, the metric update and the metric ACK are sent using the Function 4 (*void send(message M, nodehandle P)*), generating each also a corresponding message. In total, these messages dominate the message overhead. The attribute update messages have an update period of 180s and make up the missing ratio to complete the message rate.

In conclusion, we identified the main influencing message types on the traffic and messaging overhead, namely the metric update and metric update ACK messages. However, the results are predictable, as only the traffic and amount of attribute update messages are linked to the number of peers in the network and the tree level in which the peers are located. The other messages are sent periodically per peer and allow are easy to model and to predict in their behavior.

4.4.3 *Results of Parameter Variations in Smoothing Approaches*

Until now, we discussed the tree characteristics, monitoring quality and induced costs in relation to varying parameters UI and β. The monitoring results were unmodified and disseminated in the tree as aggregated by the root. While keeping the aggregated monitoring information unmodified for propagation results in fresh monitoring information, some monitoring errors and outliers may be

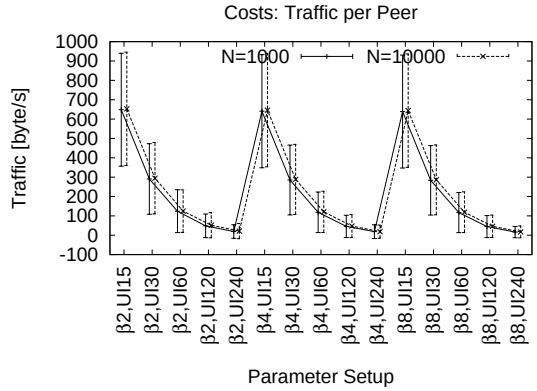
propagated as well. In Subsection 3.4.4, we motivate for this case to adopt approaches, either median-based or exponential smoothing to eliminate outliers. For that, we consider a history buffer for the monitored metrics in the root with the length H. The global view is derived then by adopting the smoothing function on the history buffer.

In the case of median-based smoothing, the middle value in the history buffer is chosen ($H = 3, 5, 7$). In the case of exponential smoothing, the value is calculated based on Equation 3.38. Let m_H be the current measure, then the value to transmit is calculated with the recursive function $s_H = \alpha m_H + (1 - \alpha) s_{H-1}$, giving prior measures (i.e. m_{H-1}) less weight in the sum. The parameter α may be varied to influence the smoothing results. In specific, we investigate in the following the effects of varying the size of the history buffer H for median-based smoothing and varying α in the case of exponential smoothing with $H = 7$. For the evaluation of the influence of the parameter variation, we choose fixed parameters for the update interval and the branching factor, namely $UI = 60s$ and $\beta = 4$.

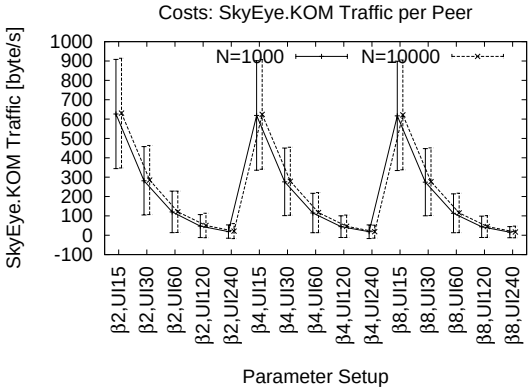
In Figure 42, we depict the effects of the parameter variation in the smoothing approaches on the monitoring quality. Figure 42a shows the effects regarding the freshness of the information and Figure 42b regarding the monitoring error of the peer count. We observe that an increased length of the history buffer leads to imprecise estimates, the same counts for taking prior measurements with a higher weight into account. Although, the freshness and monitoring quality seem to degrade in average, outliers are eliminated.

The negative effects of a large history buffer length and a high weight for prior measurements are also observable for the case of monitoring the reference signals Sine and ZigZag in Figure 43 and for the case of monitoring the total traffic in Figure 44. Smoothing, however, eliminates outliers which relate to large monitoring errors, especially in the case of monitoring the peer count. A peer may change its position in the monitoring tree due to joins and leaves of other peers and the resulting partial reassignment of the ID space. In its new position it induces its previous monitoring status and may lead to a strong bias of the monitoring view. At the root, the impact of errors are lessened through smoothing.

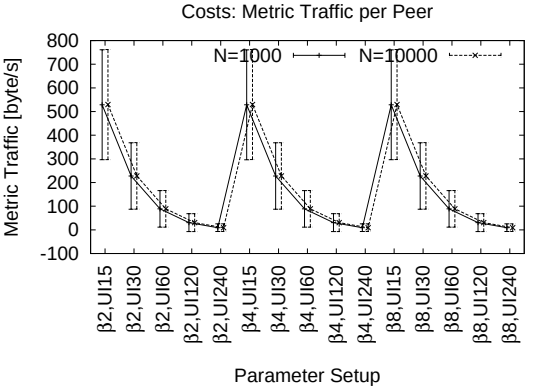
In this section, we presented the evaluation results for the parameter studies of β and UI with varying number of peers. In addition, we analyzed the effects of the parameter setups in the smoothing-based approaches. The simulation results affirmed the results of the analytical model, especially with regard to the characteristics of the monitoring tree and the freshness of the information. Next, we observing a single simulation run, as it gives a detailed view on the behavior of SkyEye.KOM over the time. Additionally, we present for comparison the results of the smoothed and synchronized approaches.



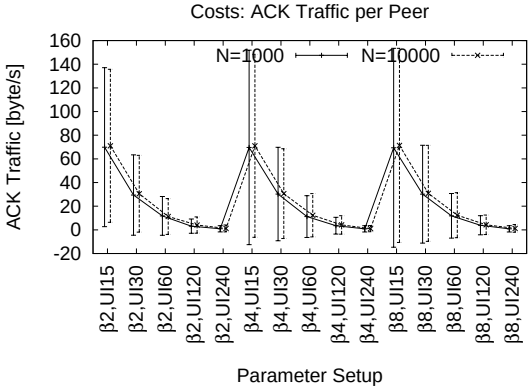
(a) Total Traffic Overhead



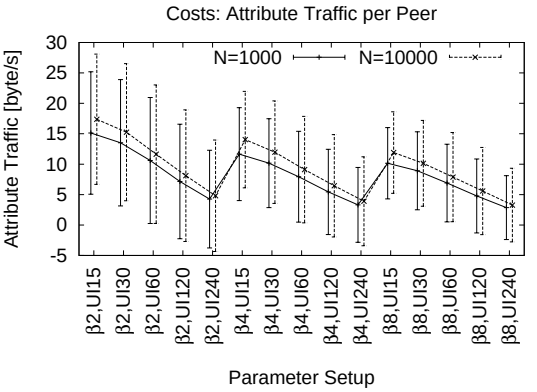
(b) Total SkyEye.KOM Traffic Overhead



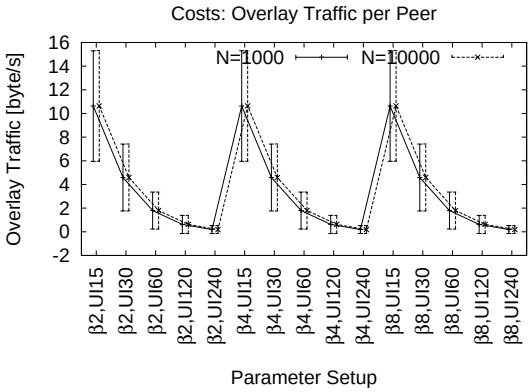
(c) Total Metrics Traffic Overhead



(d) Total ACK Traffic Overhead

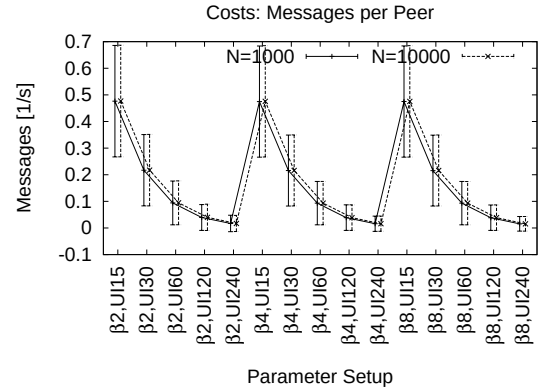


(e) Total Attributes Traffic Overhead

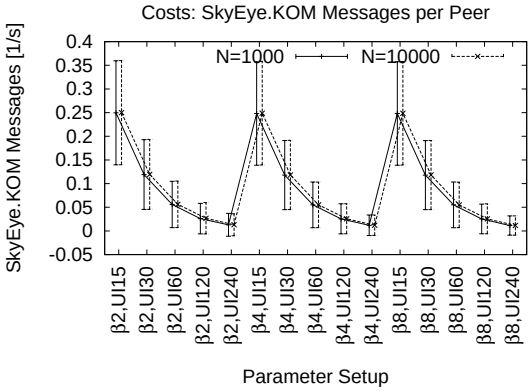


(f) Total Overlay Traffic Overhead

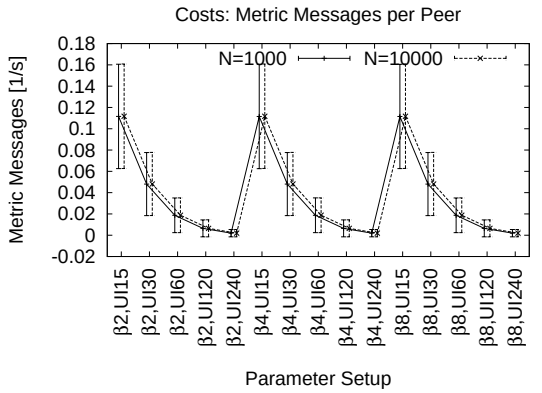
Figure 40: Effect of the Variation of the Parameters β and UI on Traffic Overhead



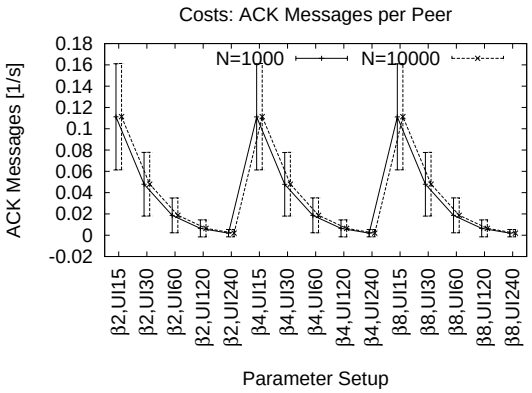
(a) Total Message Overhead



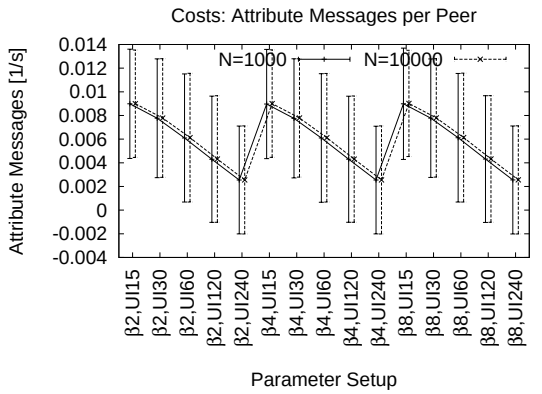
(b) Total SkyEye.KOM Message Overhead



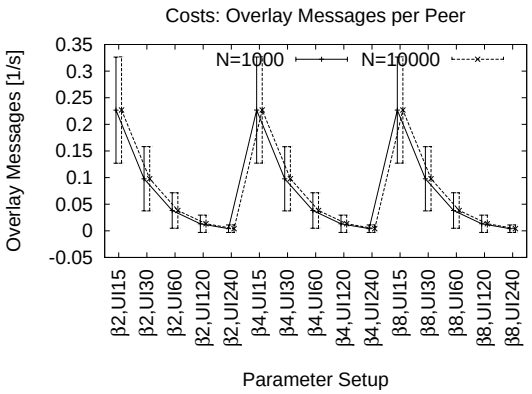
(c) Total Metrics Message Overhead



(d) Total ACK Message Overhead

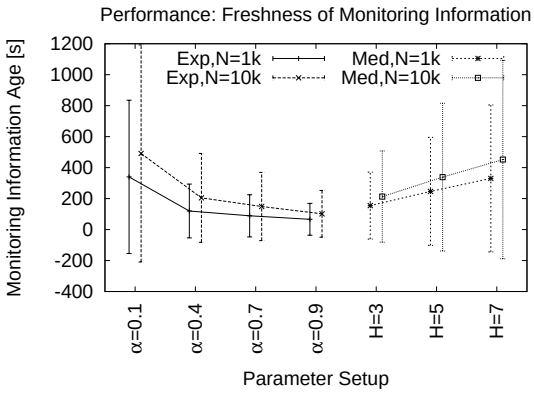


(e) Total Attributes Message Overhead

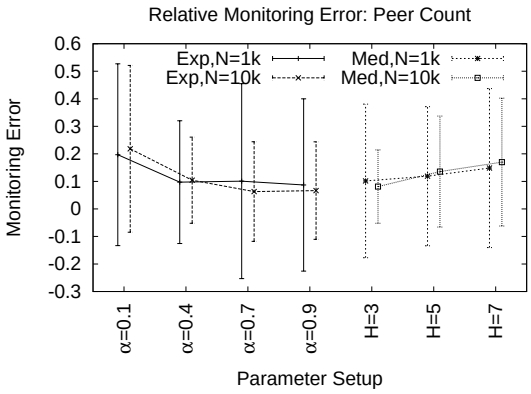


(f) Total Overlay Message Overhead

Figure 41: Effect of the Variation of the Parameters β and UI on Message Overhead

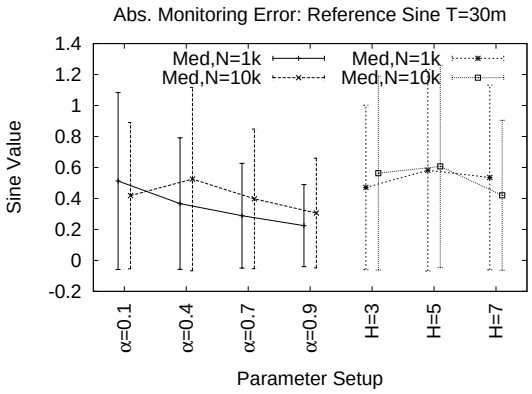


(a) Monitoring Information Freshness

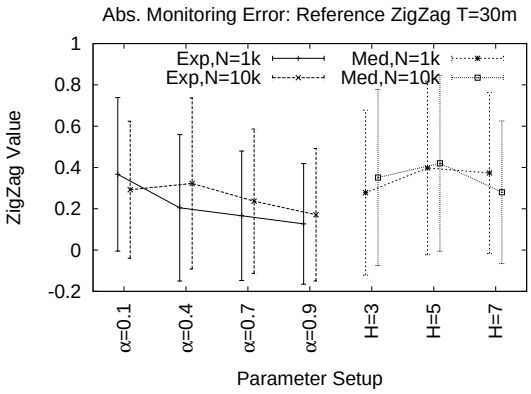


(b) Relative Monitoring Error: Peer Count

Figure 42: Effect of Smoothing on Information Freshness and Monitoring Error

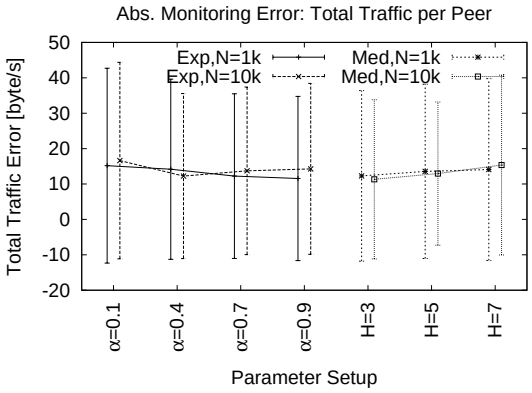


(a) Relative Error: Reference Sine T=30m

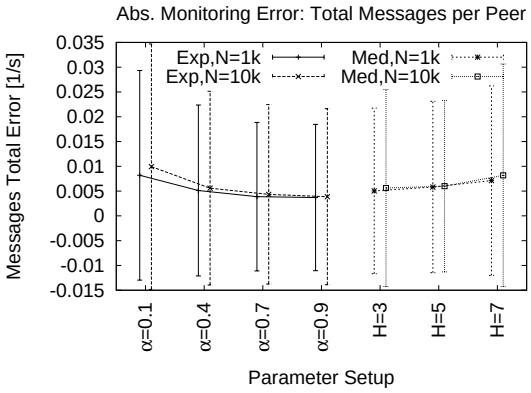


(b) Relative Error:Reference ZigZag T=30m

Figure 43: Effect of Smoothing on Reference Sine and ZigZag Signal Monitoring



(a) Relative Error: Traffic Overhead



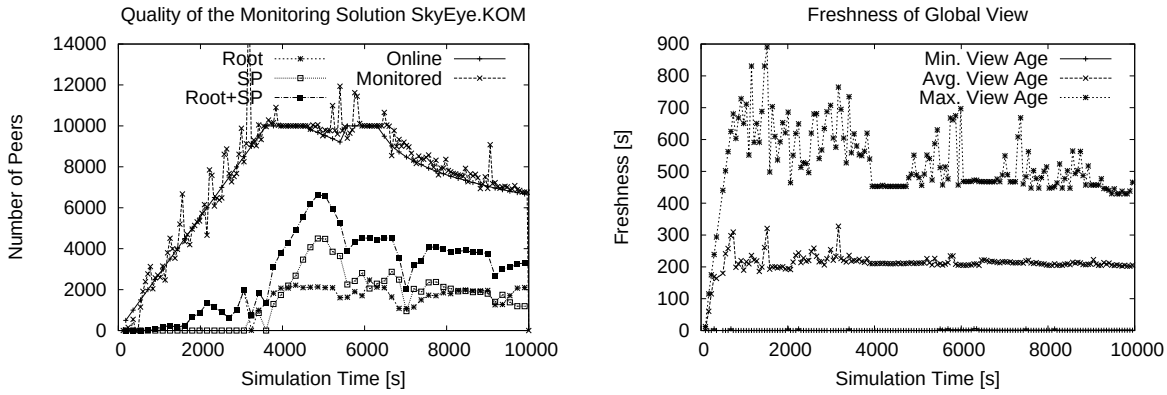
(b) Relative Error: Messaging Overhead

Figure 44: Effect of Smoothing on Traffic and Message Overhead Monitoring

4.5 DETAILED VIEW ON AN INDIVIDUAL SIMULATION RUN

In this section, we present a single simulation run in order to describe the characteristics of SkyEye.KOM in more detail. The evaluation is classified in three parts. First, we give a detailed view on the tree characteristics which affect the monitoring quality related to the global system view. Next, we investigate the performance of the capacity-based peer search and describe the quality of query resolving in detail. At last, we present the traffic and message overhead of the nodes in the simulation ranked by the quantity per peer. This gives us insights on the load level and load distribution in the p2p network.

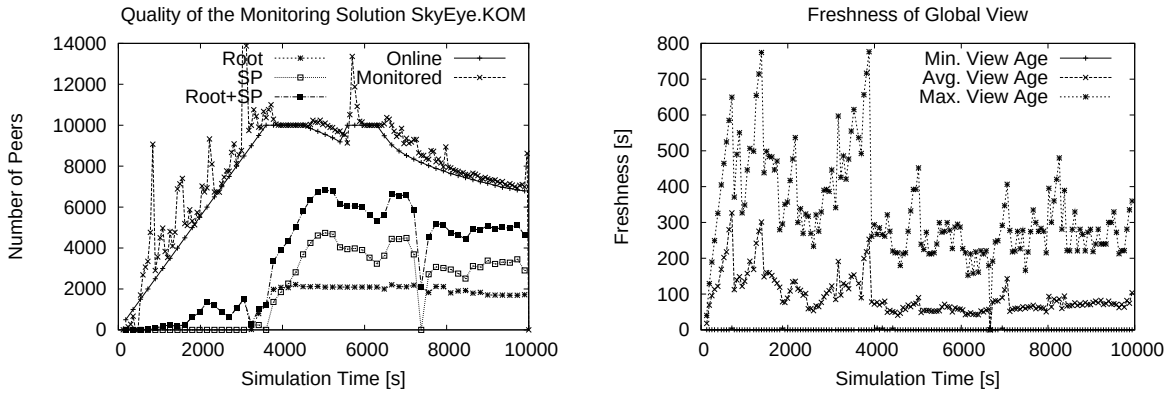
The main setup of the simulation, i.e. the evolving of the peer count over time and the actions of the peers, are depicted in Figure 45a. As parameter setup we chose $UI = 60s$, $\beta = 4$ and we simulated in a network with $N = 10000$. First, 10000 peers join in a period of $t = 0 - 3600s$, then, after a stabilization phase, we induce from $t = 4500 - 5700s$ exponential churn resulting in 10 % of the peers leaving the system, after the churn phase, the number of peers recovers to 10000 and stabilizes. At $t = 6500s$, we induce KAD churn based on measurements in the KAD network which were presented in [SENB07]. The number of peers drops in the next 3500s to 70% of the initial peer count. During the simulation each peer states a query with a probability of 5% in each update interval.



(a) Peer Count in Normal View

(b) Freshness of Normal View

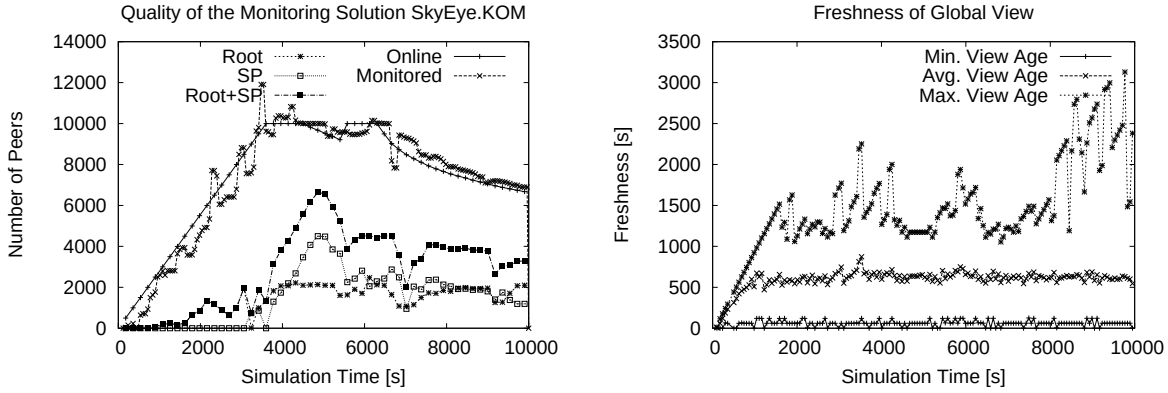
Figure 45: Monitoring Quality of Normal Approach in SkyEye.KOM



(a) Peer Count in Synchronized View

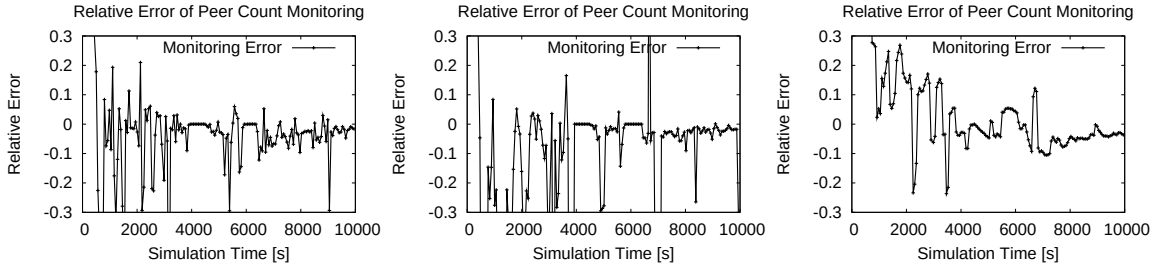
(b) Freshness of Synchronized View

Figure 46: Monitoring Quality of Synchronized Approach in SkyEye.KOM



(a) Freshness in Smoothed View (b) Freshness of Smoothed View

Figure 47: Monitoring Quality of Smoothing Approach in SkyEye.KOM



(a) Normal Monitoring View (b) Synchronized Monitoring View (c) Smoothed Monitoring View

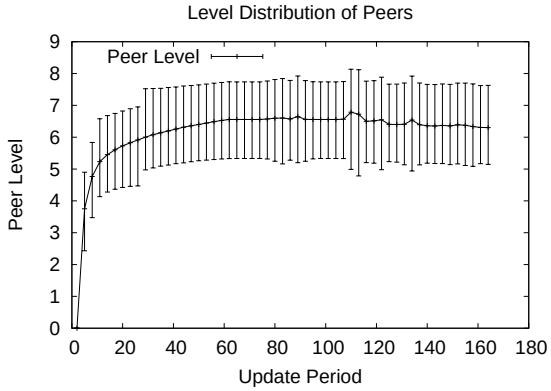
Figure 48: Relative Error of Peer Count Monitoring

4.5.1 *Results on the Tree Characteristics*

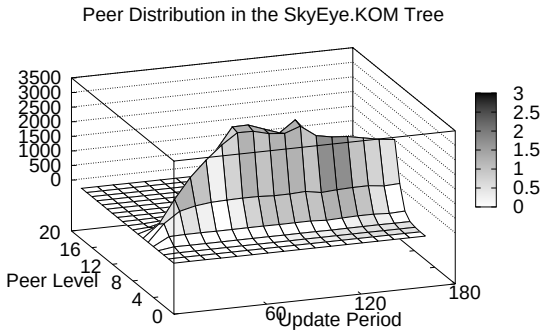
We observe in the Figure 45a that the peer count is monitored with varying quality. Figure 48a gives us more details on the monitoring quality, in which we present the relative monitoring error. The corresponding results for the synchronized monitoring view are shown in Figure 46a and 48b, for the smoothed monitoring view in Figures 47a and 48c.

One characteristic aspect of the tree-based monitoring approach are the peaks in the monitoring view during churn. These peaks occur due to the reassignment of the peer positions in the tree. We emphasize in addition that in time of no churn, the relative error is 0 and all peers are monitored correctly. For the smoothed view, as proposed in Section 3.4.4, the peaks are effectively filtered. However, the smaller variation in the monitoring error comes at the cost of an older monitoring view delivered to the peers in the tree. As a third option, we evaluated the synchronized gathering and dissemination of the monitoring view as presented in Section 3.4.4. The synchronized monitoring view shows more variation in the monitoring view due to the quicker transport of the monitoring data to the root. In the case of irregularities in the tree, the view is still gathered very quickly getting a snapshot of the (erroneous) current status.

Next, we analyze the tree characteristics which help to validate the analytical results on the tree size. The tree shape is independent from the smoothing approach. In Figure 49a, we depict the peer distribution in the tree over time. The average peer level of about 6.5, and matches the analytical results, as depicted in Figure 31b. Most of the peers (about 75%) are located around level 6 and 7, and 95% of all peers between level 5 and 8. We see that during the churn phase from $t = 4500s$ to $t = 5700s$ the peer count drops, but does not affect the shape of the tree much. The same counts for the phase of KAD churn at the end of the simulations. Although the peers fail, their positions in the monitoring tree are instantly reassigned to other peers in the network.



(a) Average Tree Depth and Standard Deviation

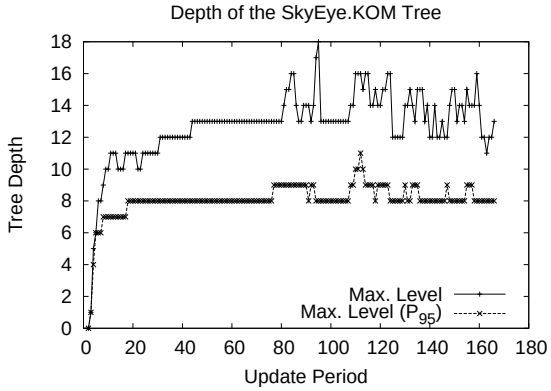


(b) Tree Evolving over Time

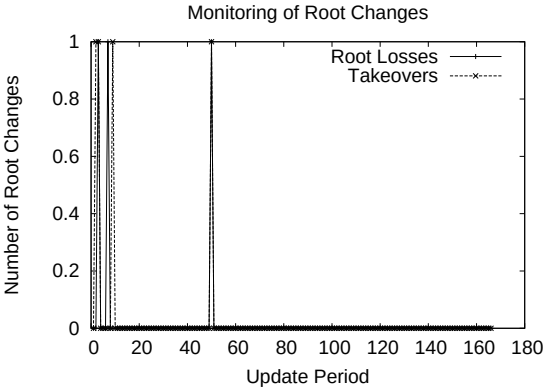
Figure 49: Development of the Tree Depth over Time

In Figure 50a, we depict the tree depth, both of all peers and of 95% of the peers. Considering only 95% of all peers, we see that in the stable period of $t = 3600 - 4500$ (or update periods 60 to 75) the optimal tree depth is hold. In addition, the tree depth varies only slightly in only a few levels. Regarding all peers, the tree depth is in general 50% deeper, resulting from a few nodes that are positioned near to each other in the ID space and lead to a deep branch of the tree in which they are located. Close IDs in the ID space result in small responsibility ranges in the DHT, and thus a small chance to have a Domain key of higher levels inside.

In 50b, we depict the root changes in the tree. Clearly, with the root change at the 50^{th} update interval, i.e. in the minute before $t = 3000s$, an outlier is generated in the tree through the monitoring view, as depicted in Figure 45a. Thus, root changes have influence on the monitoring quality. We address this issue with smoothing. The tree depth is as expected logarithmic in growth and the peers are located at only a few levels. We further observe that the tree is not degenerated, but is only a few levels deeper than the optimum. Hence, we observe a predictable behavior of the tree that match the evaluation results of the previous section and the results of the analytical model.

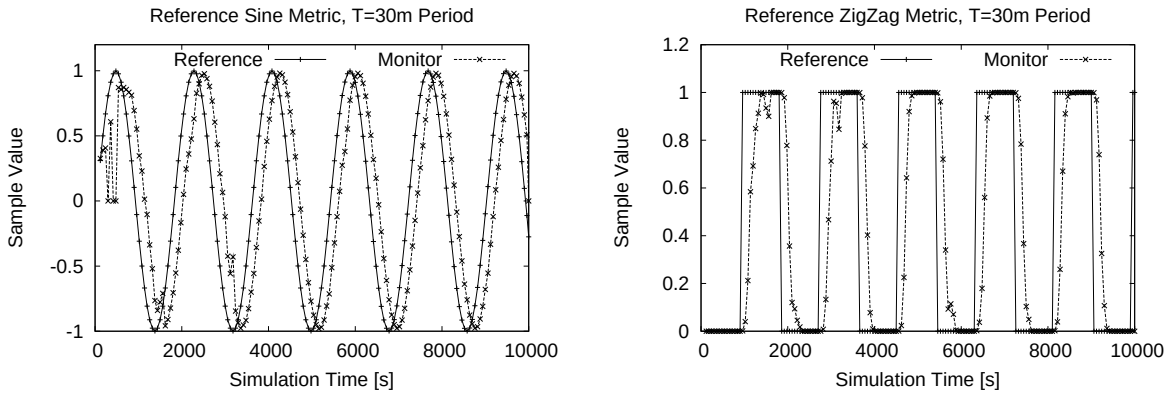


(a) Maximum and 95% Percentile of Tree Depth



(b) Root Changes during the Simulation

Figure 50: Depth of the SkyEye.KOM Monitoring Tree and Occurring Root Changes



(a) Monitoring Reference Sine, T=30m

(b) Monitoring Reference ZigZag, T=30m

Figure 51: Monitoring Quality on Reference Sine and ZigZag Signal - Normal Approach

4.5.2 *Results on the Monitoring Quality*

In this subsection, we focus on the monitoring precision of SkyEye.KOM, both, for normal, smoothed and synchronized update view. As smoothing is only applied on the gatherer information in the root, it does neither affect the monitoring overhead nor the performance of the capacity-based peer search.

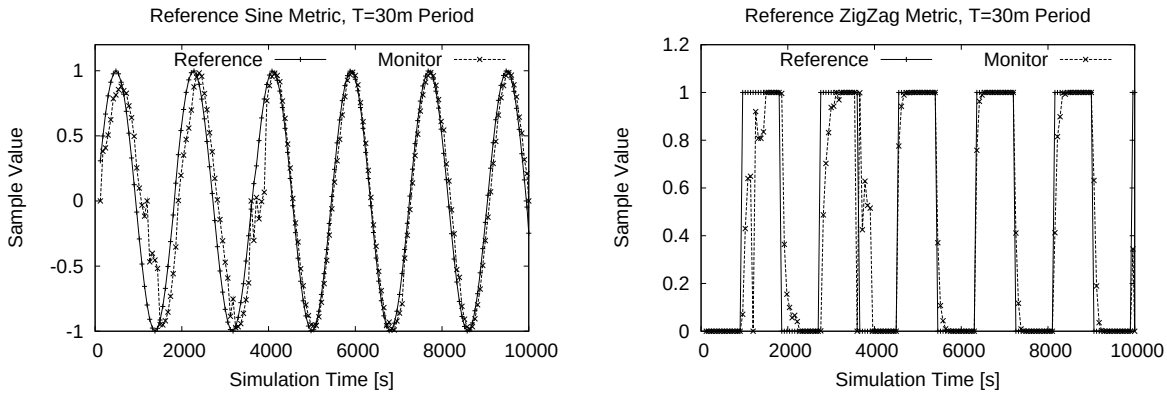
We depict the average freshness of the monitoring information used for the global view in Figures 45b, 46b and 47b. The first figure depicts the freshness of the normal approach for monitoring, the second shows the freshness of the synchronized approach and the third shows results of the the smoothed monitoring. Here, we observe that the freshness of the unmodified monitoring view is with 200s as predicted by the analytical model for $UI = 60$ and $\beta = 4$ as shown in Figure 31b. The synchronized approach provides a monitoring view that is twice as fresh as the normal view. Here, the effects of the synchronized updates are observable. Although, the maximum information age is similar, the average freshness below 100s in average. The smoothed view is in average twice as old as the unsmoothed, this is related to the median smoothing approach which uses a history buffer size of $H = 3$. Thus, in average always the second value is taken, which is twice as old as the current measure of the global view.

A more detailed analysis of the monitoring quality is given by observing the monitoring of the reference signals Sine and ZigZag. The corresponding results for normal, synchronized and smoothed monitoring we give in the Figures 51, 52 and 53. While the normal view matches the variation of the Sine and the ZigZag signal with a short delay, the smoothed values show a larger delay and a limitation in the highest values that the monitored values take. The smoothed approach aims at avoiding extreme values and results in a bound on the slope of the monitoring values. In addition, we observe the effect that all peers in the tree apply the smoothing approach and thus, drastic changes are propagated only slowly. This effect is best to see at the monitored ZigZag values, which only slowly adapt to the reference signal. With regard to the synchronized approach, we observe the best match to the reference signals. Here, the effects of the accelerated gathering and dissemination process take effect.

As a conclusion, we state that the unmodified monitoring view provides a fresh view on the status of the system. Its behavior and freshness are predictable and can be controlled by the parameter defining the update intervals UI. In the simulations, we used an update interval of $UI = 60s$, a value which is quite large, resulting in only one metric update message per minute per peer. A smaller value for UI leads to a fresher view as we have shown in Figure 31. Another important observation is that the simulation results match the analytical results and validate them.

4.5.3 *Results on Performing Capacity-based Peer Search*

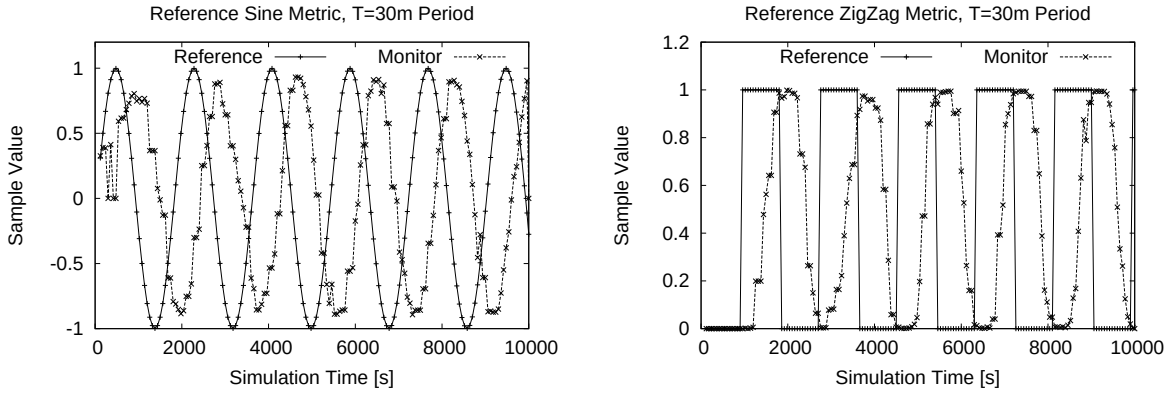
In the following, we focus on the performance of the capacity-based peer search provided by SkyEye.KOM in the simulator. The capacity-based peer search in SkyEye.KOM operates on the proactively



(a) Monitoring Reference Sine, T=30m

(b) Monitoring Reference ZigZag, T=30m

Figure 52: Monitoring Quality on Reference Sine and ZigZag Signal - Synchronized Approach



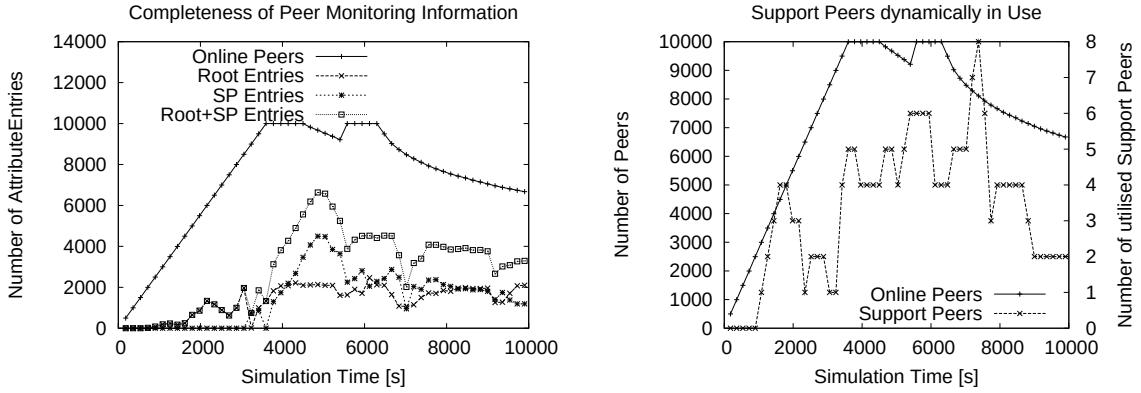
(a) Monitoring Reference Sine, T=30m

(b) Monitoring Reference ZigZag, T=30m

Figure 53: Monitoring Quality on Reference Sine and ZigZag Signal - Smoothed Approach

gathered, unaggregatable peer-specific information. In Figure 54a, we show the amount of attribute entries that are available at the root. As the root is selected based on its ID, it may be a weak peer with low capacities to handle the load of large attribute sets. We explicitly considered this case and modeled every peer in the simulations with heterogeneous capacities. In the case of a weak Coordinator in the tree, a Support Peer is chosen to assist. In Figure 54b, we show the variation of Support Peers over the simulation time. We observe that the number of Support Peers vary, as well as their offered capacity. Nevertheless, their capacity for handling attributes is larger than the capacity of the root and they assist to increase the maximum amount of observable attribute entries in the tree.

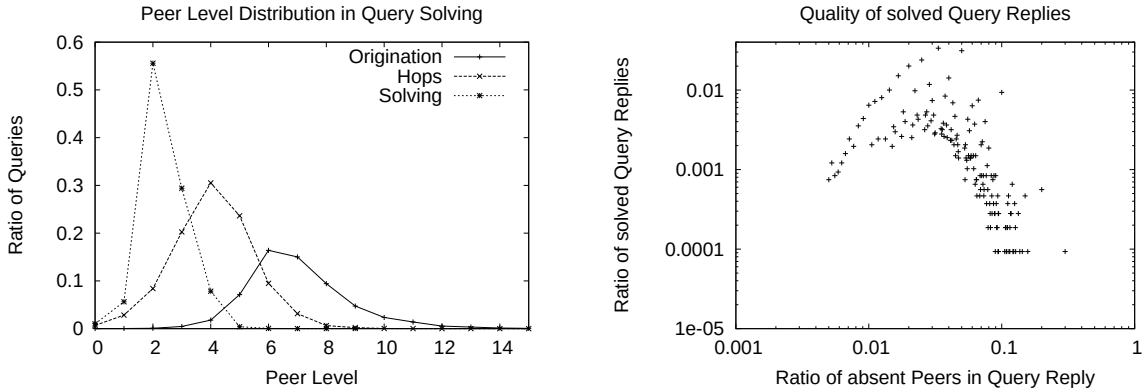
The obtained attribute information on the peers is used then for capacity-related queries. In the simulations each peer states a query with a probability of 5% in each update interval. The corresponding distribution of required hops in the SkyEye.KOM tree is presented in Figure 55a. In average, slightly more than 4 hops are needed for query resolution. Most of the peers are located at level 6.5 and thus lead to a query resolving around level 2.5. The queries as for a random number of peers ranging from 1 to 200. Some of the peers in the query replies may be absent due to churn in the meanwhile. We describe their ratio in Figure 55b. Although, churn is an influencing factor, still, only less than 0.001% of the replies contain more than 10% of offline peers in their result list. Regarding the largest fraction of peers with invalid entries in the results lists, i.e. those with more than 1% matches contain only less than 4% invalid entries in the results. Most of the reply entries are correct and even the few incorrect one contain only a small fraction of offline peers.



(a) Number of Attribute Entries in SkyEye.KOM

(b) Effective Load Allocation to Support Peers

Figure 54: Effect of the Support Peers in the SkyEye.KOM Tree



(a) Level Distribution of Query Actors

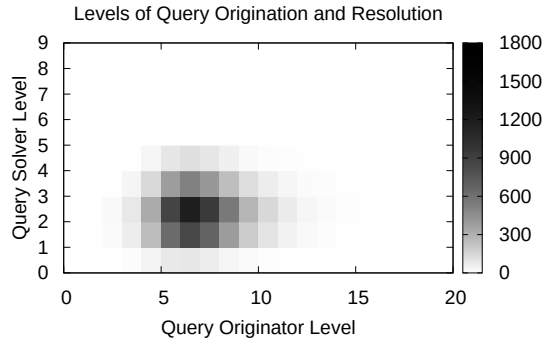
(b) Quality of Solved Queries

Figure 55: Quality and Location of Query Resolving

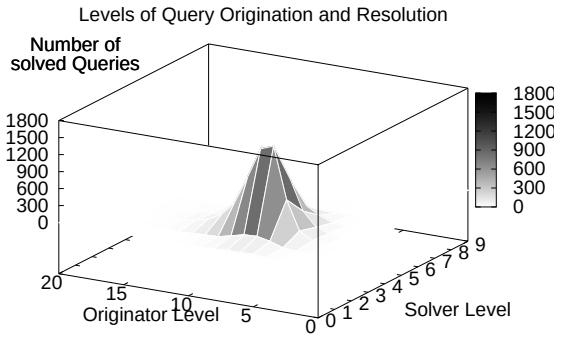
In Figure 56a, we first present the total distribution of the levels of query originating and resolving peers. Here our observation from Figure 55 is validated, most of the peers are located around level 6.5 and the queries are resolved around level 2.5. In the simulations, peers start queries for a random number of peers (1-200), a parameter which takes influence on where the query is resolved. The corresponding view on the query resolving matrix with respect to the number of peers requested, is depicted in Figure 56b. The figure shows the average number of peers requested in the queries that were resolved in this combination of query origination and resolving pairs.

One general observation is that queries for a larger amount of peers tend to be resolved at higher layers in the tree. This is due to the larger amount of attribute entries at these levels that may match to the query. Another observation we make is the concentration of difficult queries resolved at relatively low levels around 5 to 9. We also observe that this peaks are around the query origination level of 5 to 9. Thus a large fraction of difficult queries are resolved directly in the levels, where most of the peers are. This is due to the fact, that the Support Peers are recruited from these levels and a resolved query by them is accounted for the level the Support Peer is originally located at, independent of the level of the Coordinator it serves for.

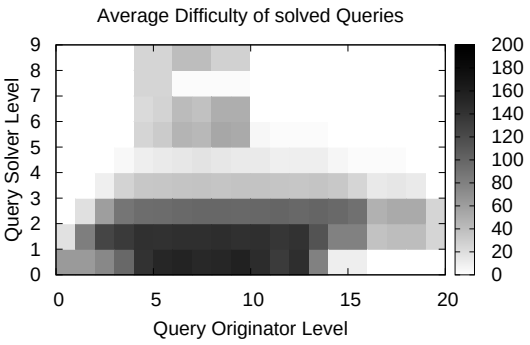
To conclude, we analyzed for the capacity-based peer search the available peer-specific information in the tree and the effects of the Support Peers on that The queries are resolved in a high quality, the set of peers in the results are only rarely offline. Further, we gained deep insights on the position of query initiation and query resolving.



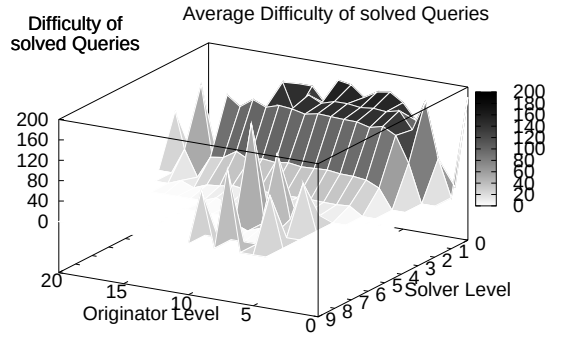
(a) Query Origination and Resolving Levels



(b) Query Origination and Resolving Levels



(c) Query Resolving with varying Complexity



(d) Query Resolving with varying Complexity

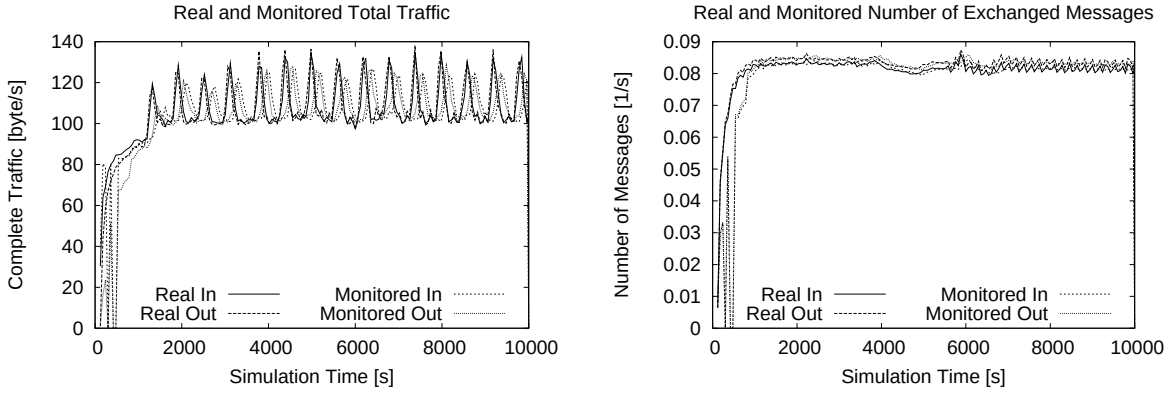
Figure 56: Overview on Query Resolving Positions

4.5.4 *Results on the Message and Traffic Overhead*

Having described the monitoring quality related to the global view on the system status, as well as for gathering and querying the attributes of the peers in the system, next, we focus on the costs needed to provide these functionality.

As main metrics for the costs, we analyze the traffic overhead and message overhead per second. Both metrics and the corresponding monitored metrics are presented in Figure 57. One first observation is that the monitoring view is very precise and follows the corresponding reference cost values with a short delay. In Figure 57a, the traffic overhead is shown, which sums up to about 220 bytes per second in average. This is a very small amount of traffic in relation to the precise monitoring view. The message overhead is depicted in Figure 57b. In average a peer sends a message every 12 second, thus 5 messages per update interval. We analyzed the reasons for this and the classification according to the messaging types in specific at the example of Figure 41. In the synchronized approach, we observe in the Figures 58a and 58b periodic peaks in the transmission of update metric and update metric ACK messages. They show the synchronized transmission of these message, which results in an optimized update delivery and a very fresh monitoring view. In contrast to the averaged traffic and messaging overhead of the simulations with parameter variations (i.e. Figure 40 and Figure 41), we are now able to take a closer look on the load distribution on all peers in the network.

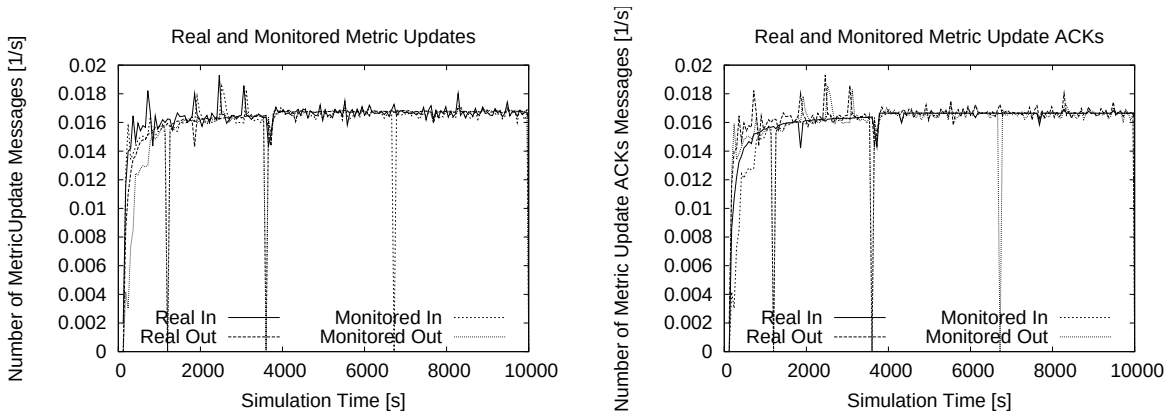
Figure 59 presents the outgoing and incoming traffic and messaging overhead for SkyEye.KOM in specific. One main observation is that the number of sent metric updates and the number of received metric ACKs is nearly constant for all peers. Every peer sends in an update period one update message and receives one. In addition, the size of the metric updates and ACK are independent of the peer position in the tree. Regarding sending ACKs, the load is more unbalanced than for the sent metric



(a) Total Traffic Overhead

(b) Total Messaging Overhead

Figure 57: Overview on the Traffic and Messaging Overhead



(a) Messaging Overhead: Metric Update

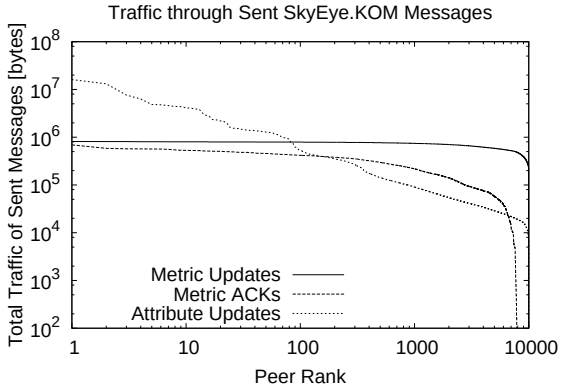
(b) Messaging Overhead: Metric Update ACK

Figure 58: Synchronization Effects on the Messaging Overhead

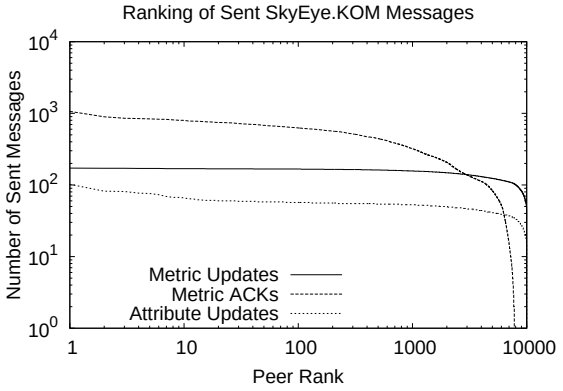
updates. The number of metric updates received and ACKs sent is defined by the distribution of the number of Sub-Coordinators a peer has. Roughly 2000 out of the 10000 peers do not have Sub-Coordinators at all and do not receive any metric updates or send any ACKs.

Another main observation is the logarithmic increase of the traffic related to the attribute updates. This is related to the fact, that the attribute entries cannot be aggregated and thus grow in size on their way towards the root. In relation to the corresponding number of messages, we see that only the traffic, thus the message size, increases but not the number of messages per peer related to the attribute entries. Regarding attribute-specific messages, the load on all peers is nearly constant. Only in the first 10-12 peers a slightly increase in the number of received and sent attribute update messages is noticeable. This is related to their role as Support Peers in the tree. These are the peers that were allocated as Support Peers and identified in Figure 54b.

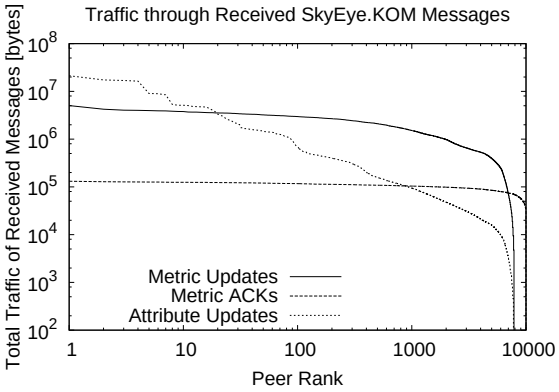
Next, we analyze in Figure 60 the message type related overhead, both in terms of traffic and massage load. In addition, we present both the real cost metrics obtained by the simulator as well as the monitored view provided by SkyEye.KOM. The monitoring view is very precise and close to the real values, only delayed by 200s in average. We observe that the messages that are sent per update interval in average can be related to the ratio of 1:3:3 regarding attribute updates, metric updates and metric ACKs. The metric update and metric ACKs are sent with a frequency of 0.166 messages per second in average, which sum up to 1 message per update interval. Attribute updates have an update interval of 180s, thus their update frequency is a third in comparison to the metric updates. Queries are started at random, with a probability of 5% per peer every update interval.



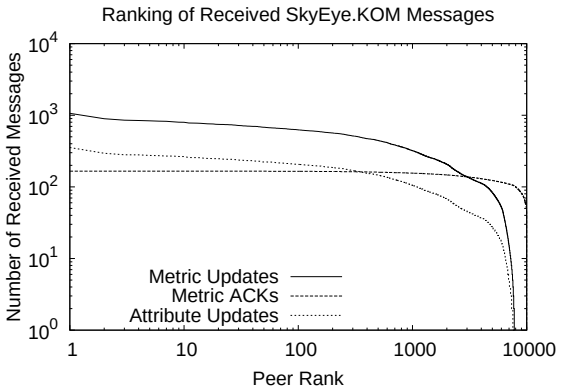
(a) Total Traffic through Sent SkyEye.KOM Messages



(b) Total Number of Sent SkyEye.KOM Messages



(c) Total Traffic through Received SkyEye.KOM Messages

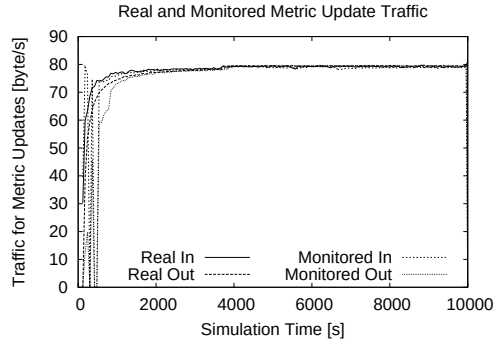


(d) Total Number of Received SkyEye.KOM Messages

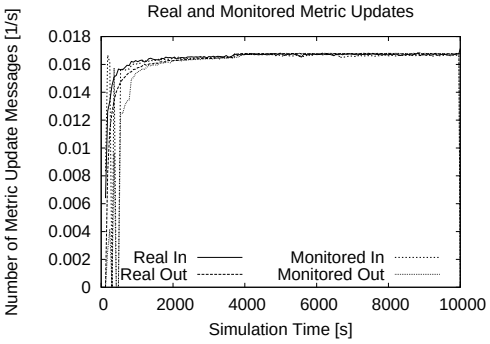
Figure 59: Overview on the Overhead through SkyEye.KOM

Regarding the traffic to messages relation, we see that only the attribute updates do not show a correlation. In specific, in the beginning of the simulation the attribute update related traffic grows, signaling that amount of peers to consider is still growing. This stops after all peers joined the network at $t = 3600s$ and varies from then on, depending on the quality of the peers in the tree and their ability to handle the load.

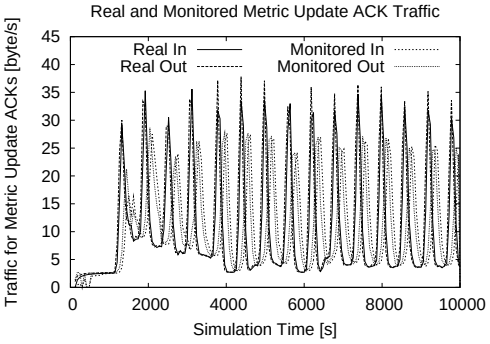
To conclude the evaluation of the traffic and message overhead of SkyEye.KOM, we summarize the main observations. First, the monitoring quality of the cost related metrics is very precise and accurate. We observe that the metric update messages and metric update ACK messages are independent of N and the tree level. They are very predictable in their behavior, as they are generated periodically per peer. The attribute update messages grow in size with increasing level of emitting and with the number of peers in the network. The composition of the message type specific overhead to an overall overhead is well observable, monitored and well understood.



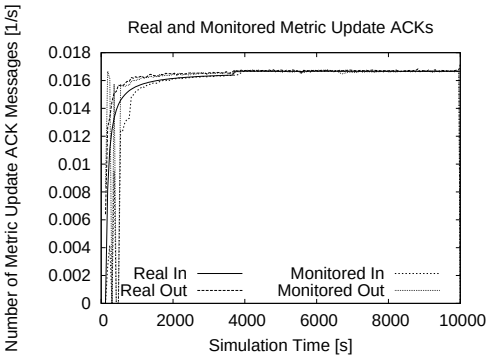
(a) Metric Update Traffic Overhead



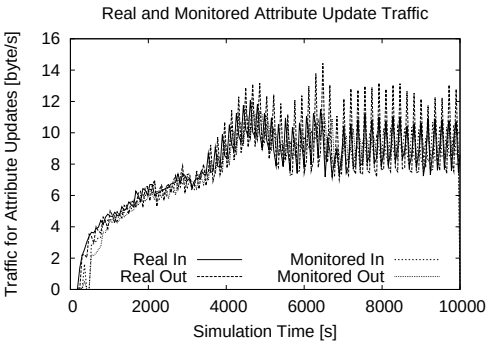
(b) Metric Update Message Overhead



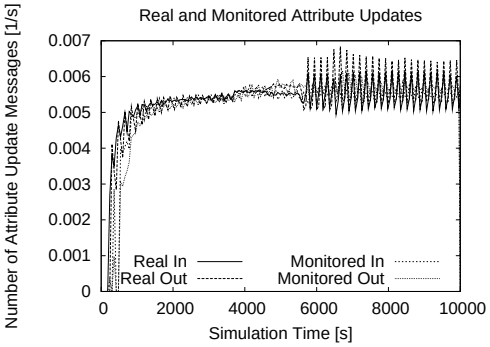
(c) Metric Update ACK Traffic Overhead



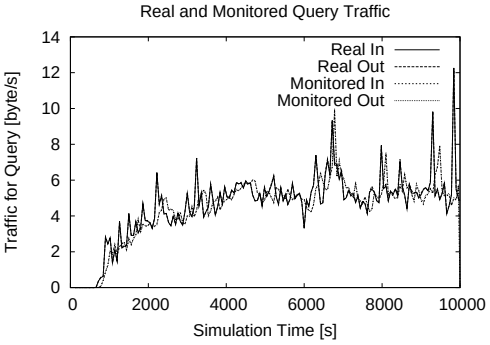
(d) Metric Update ACK Message Overhead



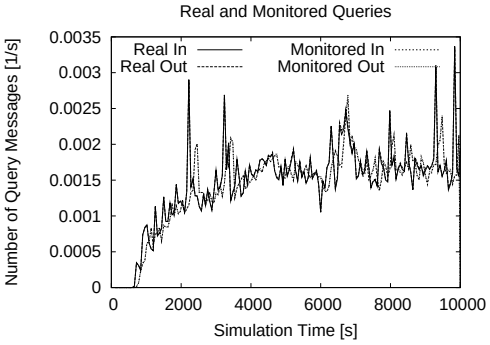
(e) Attribute Update Traffic Overhead



(f) Attribute Update Message Overhead



(g) Query Traffic Overhead



(h) Query Message Overhead

Figure 60: Overview on the Traffic and Message Overhead per Message Type

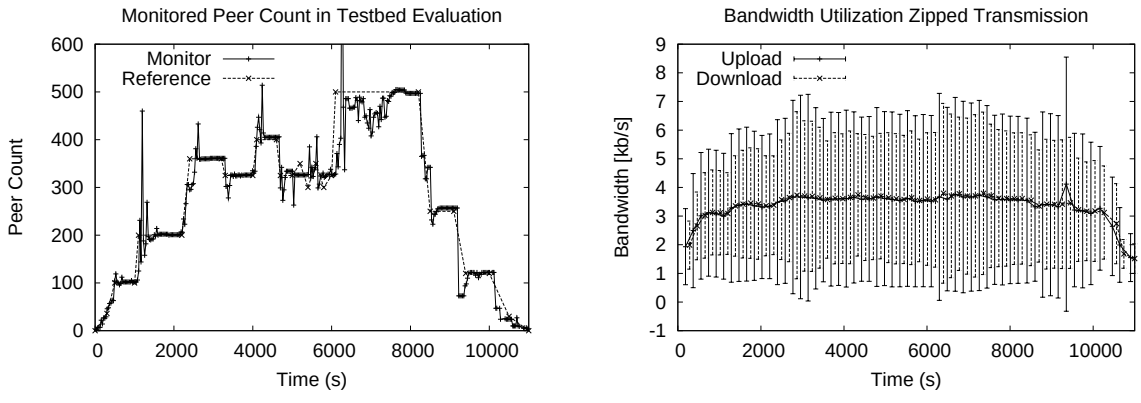
4.6 TESTBED-BASED VALIDATION OF RESULTS

We implemented SkyEye.KOM prototypically on top of FreePastry as stated in the evaluation setup in Section 4.2 and conducted a testbed evaluation with 37 PCs with a 11000s evaluation run with up to 500 peers. We chose as metric and attribute update interval $UI = 5s$ to be able to generate a reasonable evaluation trace in this time. The main focus of the testbed evaluation is to validate the analytical and simulation-based evaluation results and to investigate the effects of real churn on the monitoring performance of SkyEye.KOM. We therefore induced strong churn both for joining the p2p overlay as well as for leaving it. In specific, we analyze the effects of 10%, 20% and 50% churn on the monitoring quality and the robustness of SkyEye.KOM.

In Figure 25b, we depict the workload pattern we applied in the testbed. At first, the peers join in periods of 3 minutes to a total peer count of 100, 200 and 360. For all of these levels a stabilization period of 500-1200 seconds follows the joining phase. Subsequently to this, we induced 10% churn leading to a decrease of the peer count to 325. After a stabilization phase of 700 seconds the peer count was increased in 3 minutes to 400, where it remains for 700 seconds. Then, we turned 20% of the peers of, leading to a peer count of 325 again. In the following, we applied a random churn pattern on the peers, with keeping the total number of peers around 325. In specific, peers where switched off and on in groups of 10 peers. At the time $t = 6000s$, we increased the number of peers in 2 minutes to 500, followed by a stabilization phase of 1800s. From $t = 8200s$ on, we induced twice a fail ratio of 50% to the peers, leading to a peer count of 250 and 120 each followed by a short stabilization time of 600 seconds. From $t = 10000s$ on the peers were shot down one by one.

The monitored peer count is depicted in Figure 61a in comparison to the applied workload. We observe that the monitoring results are precise after a short stabilization time and vary under churn. In specific, during the joining phases, strong monitoring outliers may occur, for example, at time $t = 1200$. This is due to the reassignment of the ID intervals a peer is responsible for and thus the position of the peers in the tree which is derived from this. Due to the new position of the peers in the tree, they induce a biased monitoring view which is based on their previous positions. This effect was also identified in the simulations and addressed with smoothing approaches.

Regarding the costs for monitoring the system status and gathering peer-specific information, we depict in 61b the averaged bandwidth utilization of the peers. It is in average at 3 KB/s for the amount of information related to the system-specific information (as presented in Tables 2, 3 and 8) and peer-specific information (as presented in Table 7). The messages are first compressed before sent, to save bandwidth.



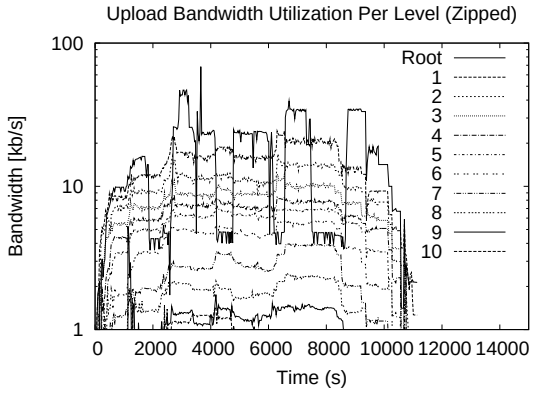
(a) Reference and Monitored Peer Count

(b) Upload and Download Bandwidth Consumption

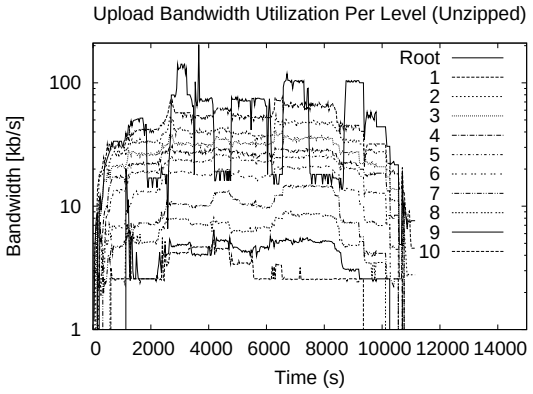
Figure 61: Peer Count and corresponding Traffic Overhead

4.6.1 *Results on the Traffic Overhead*

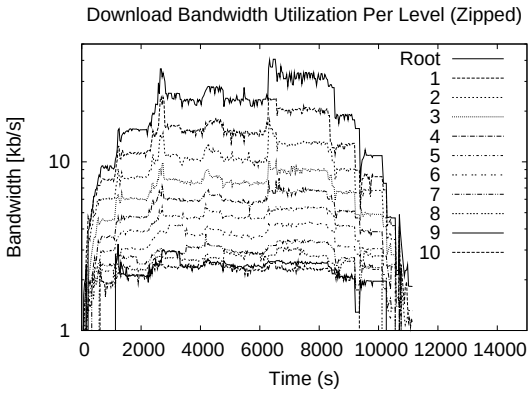
In order to give a more detailed view on the traffic overhead in the tree, we depict in Figure 62 a per-level view on the overhead generated, both on the compressed and uncompressed traffic. Compression of monitoring data is useful as we use long Strings to describe the semantics of the monitored values as well as String descriptions for the type descriptions. There is a significant saving potential on the traffic overhead while using data compression. We observe that the overhead load is directly linked to the level of the peers, with the root having most of the load, the peers at level 1, then at level 2 and so on. This is due to the gathering of peer-specific information, the attributes, that grow with every level on its way to the root. This effect was also observed in the simulations, for example, in Figures 59c and 59a. Another observation is that the upload bandwidth utilization shows higher peaks for the higher levels. However, in average both download and upload utilization are the same as shown in Figure 61b.



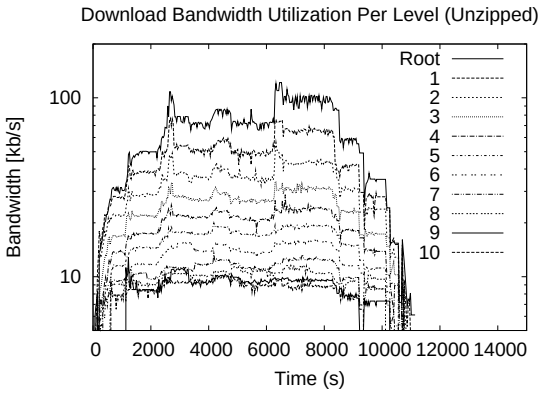
(a) Upload Bandwidth Utilization - Compressed Data



(b) Upload Bandwidth Utilization - Uncompressed



(c) Download Bandwidth Utilization - Compressed Data

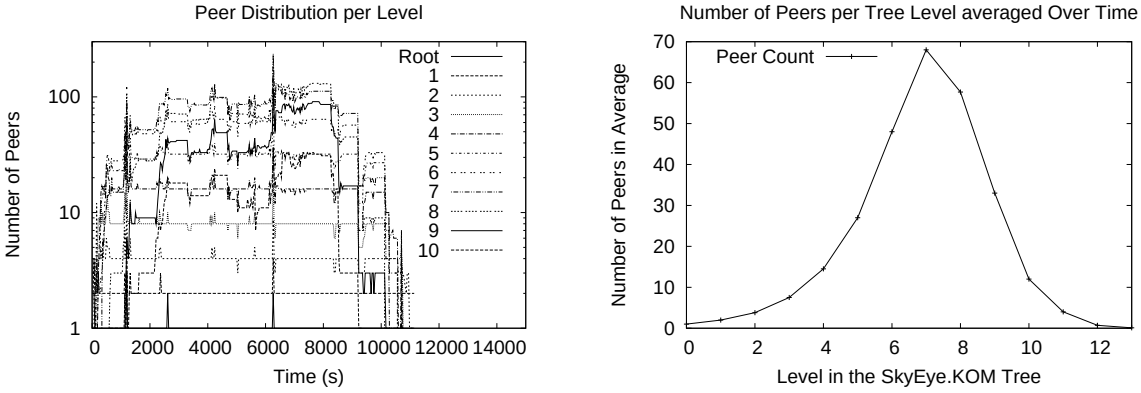


(d) Download Bandwidth Utilization - Uncompressed

Figure 62: Up- and Download Bandwidth Utilization, Compressed and Uncompressed

4.6.2 *Results on the Tree Characteristics*

In the following, we discuss the characteristics of the monitoring tree in the testbed evaluation. An overview on the peer distribution per level is given in Figure 63a. The figure shows that the number of peers per level are logarithmically distributed. An observation which is affirmed in Figure 63b, which gives us the information on the number of peers per tree level averaged over the whole evaluation time. The shape of the distribution affirms the previous observations made from the analytical evaluation and the simulations. The correspondence of the analytical, simulation and testbed results validates the evaluation.



(a) Development of the Peer Distribution per Level

(b) Overall Peer Distribution in the SkyEye.KOM Tree

Figure 63: Tree Characteristics in the Testbed Setup

In the testbed, we also measure as one of the peer-related information, the ID of their Parent-Coordinator. By combining the information of all peers related to this specific attribute, we are able to reconstruct the tree structure in the tree. Snapshots of the evolution of the tree are presented in Figure 64 for the normal case and in Figure 65 for the case of exchanging the root through reassignment of the responsibility space of the root or churn.

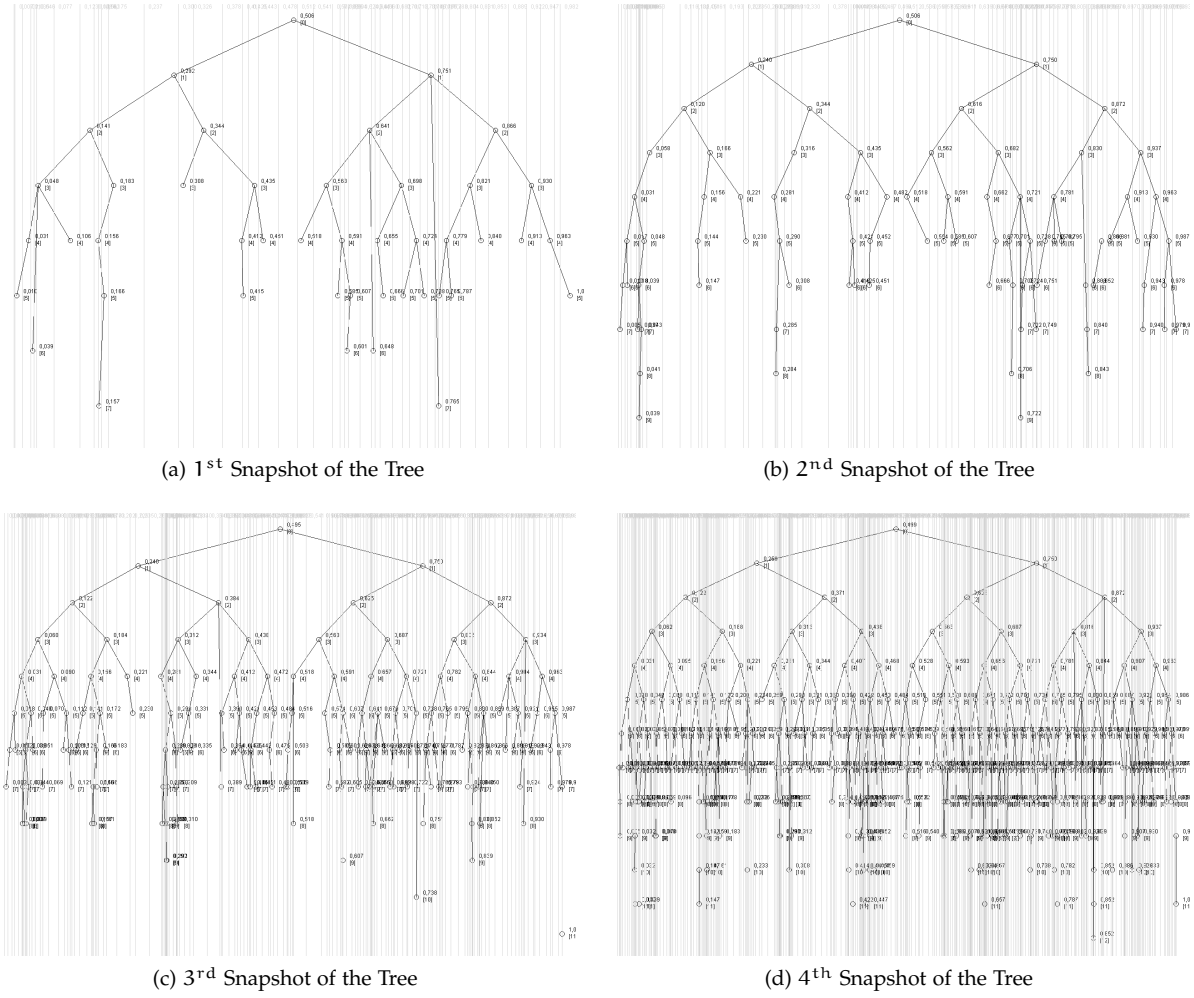
In the normal case, we observe the joining of new peers and their integration in the tree. The tree itself is not degenerated and deviates only in a few levels from the optimum. In specific, the very deep tree branches occur only in positions where the peer density is very high, for example, in Figure 64b in the left branch around the unified ID 0.040. Thus, the total tree depth is directly linked to the densest area of the peer ID distribution in the ID space. However, these areas are typically rare and for the 95% percentile of the peers the tree resembles a near-optimal tree depth, as it was also observed in the model (Figure 26b) and the simulations (Figure 50a).

In the case of the loss of the root, the new assigned root does not have a global view at first, to propagate down the tree. Regarding the monitoring view on the system statistics, the root is updated within the next update interval by its Sub-Coordinators. However, the monitoring of the tree structure (i.e. the ID of the peer-specific Parent-Coordinator) is part of the peer-specific information, which is updated in accordance to the maximum attribute receiving capacity of the peer. However, with time the view on the tree structure is gathered and can be depicted again. One remarkable aspect in relation to the recovering of the global view of the root is that although the root fails, the rest of the tree remains intact. Errors and failures are limited only to the local surrounding of a peer and are quickly resolved. The only global influence of a peer join or failure is due to the possible re-localization of the peers in the tree and the induction of system-specific monitoring information at "wrong" places, which result in outliers in the monitoring view. This issue is addressed in the simulations through smoothing approaches.

To conclude the results on the evaluation in the testbed, we summarize the main observations. As we do not have a reference at hand in the testbed evaluation, except the peer count, the main observations are made in relation to the real load distribution and the behavior under strong churn. While the monitored peer count is close to the testbed setup, the load distribution is well observable and detailed for each level in the tree. The shape of the monitoring tree is observed and monitored and affirms the evaluation results from the analytical model and the simulations. SkyEye.KOM copes with drastic churn of 10%, 20% and 50% of the peers leaving the network. Thus, the prototype demonstrates the feasibility and practical usability of SkyEye.KOM.

4.7 CONCLUSIONS

We have evaluated SkyEye.KOM thoroughly through analytical modeling, large-scale simulations and a testbed setup. Through the analytical model, we identified the main influencing factors and parameters



(a) 1^{st} Snapshot of the Tree

(b) 2^{nd} Snapshot of the Tree

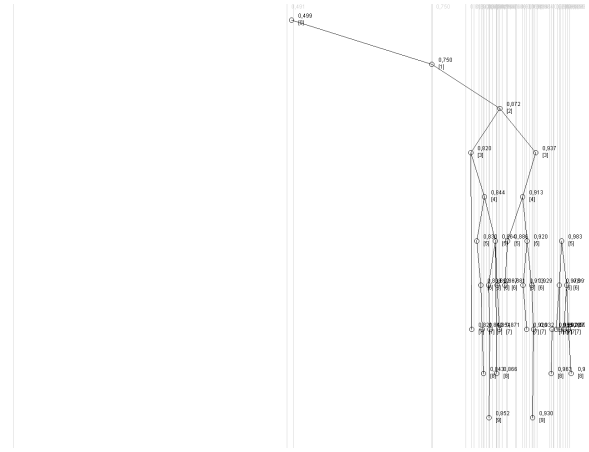
(c) 3^{rd} Snapshot of the Tree

(d) 4^{th} Snapshot of the Tree

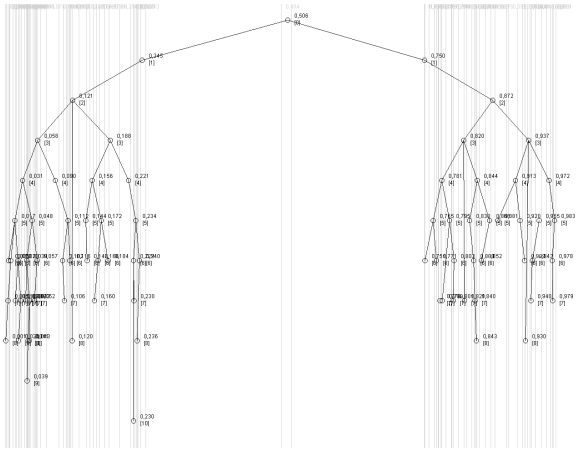
Figure 64: Sample Set of Snapshots on the Tree in the Testbed

in the behavior of SkyEye.KOM. The parameter UI and β describing the update interval of in the monitoring protocol as well as the branching factor of the corresponding monitoring tree influence the freshness and quality of the monitoring view, as well as the induced overhead. The monitoring tree grows logarithmically with the number of peers. The distribution of the levels of the peers in the tree has been modeled and validated through simulations and the testbed setup. It defines together with the update interval the freshness of the system-specific information that arrives at the root and is spread from there to all peers in the tree. Regarding the peer-specific monitoring, we modeled the peer capacities and queries performed on the them. We identified the distribution of the query resolving peers both, in relation to the level of query originating peers as well as the complexity of the query. This model was validated through simulation results.

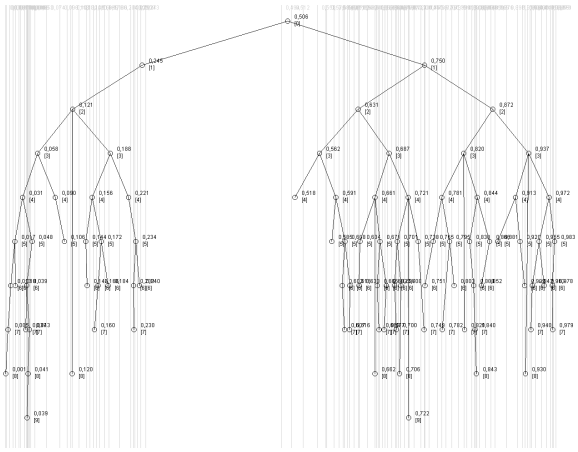
We simulated SkyEye.KOM in a large-scale network in the p2p systems simulator PeerfactSim.KOM. It allows us to use mature models for the transport and network layer as well as the overlays. We evaluated SkyEye.KOM with both 1000 and 10000 peers and analyzed its behavior with a broad variation of the parameters UI and β. We identified the trade-offs regarding the monitoring freshness and the costs, which affirmed the results of the analytical model. We observed a freshness of the monitoring of around 200s in normal mode and 100s using the synchronized approach, while having a traffic overhead of around 110 byte/s per peer in a setup with $UI = 60s$ and $\beta = 4$. In addition, we addressed the issue of outliers in the monitoring view that occurs due to churn and the reassignment of the positions of the peers in the tree. We examined median-based approaches and approaches using exponential smoothing. Here, we focused on the effects of a variation of the history buffer size on which the smoothing operates



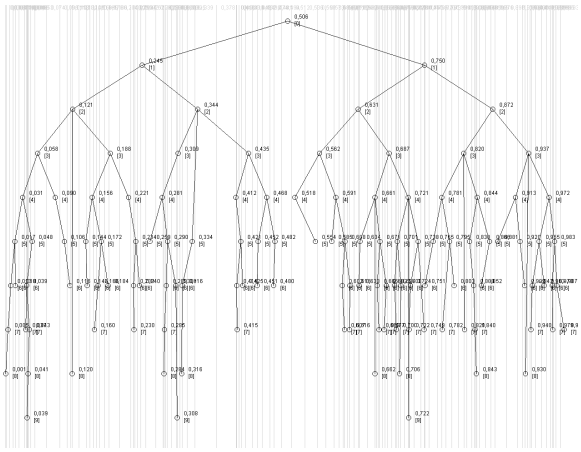
(a) 1^{st} Snapshot of the Tree, Root Change



(b) 2^{nd} Snapshot of the Tree



(c) 3^{rd} Snapshot of the Tree



(d) 4^{th} Snapshot of the Tree

Figure 65: Effect of Churn on the Tree in the Testbed

as well as the factor α in exponential smoothing. Smoothing eliminates the outliers, but results in an older monitoring view of about 500s in a setup with $UI = 60s$ and $\beta = 4$.

Having identified the influence factors of the parameters on the monitoring quality and the costs, we presented a detailed view on an individual simulation run. We identified that the tree is robust against churn and the stabilization approach of SkyEye.KOM, using only the metric updates and metrics ACKs remains intact even in the presence of strong churn. We analyzed the monitoring quality of the normal, the synchronized and the smoothing-based approach in detail and affirmed the issue of outliers due to churn in the normal mode. With smoothing, this issue is resolved at the costs of a delayed monitoring view. The outliers typically only occur during calculating the sum of a metric, which is obvious in the case of the peer count. Regarding calculating the average, minimum or maximum of a value the impact of churn is much smaller. We have shown this at the example of monitoring the reference signals Sine and ZigZag as well as monitoring the traffic and message overhead related cost metrics, which are very precise and only delay by around 200s.

We evaluated the quality of the capacity-based peer search and the effects of Support Peers in the tree. Through Support Peers, the monitoring tree is strengthened and more attribute entries can be obtained in higher levels of the tree. The queries are resolved in average in 4 hops and contain only a very small fraction of invalid results. We analyzed the position of query resolving both, in relation to the position of the query initiating peer as well as in relation to the difficulty of the query. Difficult queries are solved with higher probability in the top levels of the tree, simple queries are solved along the way. The simulation results match the results of the analytical model and affirm the deep understanding the behavior of the protocol.

The simulation also gave us a detailed insight on the costs that occur through operating SkyEye.KOM. Due to the fact that the sent metric update messages and received metric update ACK messages are emitted in a dependable regularity, the corresponding traffic is independent of the network size or the level of the peer in the tree. However, regarding the gathering of peer-specific information, we affirmed the observation of the analytical model that the traffic overhead grows exponentially with every level towards the root. For this case, we introduced the maximum threshold for attribute entries, every peer is willing to handle as well as the Support Peers which are picked according to their potential to support the Coordinators. No peer in the network is overloaded due to considering the threshold, while the quality and amount of gathered peer-specific information is high in the tree due to the Support Peers.

As a final step, we validated the results from both, the analytical model as well as the large-scale simulations with a testbed evaluation with up to 500 instances of SkyEye.KOM on top of FreePastry on 37 PCs. In specific, we focused on the tree characteristics and the peer- and level-specific load. The monitoring quality was identified as very precise and the traffic overhead as predicted by the model.

To conclude, we evaluated the monitoring solution SkyEye.KOM in detail, through analytical modeling, simulations and a testbed evaluation. The corresponding results match and validate each other. Through the deep understanding of the behavior of SkyEye.KOM, in specific of the costs that it generates, we are able to use it in a predictable manner for the management of p2p systems.

Part III

MANAGING PEER-TO-PEER SYSTEMS

In this part, we address the issue of unreliable quality of service provisioning in p2p systems. In Chapter 5 we provide a mechanism, termed P^3R^3O.KOM, for reliable resource reservation in unreliable p2p systems. Here in specific, we propose an approach to overcome challenges of long-term resource provision in p2p systems due to the limitation of the peer lifetimes. With the coordinated and managed distributed resource reservation scheme proposed in Chapter 5 we enable p2p systems to act as reliable resource platforms for various kinds of mechanisms and applications. A second limitation on the quality of service provisioning of p2p systems we address in Chapter 6. Here, we motivate and investigate the possible approaches to provide a coordinated control of the quality of service provided distributedly through the p2p system. We propose a quality management framework, termed SkyNet.KOM, which implements the autonomic computing cycle and enables a self-monitoring and self-reconfiguring quality of service management cycle for p2p systems. Both approaches, for controlling the quality of resource provision in p2p systems as well as the quality of service provided by the p2p system itself are thoroughly evaluated in simulations. The evaluation shows that with regard to the proposed resource reservation approach long-term reservations of infinite length can be guaranteed with very low traffic overhead as well as using SkyNet.KOM, p2p systems can be automatically adapted to reach and hold preset quality goals.

5 MANAGING RESOURCE RESERVATIONS IN PEER-TO-PEER SYSTEMS

Seek not, my soul, the life of the immortals; but enjoy to the full the resources that are within thy reach.
- Pindar

Man has lost the capacity to foresee and to forestall. He will end by destroying the earth.
- Albert Schweitzer

In p2p systems, the participating peers organize themselves in a p2p overlay and create an infrastructure to provide functionality for a desired application. Following the p2p paradigm, a peer is both consumer and provider of services and resources. Resources are typically addressing the storage space on peers in order to put data distributedly in the network. Various other resources may be shared and used in the p2p system as well. The CPU power of the peers could be utilized for distributed computations, the bandwidth and main memory for quick distribution of data and the online time for protocol purposes.

P2P systems are also characterized by the unreliability of the participating peers, thus the allocated resources and deployed services are unreliable as well. This issue is even more critical, as long-term resource reservations which last longer than a typical lifetime of a peer cannot be served reliably within a p2p overlay. However, for high quality services and applications based on the p2p paradigm, a reliable resource reservation is needed.

5.1 MOTIVATION

In this chapter, we motivate, present and evaluate a solution for providing long-term service level agreements for reliable resource reservation in unreliable p2p systems. The mechanism provides reservation on a set on resource requirements, a reservation time, and a degree of redundancy. Every resource reservation is managed by a small set of peers in the structured p2p overlay, which acts as a self-monitoring service management group. This group picks peers providing the desired functionality using the capacity-based peer search in SkyEye.KOM in a quantity that the probability of all of them failing is very low. Through the redundancy of the resource allocation and the managed re-nomination of failed resource providers, the desired long-term reservation requirement is met. Evaluation shows that with reasonable redundancy and low service maintenance overhead a reservation fulfillment ratio of nearly 100% can be reached.

Next, we present the functional and non-functional requirements for providing reliable resource reservation in structured p2p systems. Subsequently, we present our solution for solving this problem in Section 5.2. This chapter closes with an evaluation of the proposed resource reservation mechanism in Section 5.3, discussion of related work in Section 5.4 and a summary in Section 5.5.

5.1.1 *Functional Requirements*

One main characteristic of p2p systems is the unreliability of the peers. They come online and go offline autonomously at their will. However, the applications and services on top of the unreliable infrastructure require a reliable layer on which they can operate. Reliable resource reservation allows the higher layers to reserve a given number of peers with desired capacities (or attributes) for a dedicated time. The reservation time is expected to be longer than the lifetime of the peers, for example, some weeks or months. We depict the goal of the reservation management solution in Figure 66 and discuss it in the following.

Let $Cons_i$ be a constraint on the attributes Att_i of the peers in terms of an upper or lower bound. The functionality of capacity-based peer search is defined as:

- PeerID-list $\leftarrow$ capacity-based-peer-search(n, $Cons_1$, $Cons_2$,...,$Cons_i$) - query for n peers fulfilling a set of constraints

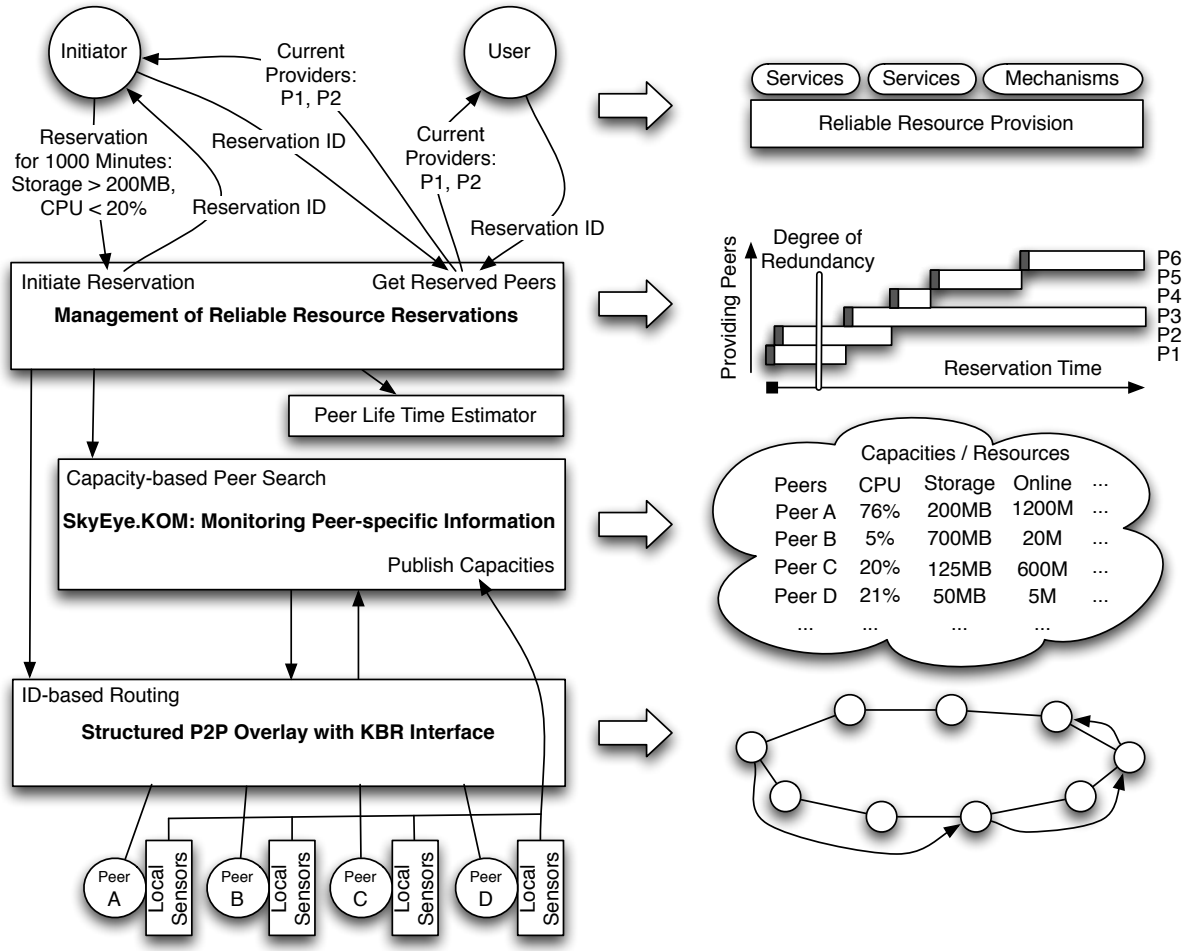


Figure 66: Reliable Resource Reservation enabled through several Mechanisms

A resource reservation extends the function of capacity-based peer search by the component of reliability and reservation time:

- Resource constraints, $Cons_i$: The amount of resources to allocate, such as 500 MB storage space and an upload bandwidth of 128 KB/s.
- Reservation time, $RsvTime \in T$: The time period for which the resources should be allocated, such as 200 days.
- Degree of redundancy, $RsvDR \in [0;1]$: A ratio of redundancy for the resource reservation. It describes the probability that should be maintained that not all of the resource providers fail within a short time span.

The goal of the mechanism for reliable resource reservation is to provide the functionality to reserve resources for a given time:

- Reservation ID $\leftarrow$ initiateReservation($Cons_1$, $Cons_2$,...,$Cons_i$, $RsvTime$, $RsvDR$) - reserves a peer set with given capacities for a time, $RsvTime$, and a given redundancy degree, $RsvDR$.
- PeerID-list $\leftarrow$ getReservedPeers($ReservationID$) - returns the list of the peers providing the reserved resources at the current time.

The mechanism, providing reliable resource reservations, takes care that the desired amount of capacities is constantly provided by the system during the reservation time. Please note that with reliable resource reservation we do not require that the resources are provided by the same peers over

the whole time. In contrary, we assume that the services and task deployed and allocated on the reserved resources are portable and may be placed on other peers on demand. We assume an explicit relocation time buffer which is needed to port the services and tasks from one peer to another. Based on the list of peers providing the desired capacities, a reliable service can be established.

In the following, we discuss the non-functional requirements on the mechanisms for reliable quality and resource reservation in p2p systems.

5.1.2 *Non-functional Requirements*

After the introduction of the functional requirements on a solution, we discuss the non-functional requirements for the problem statement.

Reliability
Reliable resource provisioning involves the maintenance of a reservation status over a long period of time. We require that the proposed solution takes care that the resource reservation is fulfilled and the desired degree of redundancy is held.

Scalability
For a mechanism for reliable resource reservation, we state the non-functional requirement that the number of peers in the network does not affect the reservation costs. Regarding the constraints and reservation time, we assume reasonable requests of the participating peers, as naturally an increased reservation time and constraints set lead also to increased reservation overhead. To conclude, the traffic costs generated by the reservation maintenance should not be influenced by the number of peers in the network, but only by the reservation time as well as toughness and number of the resource constraints. We assume the existence of a mechanism for capacity-based peer search, which may create traffic overhead in the p2p system as well.

Efficiency
The main performance indicator for the reliable resource reservation is the success ratio of the reservation. A 100% success ratio for all reservations is aimed, while providing the desired capacities. However, a high success ratio requires also a many redundant resource allocations for the reservation, leading to many capacities blocked in the system. An efficient mechanism for reliable resource reservation provides the desired reservation with as few peers involved in the reservation as possible. In order to allow users to define their desired degree of redundancy, the parameter RsvDR may be set more modestly. Another aspect of efficiency is the response time for resource reservations. A reservation request should be answered in a short time.

Stability
One main aspect of the reservation approaches is to stabilize the resource provisioning in the presence of churn. Churn or more generally varying user behavior introduces dynamics in the p2p system, affecting the resources provided for the system. In order to counteract these dynamics and to provide a stable resource offer, the proposed mechanism is in place. A non-functional requirement on the stability of the proposed solutions is that it copes with frequent and drastic changes in the number of peers in the system and keeps on providing the desired resource reservation.

Consistency
For a mechanism for reliable resource reservation, a consistent approach takes care that the same capacities of peers are not allocated for various reservation requests at the same time. Additionally, all peers should operate on the same set of resources, thus having the same view on the resources in the p2p network.

In the following, we discuss the challenges and design decisions made for the mechanism for reliable resource reservation. After that, we present the assumptions for the solutions and the components it consists of. This overview is followed by the description of the protocols for managing and enforcing reliable long-term resource reservations.

5.1.3 *Design Decisions*

Next, we discuss the design decisions for reliable resource reservation in p2p systems.

HOMOGENEOUS VS. HETEROGENEOUS ROLES

One single peer in general cannot provide the desired resources over the reservation time, thus the resource provision has to be distributed over a set of peers. A design decision to make is, how the peers should cooperate to provide the resources. The resource provision can be subdivided into two tasks: managing (and supervising) the reservation and providing the desired resources. On the one hand, these two tasks could be provided by the same set of peers. However, the requirements for both tasks are different, i.e. the one requires long-living peers for supervising the reservation and the other task requires peers with the requested resources. In our approach, we separate the functionality and assign them to different peer sets. For the reservation maintenance, peers are needed which stay online for a long time. They supervise the reservation status and success probability and pick peers for the resource provision.

ORGANIZATION OF THE RESERVATION MANAGERS

Both for the reservation managing peer and the resource providing peers we need to add backup peers to address churn in the network. The reservation managers supervise the reservation providing peers and control the redundancy by adding more peers to the set of resource providers in case of churn. However, the task of reservation management requires also redundant management peers to cope with churn. This set of management peers decide on the number of resource providing peers distributedly. A *quorum-based* and *token-based* approach could be used. In the token-based model, one single peer in the group will alone decide on the resource providers. The other peers act as backup for the case that the deciding peer fails. In a quorum-based approach, like in [LHK+06], the management peers decide on the number of resource providers through a distributed voting scheme, which may be secured as in [KHS09]. However, this requires the synchronization of the peers and leads to additional overhead without functional benefits. In our solution, we follow the token-based approach.

FUNCTIONAL DECOMPOSITION

The task of managing and enforcing resource reservations is composed of several sub-tasks. One prominent among them is the task of capacity-based peer search. A solution for the problem of resource reservation may solve this sub-task as well or require a functional component providing this service. For the clarity of the solution and the extendability of it, we use SkyEye.KOM providing this service. By this, both the resource reservation scheme and the capacity-based peer search functionality may be optimized independently.

LEVEL OF AVAILABILITY

The reserved amount of resources should be available all the time. Thus, a mechanism that only reacts upon a violation of the reservation is unacceptable. We need a mechanism that also takes care that the resource reservation is not endangered to fail. For that, the reservation contains a parameter for the success probability. Although it is requested to fulfill 100% of the reservations in the system, the degree of availability and effort may vary. The degree of redundancy, $RsvDR$, defines how aggressively this goal of providing the reservation is followed, i.e. with which probability all reserved peers are expected to stay alive within a short time span. Thus, besides the main objective of providing the resource reservation by 100%, we also follow the subordinated objective of keeping the probability of not failing above the success probability parameter of the reservation, $RsvDR$.

In order to do so, the mechanism may follow two approaches: either considering or not considering the status of the resource providing peers. Focusing only on the number of resource providers is easier to implement, but may lead to higher resource provision costs as the mechanism would assume the worst case for the remaining lifetime of the peers. Using an estimation on the lifetime of the peers results in higher success probabilities for the resource provision with the same number of peers. This is due to the more valid lifetime estimations than a worst case assumption. For this purpose, our proposed mechanism includes and uses an approach for estimating the lifetime of the peers.

5.1.4 *Assumptions*

Having discussed the functional and non-functional requirements for a mechanism for reliable resource reservation, we now discuss the assumptions stated for our approach. In order to provide the desired functionality of reliable long-term resource reservation, we require following mechanisms:

- A *structured p2p overlay* allows to assign roles and data to given IDs in the p2p overlay. With the introduction of roles, resource reservations may be stored, managed and enforced in a deterministic and reliable manner. The structured p2p overlay complies with the KBR interface as described in [DZD+03] and Section 3.1.
- The functionality of *capacity-based peer search* allows for identifying a desired set of peers with required capacities. With this tool at hand, resource reservations can be conveniently enforced by picking suitable peers to provide the resources. We use the functionality of SkyEye.KOM as described in Section 3.5. The assumption of a *structured p2p overlay* is also made for the monitoring mechanism, SkyEye.KOM. Here, we assume that the common API for structure p2p overlays, as described in [DZD+03] is given.
- A *function to estimate the peer lifetime* based on its current lifetime is needed in order to prepare for peer failures and churn. We describe this functionality in the following.

The peer lifetime estimator is a function that gives the probability $P_{fail}(p, t_{online}(p), t_R)$ for a peer p with current lifetime of $t_{online}(p)$ that the peer p will fail in the upcoming time period of length t_R. We introduce a short form for P_{fail}, as the second parameter is always depending on the first:

$$\begin{aligned} &t_{online}(p) : P \to T \\ &P_{fail}(p, t_{online}(p), t_R) : P \times T \times T \to [0,1] \\ &P_{fail}(p, t_R) : P \times T \to [0,1] := P_{fail}(p, t_{online}(p), t_R) \end{aligned} \tag{5.1}$$

For the determination of the remaining lifetime of a peer, based on its current lifetime, various approaches exist. In [SENB07] the author presents the results of having crawled continuously the KAD network for about 6 months. The report regards the geographical distribution of peers, session times, peer availability and peer lifetime. We modeled the lifetime of the peers, based on the churn behavior of the peers as it was measured in the KAD network. The measurement results show that the peer lifetime is Weibull distributed with following parameters:

$$\begin{gathered} mean = 266.5358,\ standard\ deviation = 671.5063, \\ scale = 169.5385,\ shape = 0.61511 \end{gathered}$$

One important result of the study is that a long peer uptime leads to a higher probability of staying online than a short uptime. As a result, the probability for staying online increases with every minute a peer stays online.

Let $F(t, k, \lambda)$ be the cumulative distribution function for the Weibull distribution with Weibull parameters for scale, k, and shape, λ. It gives for a given time span t the probability of being offline. Thus, we can calculate for all peers $p \in P$ the probability to fail in the next t_R minutes based on the their lifetime $t_{online}(p)$.

The probability $P_{fail}(p, t_R)$ that a peer p will go offline in the next t_R minutes (e.g. $t_R = 10$ minutes) can be calculated according to Equation 5.1 and $F(t, k, \lambda)$ as

$$P_{fail}(p, t_R) = \frac{F(t_{online}(p) + 10, k, \lambda) - F(t_{online}(p), k, \lambda)}{1 - F(t_{online}(p), k, \lambda)} \tag{5.2}$$

With this assumptions in mind, next, we present our solution for reliable long-term resource reservation.

5.2 A SOLUTION FOR RELIABLE LONG-TERM RESOURCE RESERVATION

The goal of the mechanism for reliable resource reservation is to provide the functionality to reserve and provide resources for a given time. We propose P^{3}R^{3}O.KOM, a p2p protocol for reliable resource

reservation and offering. It offers reservations with specified resource constraints, a reservation time and a degree of redundancy as stated in Section 5.1 as functional requirements:

In the following, we describe the components of our mechanism and subsequently the protocols for initiating and maintaining reservations as well as for enforcing these reservations.

The main challenge for reliable resource reservation is to reserve the desired amount of resources for a long time, which is expected to exceed the lifetime of each peer in the network and especially the lifetime of the resource reserving peer. However, the resource provision must be enforced and reliably fulfilled. Churn is the main threat for long-term resource reservations. In order to provide the desired resources with a specific availability, resources are allocated and provided redundantly. In this case, individual peer failures are not critical as other peers in the resource providers' set remain active. For the resource reservation and provision process, two steps are conceptualized:

- *Reservation Management*: A set of peers supervises the status of the resource provision, i.e. the set of resource providing peers, and adapts the redundancy level in this set in case of churn. For that, the reservation management peers predict the lifetime of the current resource providers and estimate the success ratio for at least one of the peers staying alive until the next round. These peers continuously manage the set of resource providing peers.
- *Reservation Enforcement*: A set of peers provides redundantly the desired resources for the given reservation. The peers all fulfill the resource requirements stated in the resource reservation.

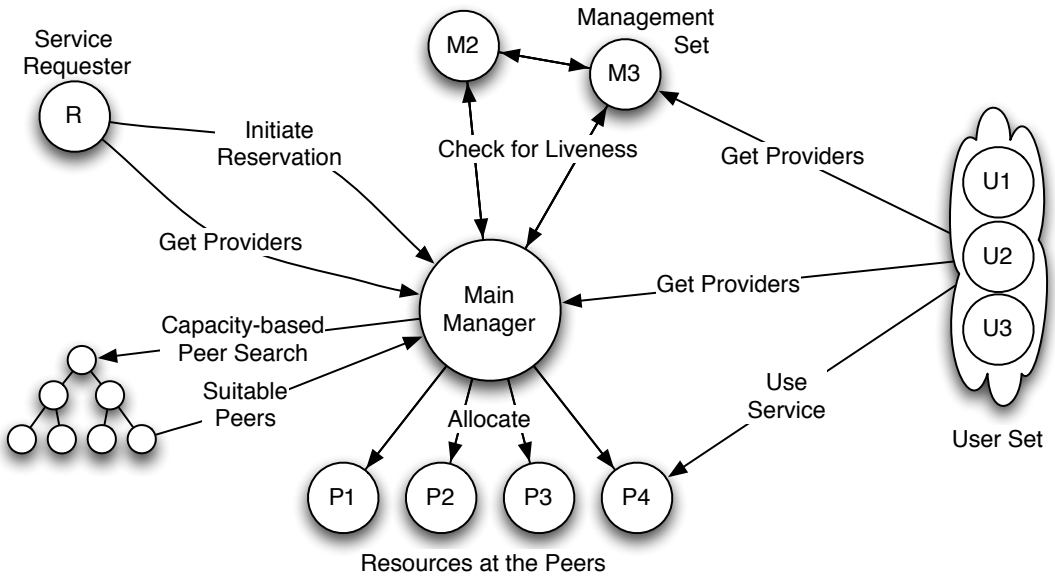


Figure 67: Components Interaction Overview in P^{3}R^{3}O.KOM

In the following, we describe the protocol used to initiate, maintain and enforce a resource reservation: An overview on the components is given in Figure 67.

SETTING UP A RESERVATION

A peer in the network creates a reservation request describing the resource constraints, reservation time and degree of redundancy. It also derives locally a corresponding Reservation ID, for example, the hash of the reservation parameters. This ID is mapped to the peer responsible for the specific ID in the structured p2p overlay. The resource reservation is stored on this peer in a reliable manner.

This peer additionally adds further information to the reservation request. It adds the initiation time, reservation starting time, reservation finishing time and a list of variable information that change over the time of the reservation. The variable part contains information about the current reservation status, i.e. the probability to succeed and a list of the current resource providers. The peer responsible for the Reservation ID is the first peer in the management set for this reservation and termed *main manager*.

MAINTAINING THE MANAGEMENT SET

One of the two tasks of the management set is to store and manage the status of the reservation. In order to do so, the reservation information must be stored reliably and must be always up to date, even in the case of the leave of the main maintainer. We add redundancy to the reservation management set for the case, that the main maintainer leaves the network. In that case, the new peer responsible for the Reservation ID must take over the management role for the reservation. The main manager makes sure that it has a fixed set of backup peers (e.g. 2) that keep a local copy of the reservation status and update a new main manager in case of churn.

In this step, the main manager checks periodically the liveness of the backup peers or chooses new ones. New management peers are chosen using the function of capacity-based peer search and looking for peers that promise to stay long online, by already having a long online time. Also the backup peers check the liveness of the main manager and update the new peer responsible for the Reservation ID in case of churn.

DERIVING THE SUCCESS PROBABILITY FOR A ROUND

The second task of the management set is to provide the functionality of *PeerID-list* ← *getReserved-Peers(*ReservationID*)*. The peer list contains peers providing resources for this specific reservation. Thus, the main manager must take care that enough peers are allocated, so that the desired redundancy degree, RsvDR, is reached.

Periodically, following steps are done by the main reservation manager in a round-based manner: As a first step, the main manager checks the liveness of the resource providers in order to derive an up-to-date list of resource providers. Next, it calculates based on the current lifetime of the peers the probability that the current resource providers will be online until the next round. For that it uses the function to estimate the peer lifetime in order to retrieve a probability for liveness in the next round. The product of these probabilities tells the success probability to provide the reservation until the next round, it is equal to the probability of at least one peer staying alive.

According to Equation 5.2, we can calculate the probability of one single peer to fail in a given time span t_R. Now, we evaluate the probability for a set of peers to fail. Let $P_i \subseteq P$ be a subset of the peers with cardinality i. The probability $P_{fail}(P_i, t_R)$ denotes the probability that all of the i peers fail in the next t_R minutes. We can calculate this probability as:

$$P_{fail}(P_i, t_R) = \prod_{p \in P_i} P_{fail}(p, t_R) \tag{5.3}$$

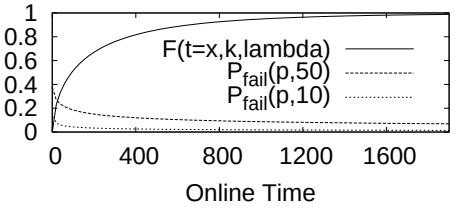
The probability for successfully "surviving" a time span of t_R with the given set P_i of i resource providers is:

$$P_{succ}(P_i, t_R) = 1 - P_{fail}(P_i, t_R) \tag{5.4}$$

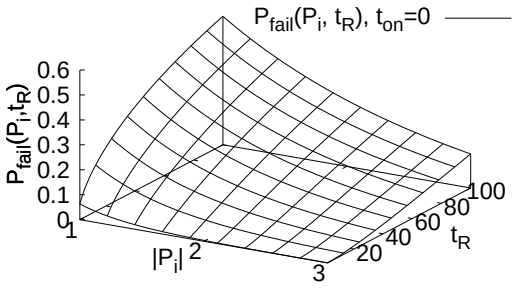
Thus, the mechanism for reliable resource reservation must aim at keeping the number of resource providing peers, i, so high that the probability $P_{succ}(P_i, t_R)$ is above the degree of redundancy RsvDR. We depict in Figure 68a the distribution $F(t, k, \lambda)$ and $P_{fail}(p, t_R)$ with $t_R = 50$ and $t_R = 10$. In Figure 68b, we depict $P_{fail}(P_i, t_R)$ with $t_{on} = 0$ and varying $\|P_i\|$ and t_R. The figure visualizes the lifetime estimation both for a single and group of peers.

CHECKING THE RESOURCE PROVIDER SET

The success probability $P_{succ}(P_i, t_R)$ is matched to the degree of redundancy required in the resource reservation, RsvDR. In order to not run out of resource providers and fail the whole reservation, the mechanism aims at increasing the actual success probability of the reservation above the required degree of redundancy specified in the reservation. In case of an endangered success probability, the main manager searches for peers to add to the resource provider set. It picks suitable peers for the reservation in a quantity that pushes the expected success probability above the degree of redundancy specified in the reservation. In order to find suitable peers, it uses the capacity-based peer search functionality provided by SkyEye.KOM. For that, two sub-tasks need to be solved. First, identifying the number of additionally needed peers and second, picking suitable peers in this magnitude.



(a) Peer Lifetime Estimation



(b) Peer Failure Probability

Figure 68: Peer Lifetime and Failure Probability based on KAD Measurements

IDENTIFYING THE NUMBER OF MISSING RESOURCE PROVIDERS

The amount of peers to be added to the resource provider set is considered to keep the success probability $P_{succ}(P_i, t_R)$ for the next time span t_R above the degree of redundancy, RsvDR, as specified in the reservation request. The variable $P_{succ}(P_i, t_R)$ is the probability that at least one of i peers in the set of resource providing peers, P_i, stays alive for a round of length t_R.

Let $Num_{exact} : [0, 1] \to \mathbb{N}$ be a function calculating for a given success probability RsvDR the number of peers needed, so that $P_{succ}(P_{Num_{exact}}, t_R) = RsvDR$. As this number of peers is calculated exactly, in case of churn, the probability for all peers staying alive in the round drops below the probability RsvDR. In order to not fall below the threshold of RsvDR, we present two possible approaches for deriving the desired quantity of resource providing peers Num_{suff}, with $P_{succ}(P_{Num_{suff}}, t_R) \geqslant RsvDR_{new}$, even under churn.

Probability Buffer Assignment (PBA)

We pick the number of resource providers with a slightly increased degree of redundancy, i.e. by adding a probability buffer of $P_{buff} \in [0, 1]$. Thus, we try to pick an amount of peers, so that a higher aliveness probability is reached than RsvDR and there is a buffer to address the case of churn:

$$Num_{suff}^{PBA} = Num_{exact}(RsvDR + P_{buff} \cdot (1 - RsvDR)) \tag{5.5}$$

As an example, we choose $P_{buff} = 20\%$ and the degree of redundancy $RsvDR = 90\%$. In that case Num_{exact} results in the number of peers so that the probability of at least one of them surviving is $RsvDR = 90\%$. The variable Num_{suff}^{PBA} results in the number of peers so that the probability of at least one peer surviving is $RsvDR + P_{buff} \cdot (1 - RsvDR)$, thus $90\% + 0.2 \cdot 10\% = 91\%$.

Redundant Provider Assignment (RPA)

The redundant peer assignment approach adds to the exact number of peers required to meet the degree of redundancy an additional amount of peers, N_{buff}. For that, it assumes that the N_{buff} peers, that make the most considerable contribution to the resource reservation, will leave and need to be replaced before the next round begins. Thus, the resulting number of peers needed is:

$$Num_{suff}^{RPA} = Num_{exact}(RsvDR) + N_{buff} \tag{5.6}$$

As an example, we choose $N_{buff} = 3$ and the degree of redundancy $RsvDR = 90\%$, let the resulting number of peers needed be $Num_{exact}(90\%) = 2$ $Num_{suff}^{RPA} = Num_{exact}(RsvDR) + 3 = 5$.

PICKING NEW RESOURCE PROVIDERS

In order to add new peers to the resource provider set, the main manager uses the function for capacity-based peer search of SkyEye.KOM. It defines the desired capacities and requests an amount of peers which pushes the probability of at least one peer staying alive until the next round above the desired

threshold for the degree of redundancy. SkyEye.KOM provides a list of Peer IDs with suitable capacities, as described in Section 3.5. Additionally, the attribute entries of the new peers give information on the online time of the picked peers.

The main manager contacts the new peers in order to allocate their resources for the given reservation. The contacted peers allocate the requested resources for the reservation and acknowledge the allocation to the main manager. We also consider during the picking of new peers, a specific time to deploy and start a given service on the allocated resources. Thus, new peers are not instantly used in the resource reservation, but only after a short time.

PREPARING THE NEXT ROUND

The steps mentioned above are repeated periodically in a round-based manner. In every round the main manager updates the management set as well as the set of resource providers. The set of reserved peers is now up to date and the function `PeerID-list` ← `getReservedPeers(`*ReservationID*`)` is served by the main manager, which is identifiable by the Reservation ID. As the prediction on the lifetime of the resource providers is only precise for a short time, the main manager repeats this step every round.

The round length, t_R, has a great influence on the success of a reservation and the costs induced by the reservation management. A high reservation sampling time results in long intervals of unobserved resource provision. In this period, to many resource providers may leave, thus leading to too less resource providers for the desired degree of redundancy. On the other hand, too frequent sampling of the resource provider set leads to a quick adaptation of the set size to the desired degree of redundancy. It also leads to an increased traffic overhead for reservation maintenance as the validity of the resource provider and management set is checked in every round.

Having presented our approach for providing reliable resource reservation in structured p2p systems, next, we discuss the performance and cost of the solution.

5.3 EVALUATION

The goal of the evaluation is to measure the functionality, performance and cost of the proposed mechanism for reliable resource reservation, P^3R^3O.KOM. We discuss the influence of the configuration parameters for both approaches, the PBA approach and the RPA approach.

For the evaluation, we first introduce the relevant metrics in Subsection 5.3.1 and the simulation setup in Subsection 5.3.2. In Subsection 5.3.3, we present the evaluation results, discuss related work in Section 5.4 and draw conclusions in Section 5.5. Specifically, we are interested whether the proposed approaches fulfill the desired resource reservations and how the parameters in PBA and RPA take influence on the reservation provision quality.

5.3.1 *Metrics*

In the following, we list the metrics to validate the functionality of reliable resource reservation and to measure the performance and cost of the approach.

The metrics regarding the performance of the reliable resource reservation mechanism are:

- Redundancy Success Ratio
  The ratio of services which succeeded in providing the desired resources for the desired time and also did not violate the threshold given by the degree of redundancy (RsvDR) of the reservation.
- Reservation Success Ratio
  The ratio of reservations which were successfully provided during the reservation time.
- Failure Ratio
  The fraction of reservations which ran out of resource providers during the reservation time.

Correspondingly, the metrics regarding the costs for maintaining and providing the reservations are:

- Total Number of Providers
  The total number of resource providing peers to accomplish a resource reservation.

General		**Network and User**	
Simulation time	5000 min	Churn Model	KAD [SENB07]
Number of nodes, N	1k,2k,4k	Average online time, t_{on}	266 min
Number of Reservations	5% of N	Reservation Time $Rsv Time$	3000 min
SkyEye.KOM		**Reservation Management**	
Metric UI	30 s	Round time, t_R	50 min
Attribute UI	60 s	Management backup	4
Tree degree	4	Backup checking frequency	10 min
Reservation Complexity		**Approaches**	
Service complexity	medium	PBA (* 10%)	1,3,5,7,9
Degree of Redundancy	0.9	RPA (+ peers)	1,2,3,4,5

Table 16: Setup for the Simulation of P^{3}R^{3}O.KOM

- Average Number of Providers
  The average number of peers providing resources during the resource reservation phase. This metric sets the total time for resource provisioning of all involved peers in relation to the total reservation time. The resource provision costs is always greater or equal to 1 for the case that the reservation was successful. The smaller the resource provision cost is, the less resources are wasted and the mechanism is more efficient. The variable $RsvTime_r \in T$ denoted the time period for which the resources should be reserved of the Reservation ID r. Let $P^r_{prov} = \{p^r_1, p^r_2, ..., p^r_m\}$ be the set of peers, which provided their resources for the reservation r. Let further be $T_{prov} : P_{prov} \to \mathbb{R}$ a function mapping each resource provider to the time it provided resources for the given reservation. Then the reservation provision cost $RPC : r \to \mathbb{R}$ can be calculated as

$$RPC(r) = \frac{\sum_{i=1}^{m}(T_{prov}(p^r_i))}{RsvTime_r} \tag{5.7}$$

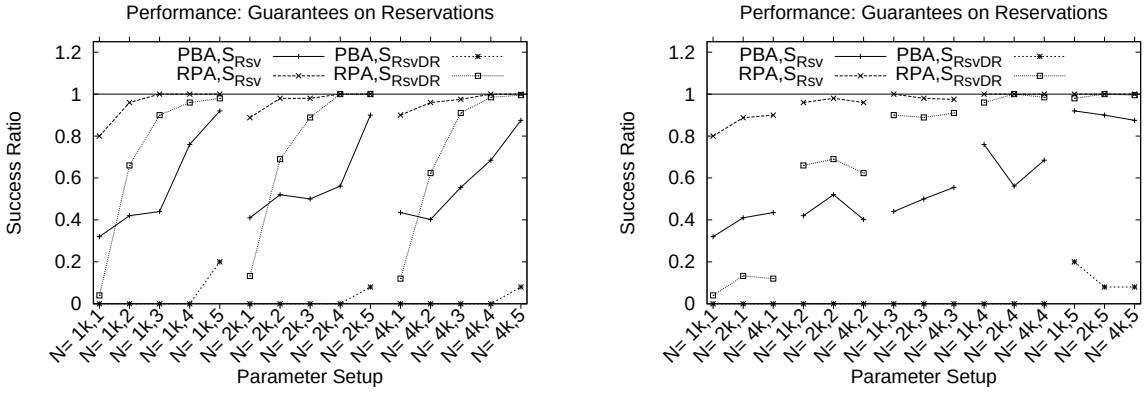
- Traffic Overhead per Reservation
  This metric measures the reservation maintenance overhead terms of traffic.

- Total Queries per Reservation
  The total number of queries for capacity-based peer search used to complete a reservation.

5.3.2 *Simulation Setup and Workload*

We simulated our approach on PeerfactSim.KOM [KKM+07], which allows for the simulation of layer-based p2p systems. The simulator has also been used for publications at IEEE P2P in the previous years as in [KLS07, KLKP08, GSR+09]. We introduced the simulator in more detail in Section 4.2. As a workload model we use a churn model based on KAD measurements [SENB07] and the underlay model uses measurement data on real-world round-trip times [NZ02]. In Table 16, we depict the setup for the evaluation. We simulated a p2p system with 1000, 2000 and 4000 peers with KAD churn for 5000 minutes. We varied the number of nodes to observe the impact of the scale of the network on the reservation results. First all peers join the idealized DHT overlay, CDHT, which offers the KBR-functionality [DZD+03]. It provides reliable and consistent ID-based routing in the presence of churn. A fraction of 5% of the peers initiate, after joining of the p2p overlay, a reservation request with a reservation time of 3000 minutes. This results in 50 reservations in the network with 1000 peers, 100 reservations with $N = 2000$ and 200 reservations with $N = 4000$. Specifically, we analyzed long-term reservations which had a reservation time significantly longer than the average online time of a peer (266 minutes). We used $t_R = 50m$ and $RsvDR = 0.9$, meaning that P^{3}R^{3}O.KOM takes care that the probability for at least one resource providing peer to survive a period of 50 minutes is higher than 90%.

All peers join into the network during the first 100 minutes. The second part of simulation begins right after the stabilization phase finishes, from minute 100 to the end of simulation. Directly after minute 100, services are requested and maintained. Additionally, churn is induced using the KAD churn



(a) Success Ratio of Reservations

(b) Success Ratio of Reservations

Figure 69: Performance of Reservation Maintenance

behavior. KAD churn introduces both joining and failing peers. Due to the larger rate of failures, the number of peers decreases over a time of 50 hours to 50% of the initial peer count. In the evaluation results, we describe the success ratio and costs of all services over the whole time.

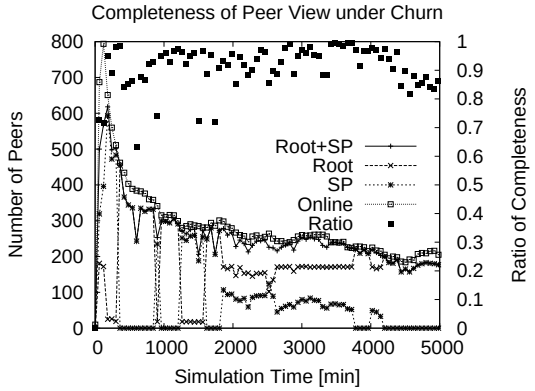
5.3.3 *Evaluation Results*

We show the evolving of the number of peers in the p2p network under churn in the setups with $N = 1000$, $N = 2000$ and $N = 4000$ in the Figures 70a, 70c and 70e. In these figures, we also depict the number of attribute entries at the root and its Support Peer which are available to use for the capacity-based peer search. We observe that although the root is sometimes weak and only able to bear a small amount of attribute entries, in combination with its Support Peers the root is able to monitor more than 80% of the peers in the network for most of the time.

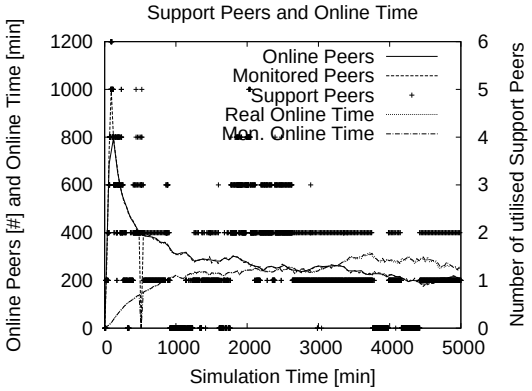
In Figures 70b, 70d and 70f, we show the trace of the number of utilized Support Peers in the SkyEye.KOM tree, the real and monitored peer count as well as average online time of the peers. The graphs show the monitoring view of every 60th measurement, i.e. one hour, for clarity of presentation. In a graph with every monitoring snapshots a lot more positions exist, where the monitoring view drops to 0 due to the reassignment of the root position under churn. However, considering only the snapshot of every hour, the monitoring view is very precise. The figure depicts on the one hand the bound online time of the peers. The network and churn model we used, inherits the characteristics of KAD churn and resembles and average online time of 266 minutes per peer. On the other hand, it shows the dynamic allocation of Support Peers in the tree. Through the utilization of Support Peers, strong peers assist in the monitoring duty and no peer is overloaded. For the quality of resource reservations, the completeness of the monitoring view is essential, as P^3R^3O.KOM uses the capacity-based peer search provided by SkyEye.KOM as a basis to find suitable peers for the reservations.

The main metric for the performance of the resource reservation mechanism P^3R^3O.KOM is the reservation success ratio, for the cost it is the number of average reservation providers during the reservation time. First, we discuss the performance of the solution and subsequently the overhead the mechanism. In Figure 69, we depict the performance related metrics of the reservation success.

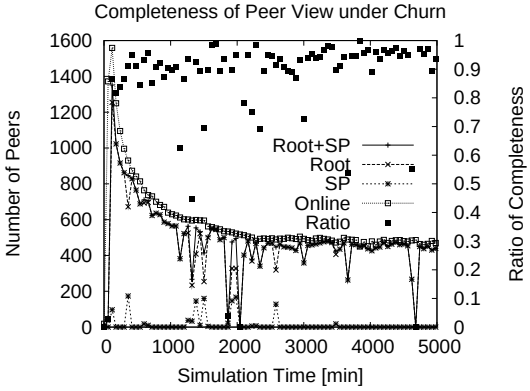
The x-axis lists the various setups for RPA and PBA for a variation of network sizes. Here, the setup number 3, for example, describes the PBA approach with $P_{buff} = 50\%$ and the RPA approach with $N_{buff} = 3$. One main observation is that with increasing setup counter, i.e. with a higher security buffer, the success ratio increases both for the whole resource reservation as well as maintaining the redundancy level. The RPA approach clearly outperforms the PBA approach in terms of reservation success and reaches with $N_{buff} = 4$ and $N_{buff} = 5$ a 100% reservation success ratio. We further notice that the scale of the network size does not affect the reservation quality, in the case that sufficient suitable providers exist.



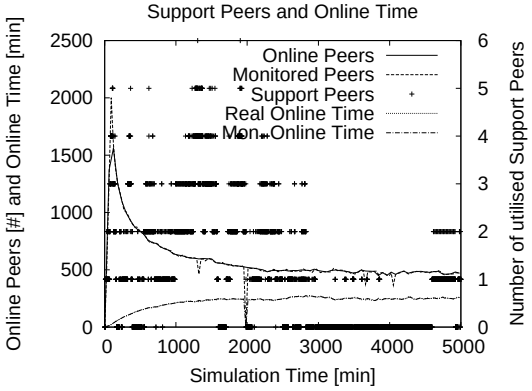
(a) Completeness of Peer View, N=1000



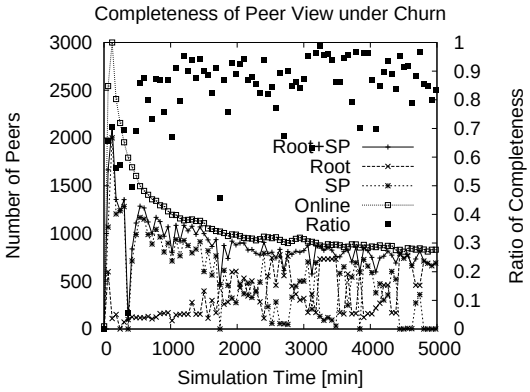
(b) Support Peers and Online Time, N=1000



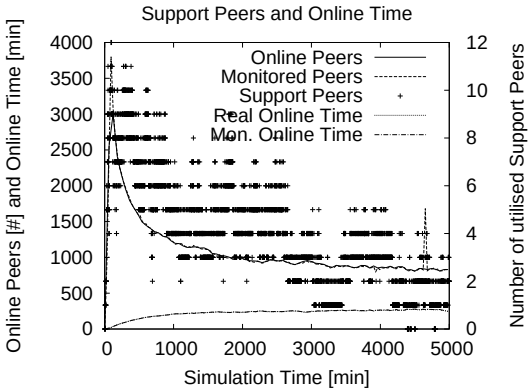
(c) Completeness of Peer View, N=2000



(d) Support Peers and Online Time, N=2000

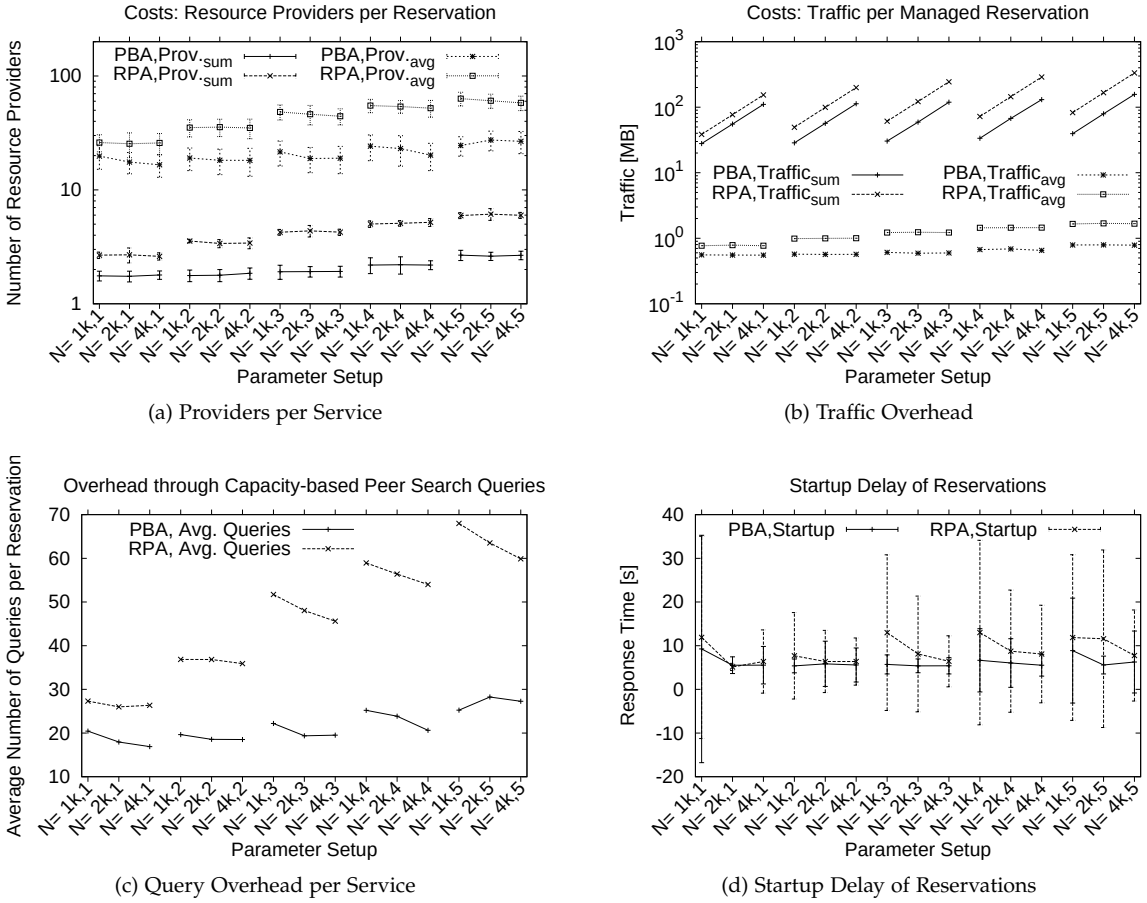


(e) Completeness of Peer View, N=4000



(f) Support Peers and Online Time, N=4000

Figure 70: Peer Count, Completeness of Attribute Information View and Bound Online Time



(a) Providers per Service

(b) Traffic Overhead

(c) Query Overhead per Service

(d) Startup Delay of Reservations

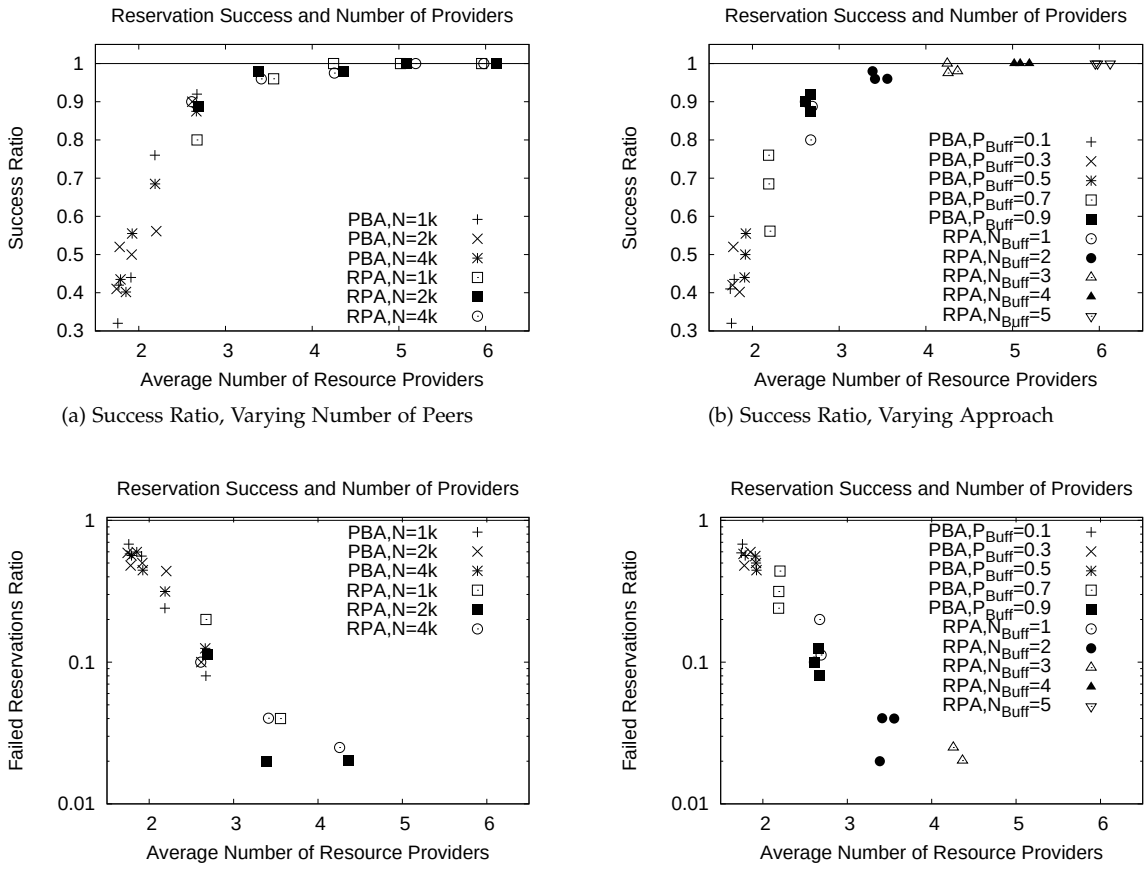
Figure 71: Costs related to the Resource Reservations

In order to maintain the resource reservations, several kinds of overhead occur, which we depict in Figure 71. The main metric depicting the costs, $\mathrm{RPC}(r)$, measures the average number of parallel resource providers for a reservation. As in an optimal case, only one single peer provides the resource for the whole time, we can compare and benchmark the reservation mechanism against this cost metric. The question is, how many resource providers are needed in parallel in order to provide 100% of the resource reservations successfully. Figure 71a presents the total and average number of resource providers for a reservation. We observe that both metrics increase linearly with setups for RPA, i.e. in specific with N_{buff}. The PBA approach on the other hand allocates much less resource providers for the resource provisioning. Due to this, the performance of the approach is also worse. Regarding the total number of providers, we see that for a resource reservation that last for 3000 minutes approximately 50 to 60 peers are used in total to provide all reservations (e.g. RPA with $N_{buff} = 4$). Although this number seems large, it characterizes the main purpose of P^3R^3O.KOM, to allocate resources for long term reservations that cannot be provided by single peers due to churn.

The corresponding traffic for the reservation maintenance is shown in Figure 71b. The maximum average traffic overhead of reservation management is reached with RPA and $N_{buff} = 5$, the average reservation management overhead is in this case 1745 KB over a time period of 3000 minutes, i.e. slightly more than 2 days. In 3000 minutes and a check every $t_R = 50m$, in total 60 checks have been conducted by the main manager, each with 30kb overhead in average of maintenance messaging. This overhead is very low and underlines the practical usability of the approach. The total traffic overhead for the 50, 100 and 200 reservations in total are depicted in Figure 71b as well.

In Figure 71c, we present the capacity-based peer search related queries that are initiated by P^3R^3O.KOM in SkyEye.KOM in order to find suitable peers. The number of queries is directly related to the total number of resource providers per reservation. Periodically in an interval of $t_R = 50m$,

the main manager checks whether the amount of resource providers is sufficient or not. In the case that not enough resource providers are there or that they are expected to leave soon, a capacity-based peer search query is initiated for the missing amount of resource providers. The peers in the reply are instantly allocated for resource provisioning. The startup delay for the reservations is depicted in Figure 71d. It is in average less then 15 seconds for any setup for P^{3}R^{3}O.KOM. The time is used to contact the main manager, which itself emits a query for suitable peers using the capacity-based peer search functionality of SkyEye.KOM. The query traverses the monitoring tree and is replied to the main manager with a set of suitable peers. These queries are in small networks the hardest to solve as less qualified peers are available. However, a reservation setup delay of several seconds is to be seen in relation to the reservation time, which was chosen as 3000 minutes.



(a) Success Ratio, Varying Number of Peers

(b) Success Ratio, Varying Approach

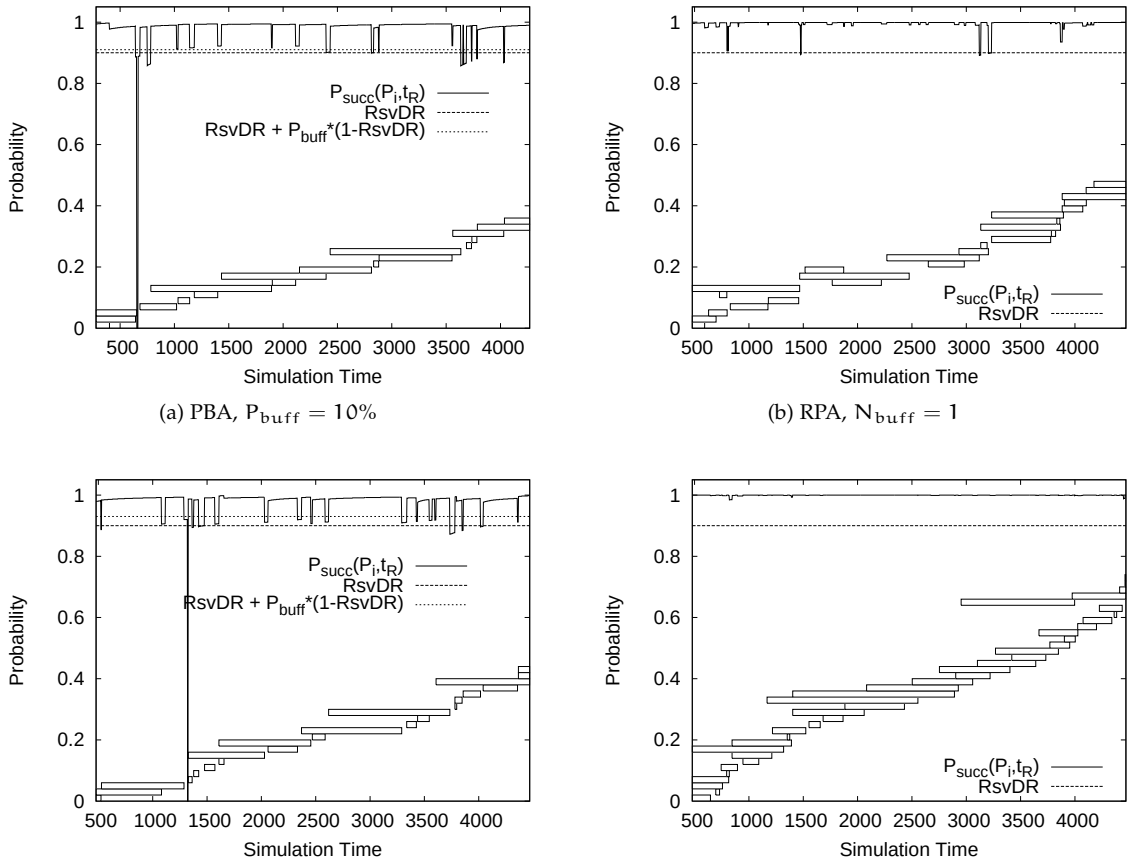
(c) Success Ratio, Varying Number of Peers

(d) Success Ratio, Varying Approach

Figure 72: Success Ratio in relation to Average Number of Resource Providers

Having presented the evaluation results regarding the performance and costs of the approaches RPA and PBA of P^{3}R^{3}O.KOM, next, we investigate in Figure 72 the direct relation between the number of redundant resource providers and the resulting reservation success probability. The metrics regarding the average and total number of providers and traffic overhead are directly linked to the reservation success ratio. The more redundantly the PBA and RPA approaches pick and provide resource providers, the less probable it becomes that the desired degree of redundancy is missed or the whole reservation fails. The PBA approach allocates in maximum 2.67 resource providers which is not sufficient to fulfill all reservation guarantees. The redundant providers approach, RPA, provides with $N_{buff} = 3$ in the network with 1000 a 100% reservation success ratio, but fails in the network with 2000 and 4000 peers in two cases. With $N_{buff} = 4$ and $N_{buff} = 5$, 100% of the reservations are successfully fulfilled. For that, in average 5.10 and 6.02 peers are invoked simultaneously in average for resource provisioning.

To conclude the evaluation on the reservation management solution P^{3}R^{3}O.KOM, we present a series of reservation requests, the corresponding intervals in which resource providers contribute to



(a) PBA, $P_{buff} = 10\%$

(b) RPA, $N_{buff} = 1$

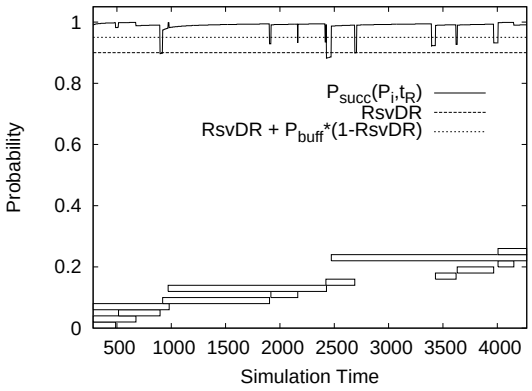
(c) PBA, $P_{buff} = 30\%$

(d) RPA, $N_{buff} = 2$

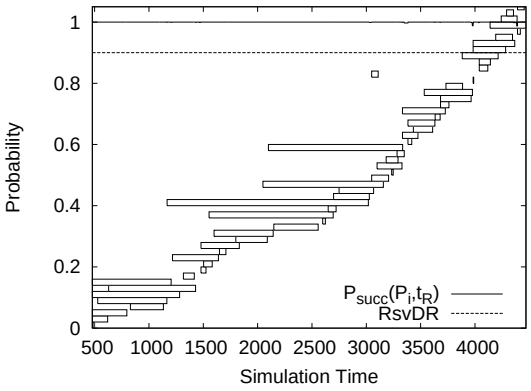
Figure 73: Example Reservations managed with PBA and RPA in P^{3}R^{3}O.KOM - Part 1

the reservation as well as the corresponding reservation success ratio. The series are taken from the simulations with 4000 peers in which 200 reservations were initiated. In Figures 73 and 74, we depict these series using the PBA approach with $P_{buff} \in \{10\%, 30\%, 50\%, 70\%, 90\%\}$ and the RPA approach with $N_{buff} \in \{1, 2, 3, 4, 5\}$. The rectangles in the figures depict individual peers providing resources for the reservation. It is obvious that the average lifetime of a peer is much shorter than the reservation time. Without our solution, P^{3}R^{3}O.KOM, the desired resources could not be used over this long time in the presence of churn. In addition, we show for the series using the PBA approach the corresponding aimed survival probability $RsvDR + P_{buff} \cdot (1 - RsvDR)$. In all cases of the PBA approach, the desired redundancy degree is failed. In the case of RPA only for $N_{buff} = 1$.

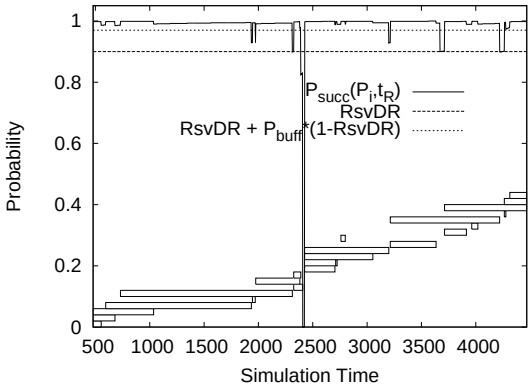
Figure 73a shows that for PBA with $P_{buff} = 10\%$, the aimed success probability of $RsvDR + P_{buff} \cdot (1 - RsvDR) = 0.9 + 0.01 = 0.91$ is not enough to maintain the success probability higher than RsvDR and to maintain the reservation. In the time from minute 654 to 663 no providers exist and the reservation breaks. Although the main manager picks then new providers, the interruption of the reservation is unacceptable. The resource reservation is failed with PBA in the cases of $P_{buff} = 10\%$, $P_{buff} = 30\%$ and $P_{buff} = 70\%$. Figure 74e shows a service example using PBA with $P_{buff} = 90\%$. The resources are continuously provided and the reservation is successful. However, the aimed degree of redundancy is missed three time, at minutes 3425, 3491 and 4278. In Figure 73d, we depict an example using RPA with $N_{buff} = 2$. The behavior is similar to the PBA approach with $P_{buff} = 90\%$, however the result is better due to the increased average number of providers. The RPA approach adds 2 additional peers for providing the resources and in order to strengthen the reliability, thus at least 3 providers in parallel are aimed at. In Figure 74f, we show an example with RPA and $N_{buff} = 5$. In this case, the approach aims at 6 peers in parallel leading to a success ratio for the reservations of 100%.



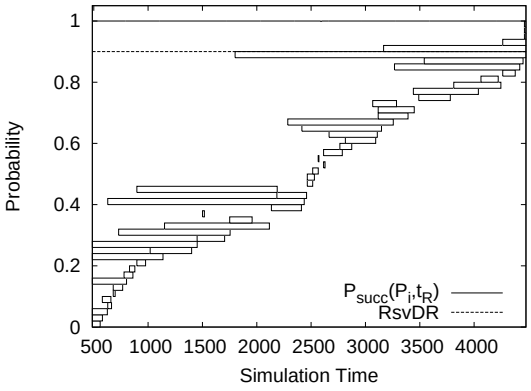
(a) PBA, $P_{buff} = 50\%$



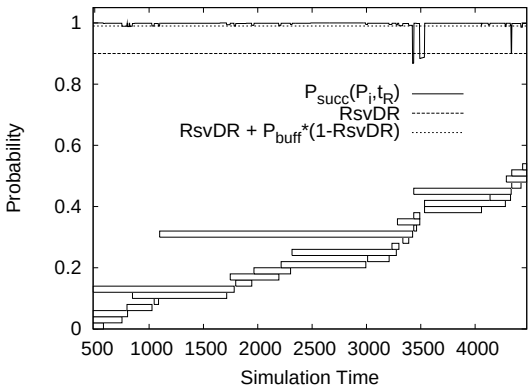
(b) RPA, $N_{buff} = 3$



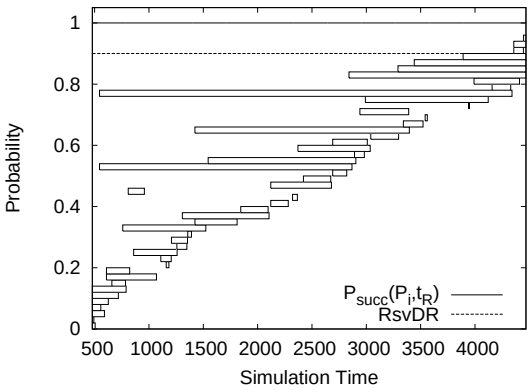
(c) PBA, $P_{buff} = 70\%$



(d) RPA, $N_{buff} = 4$



(e) PBA, $P_{buff} = 90\%$



(f) RPA, $N_{buff} = 5$

Figure 74: Example Reservations managed with PBA and RPA in P^{3}R^{3}O.KOM - Part 2

5.4 RELATED WORK

Resource reservations on an application level in a distributed system have been discussed in the field of GRID computing and service oriented architectures. In these, however, the resources are assumed to stay online enforced by service level agreements. The main question in GRID and service oriented architectures is which resource provider to choose. In p2p systems, the problem statement is different, as peers may fail. In our approach, we do not optimize the resource provider selection but provide reliable long-term provision of resources in a quantity that is desired, regardless of the specific provider.

For service oriented architectures, Berbner et al. and Repp et al. describe in [Ber07] and [Rep09] approaches to manage the quality of service of workflows, to identify quality violations in the execution and to react on them. Regarding p2p-based service oriented platforms, several approaches have been published. In [BDA07], the authors propose an approach to deploy services on peers, aiming at low delays to customers. The approach proposed in [ARZC05] delegates and uses services on peers in a p2p network. Both approaches put the reservation initiation in place to supervise the execution. We aim at long-term resource reservation which allows the initiator to leave the system in the meanwhile. In [LBB08] and [GB09], a p2p-based approach for service discovery in GRIDs is given. The integration of self-organizing elements in GRIDs is presented in [SSR03]. The authors of [CKWC03] propose a p2p-based job scheduler for GRIDs, allowing peers in the GRID to derive their own schedules. The authors of these papers focus on interconnecting and improving existing GRIDs, while we fully utilize the resources of the user devices.

In the field of p2p system, some papers address reliable resource reservations in unreliable p2p systems. The authors of [LCL+09] propose a new overlay for managing resources. We proposed a dedicated mechanism that relies on existing, well evaluated structured p2p overlays. In [DR05], a portal layer is introduced on top of the p2p overlay and resources are communicated with roaming agents. This information is hardly to be complete and does not aim at providing reservations on these resources. In [UJJL05], a p2p grid for distributed resource management in unstructured p2p systems is discussed. The authors propose approaches for service registration and discovery, but do not address the issue of churn. The authors of [RR04] focus in their paper on the retrievability of resource information in p2p networks, they do not focus on their long-term reservation.

Considering related work in the field of reliable resource reservation, we state the claim that only few work has been conducted by the community. This is to our believe based on the fact, that for reliable resource reservation a mechanism is needed to find suitable peers and this field is sparsely investigated as well. P2P systems mainly focused on data-centric application scenarios or on short term resource consumption at the edge. With the upcome of service oriented architectures, however, the topic of a p2p service oriented architecture is emerging as well. We address this field with our approach for reliable resource reservation in unreliable p2p systems.

5.5 CONCLUSIONS

Having introduced, P^3R^3O.KOM, a protocol for initiating, maintaining and providing resource reservations. In the following, we discuss the features of the solution. Churn is the main influence factor for unreliable reservations. We address this issue by redundancy, both of the resource providers as well as the resource reservation managers. The set of resource providers is periodically checked and more providers are added if the success of the reservation is endangered. The set of reservation managers consists on the one hand of a deterministically chosen peer based on the Reservation ID and additional peers for backup. The main manager, being responsible for the Reservation ID in the DHT, maintains the set of resource providers and reservation managers. In the case of its failure it is immediately replaced and updated by the backup maintenance peers. The reservation requester as well as other users may always ask the main reservation manager, identified by being responsible for the corresponding Reservation ID, for an actual set of resource providers.

In this context, the functionality of the structured p2p overlay is important, as it provides key-based routing and supports thus dedicated roles in the p2p network. We rely on the KBR interface of the used p2p overlay and assume that the p2p overlay provides dependable routing and copes with churn, keeping the ID assignments always up to date.

The same assumption is also used by SkyEye.KOM in order to create and maintain the monitoring tree, as described in Chapter 3. We use the functionality of capacity-based peer search of SkyEye.KOM to pick suitable peers in the required quantity for the reservations. Through the modular approach, functionality is reused and can independently be evaluated and improved

Regarding the resource reservation, we proposed a twofold mechanism with a distributed reservation management set and a set of resource providing peers. The management set acts as contact group for reservation initiations and access to the list of resource providing peers. This set of resource providers is controlled by the resource manager and supplemented with a sufficient number of additional peers in the case that the peers in the set are not expected to stay online until the next liveness check. For the peer lifetime estimator, we used a Weibull model derived from KAD measurements [SENB07], providing for a given lifetime the probability that the peer will stay additionally online for a given round length more. However, a more optimized lifetime estimator could be used, based on the monitoring of join and leave patterns using SkyEye.KOM. For the evaluation of the approach the used model is already very precise.

We proposed two approaches to identify the required number of redundant peers for resource provisioning, the probability buffer assignment approach and the redundant peer assignment approach. The evaluation shows that the proposed solution in combination with the redundant peer assignment approach fulfills the resource reservations in 100% of the cases with $N_{buff} = 4$ and $N_{buff} = 5$ with a service cost of 5.10 and 6.02 times more resource investing than requested. The traffic overhead for maintaining the reservations by the resource managers is in maximum 1745 kb in slightly more than 2 days, for which the reservation was lasting. Thus, the costs for maintaining are considered very low while the resource reservation is provided with 100% success ratio.

This solution allows for reliable resource reservation in unreliable p2p networks with continuously joining and failing peers. Through the creation of a reliable platform, service may be deployed in a distributed fashion implementing quality controlled resource usage and service deployment, allowing to create a p2p-based service oriented architecture.

6 MANAGING THE QUALITY OF SERVICE OF PEER-TO-PEER SYSTEMS

Notice that the stiffest tree is most easily cracked, while the bamboo or willow survives by bending with the wind. All fixed set patterns are incapable of adaptability or pliability. The truth is outside of all fixed patterns.
- Bruce Lee

All our knowledge begins with the senses, proceeds then to the understanding, and ends with reason.
- Immanuel Kant

6.1 MOTIVATION

The focus of this chapter is on managing and controlling the quality of service in a structured p2p system. The quality of service of a p2p system describes the behavior of a p2p system in quantifiable metrics. These metrics describe, for example, the lookup delay, the traffic overhead, hop count in the overlay and many more. They are influenced by various aspects of a running p2p system. On the one hand, the design, architecture and configuration of the p2p mechanisms used influence the quality of service of the p2p system. On the other hand, the demand on the quality of service of a p2p system is given by the applications and influenced by the scenario in which it is used as well as the behavior of the peers.

The p2p application defines the quality of service expectations on the p2p system. A p2p overlay used for managing video streams with tight user interaction results in low delay requirements, whereas the same p2p overlay used for reliable data backup should provide consistent lookup results. Although both quality aspects may be important in both application fields, each application may still define individual quality of service requirements on the same p2p mechanism, such as the overlay.

The scenario a p2p system is used in defines the resources available for the operation of the p2p system and defines thus also the resulting constraints and challenges. Considering the metric which describes the traffic overhead of a p2p overlay, we give an example illustrating the issue. In a mobile scenario it is more important to minimize the traffic consumption than in an enterprise scenario within a local area network. Although both scenarios may use the same p2p system and mechanisms, the influence on the quality of service provided by the p2p system is given through the constraint of limited bandwidth. The p2p system behaves differently and it is more challenging to provide the same quality of service in the mobile scenario as in the LAN scenario. Obviously, the p2p system should adapt (itself) to cope with the given scenario.

The behavior of the peers in the p2p system also has a large influence on the quality of service of the corresponding p2p system. Variable peers join and leave as well as user access patterns define a workload on the p2p system that influences its quality of service. Thus, the system properties resulting from the design and the configuration must be matched to the quality of service expectations defined by the p2p application as well as the influencing factors of the scenario and the peer behavior.

In order to reach and keep a specific level of the quality metrics in a systematic way, a static configuration setup for the p2p system is not sufficient. We devise in this chapter *SkyNet.KOM*, a framework for managing the quality of service of p2p systems, which automatically adapts the configuration of the p2p system in order to reach and hold the preset quality of service requirements.

For this, we adopt a self-optimization cycle, similar to the idea of autonomic computing. A system provider defines the boundaries for metrics of interest that represent the quality of the p2p system, such as keeping the lookup delay below 200ms. Using SkyNet.KOM, the systems' quality is monitored using SkyEye.KOM and evaluated in comparison to the preset goal quality intervals. Additionally, the configuration parameters of the peers are monitored. Once the quality goal, i.e. the preset metric intervals, is missed, the configuration of the system is automatically adapted in order to reach the preset quality intervals. Besides monitoring the quality metrics and system configuration, the interdependencies between the configuration parameters and quality metrics may be analyzed.

Next, we present the functional and non-functional requirements for providing controlled quality of service in structured p2p systems. For the investigation of the solution space, we present in Section 6.2 a discussion of approaches related to the management of p2p systems. In specific, we advocate that monitoring and autonomous self-configuration of p2p systems is a viable and effective way for the solution. As a general approach to apply the investigation results, we present the idea of autonomic computing and its applicability in the field of p2p systems. Subsequently, we present SkyNet.KOM, our solution for solving this problem in Section 6.3. This chapter closes with an evaluation of the proposed self-configuration framework in Section 6.4, discussion of related work in Section 6.5 and a summary in Section 6.6.

6.1.1 *Functional Requirements*

P2P systems are designed to provide various functionality for the interconnected peers, for example, allowing ID-based routing and ID-based distributed storage and retrieval of objects. When building p2p systems, various mechanisms are combined to provide the functionality for an application and to reach the desired quality goals for the specific application. For example, a p2p-based voice over IP application would require lookup times of below 100ms in the p2p overlay. A distributed data storage application may not state tight requirements on the lookup time but rather on the availability of the stored objects.

In this context, we focus on the provided quality of the service of the p2p system. The quality of service is described in terms of metrics and may be monitored with SkyEye.KOM as described in Chapter 3.4. System monitoring offers a view on the current status of the p2p system, but no tool or functionality to modify or adapt the observed quality. Management of the quality of p2p systems aims at the adaptation of the system quality, i.e. system metrics. This is a task that goes beyond the observation of the status and requires a direct and effective influence on the control points of a large-scale, distributed p2p system.

Each metric has its own $name$, e.g. relative upload bandwidth consumption, and the statistical aspects of it correspond to a specific $type$, such as the average. The value $V_{type,name}$ describes the statistical value over all peers in the tree, regarding the metric $name$ (e.g. hop count) and the statistic aspect $type$.

This metric may be one of the monitored metrics, as depicted in Table 2. Managing the quality of service of a p2p system means that the metrics of the p2p system can be controlled and kept in a desired metric interval. To achieve this, the p2p system provider may define metric boundaries or intervals, $(G^{min}_{type,name}, G^{max}_{type,name})$ on a set of metrics. An interval defines a valid range for a metric, such as (0ms, 100ms) for the lookup delay or (0 KB/s, 10 KB/s) for traffic overhead. Please note that the interval ranges may also contain the lowest (0 or -∞) and highest (0 or ∞) values for the metrics.

The interface provided by this mechanism is:

- void setMetricInterval(Type type, Name name, Value min, Value max) - defines a valid interval for the specified $type$ of the metric called $name$

Goal of a mechanism for managing the quality of service of p2p systems is to bring and keep the monitored metrics in the predefined quality interval:

$$G^{min}_{type,name} \leqslant V_{type,name} \leqslant G^{max}_{type,name} \tag{6.1}$$

In order to implement this functionality, we require both, mechanisms that are able to be adapted in their optimization goal as well as mechanisms that allow to identify and communicate new optimization goals for the system quality.

6.1.2 *Non-functional Requirements*

After the introduction of the functional requirements of a solution, we discuss the non-functional requirements for the problem statement. Mechanisms for providing reliable quality of service in structured p2p systems may follow various design decisions. However, following non-functional requirements must be met by a solution.

SCALABILITY
A mechanism for managing the quality of service of a p2p system must be applicable in networks with millions of nodes without severe consequence for the p2p system. Thus, a solution must be lightweight and the load generated on each peer must be independent of the number of peers in the network. However, the number of considered metrics and constraints is not expected to scale too high, as we expect that a system provider defines quality intervals on the most important metrics of the system. The number of reasonable system metrics is limited and we do not expect it to scale to thousands.

RELIABILITY
Reliable quality of service control involves the maintenance of a desired quality level or reservation status over a long period of time. Thus, a reliable approach for controlling the quality of service of a p2p system, does not just enables the p2p system to reach the desired quality levels, but also to keeps the system at these levels. In the case of violated quality metrics, the approach should automatically initiate counteractions to restore the desired quality of service. The reliable provision of quality of service is the core goal of the discussed mechanism in this chapter.

EFFICIENCY
Regarding the performance and costs for a proposed solution, we assume that the most important performance indicator for the management of the quality of service is the time the mechanisms requires to stabilize the system and to affect a desired metric to reach a valid quality state. This metric defines how quickly a metric violation is resolved. The costs for resolving the metric violation may be measured by traffic costs in total and on average per peer. An efficient management solution for the quality of service of p2p systems minimizes the traffic overhead for reaching a designated metric interval with a violating metric.

STABILITY
One main aspect of the management approach is to stabilize the quality of service in the presence of churn. Churn or, more generally, varying user behavior, introduces dynamism in the p2p system, affecting the quality of service of the p2p system and the resources provided for the system. In order to counteract this dynamism and to provide a stable quality of service, the proposed mechanism is in place. A non-functional requirement on the stability of the proposed solution is that it copes with frequent and drastic changes in the number of peers in the system and keeps on providing the desired quality of service and resource reservation.

CONSISTENCY
Consistency requires a common view on the system among the peers. Thus, a consistent mechanism for providing managed quality of service must also take care that all peers experience similar quality of service and that the management includes them all. For example, the lookup delay among the peers should have a low standard deviation and all peers should be involved to reach this goal.

After the review of the quality properties and the requirements on the quality management process in p2p systems, we analyze the potential approaches for influencing the quality of service provided by a p2p system in a systematic way. We aim to deriving best practices and a viable approach for a quality of service management framework.

6.2 INVESTIGATION ON THE INFLUENCE OF QUALITY OF SERVICE IN P2P SYSTEMS

In this section, we discuss approaches to systematically affect and manage the quality of service in a p2p system. We aim in this section at investigating the possible approaches to affect the quality of service in a distributed large-scale network with autonomous peers. As no central coordinating instance exists, a distributed approach needs to be used. We sketch a solution space and exemplary analyze the tools available to coordinately affect the behavior of the p2p system. We first describe the influencing factors on the behavior of a p2p system, specifically the various kinds of dynamism. In order to address this dynamism and resulting, misleading quality trends various stabilizing mechanism may be adopted that

aim to restore the quality of service of the p2p system. As a conclusion of this section, we identify a viable approach to generally manage the quality of service of p2p systems.

We assume that each peer individually applies the depicted stabilizing mechanism. In order to decide on the strategy to follow, each peer individually decides based on the information it has regarding the quality of service of the p2p system. We define three categories for the quality management mechanisms based on the scope of the monitoring view of a single peer: local view only, partially global view and global view.

In the following section, we discuss these three categories. In the first category, described in Subsection 6.2.1, a peer has only a *local view* on the quality of the p2p system, that is, it only gains information about its own performance in the system. We discuss this category on the basis of KBR-compliant p2p overlays, which aim to provide differentiated quality of service for the routing of dedicated message types. We show that the configuration of a mechanism applied in the p2p system offers a viable approach to manage the quality of service of the mechanism in the p2p system. We also show that the influence of such an approach is linked to the monitoring scope.

The second category, described in Subsection 6.2.2, extends the monitoring scope from a local view, to a *partially global view*. Thus the mechanism we focus on has a view on a subset of the p2p network and is thus a partially global view. We show that in this partial global view, metrics may be managed that relate to the whole p2p system, such as the load distribution. On the basis of a multimedia streaming scenario, we show that stabilizing mechanisms are possible that are configurable in the goal of quality of service and able to adapt both peer-specific (e.g. download time) as well as system-specific (e.g. load distribution) quality of service metrics.

The third category of quality management mechanism, described in Subsection 6.2.3, uses the *global view* on the quality of service of p2p system in order to derive an optimized configuration for the p2p system. In order to show the potential of this approach, we give a brief overview on current approaches in the literature to manage the quality of service of individual p2p mechanisms. We show that several approaches discussed in the literature may be combined and used on demand through an automated configuration approach.

At the end of this section on investigating viable approaches to influence the quality of service of p2p systems, we give a short summary and motivate an automated self-configuration approach for p2p systems based on the global view provided by SkyEye.KOM for the systematic management of the quality of service of p2p systems.

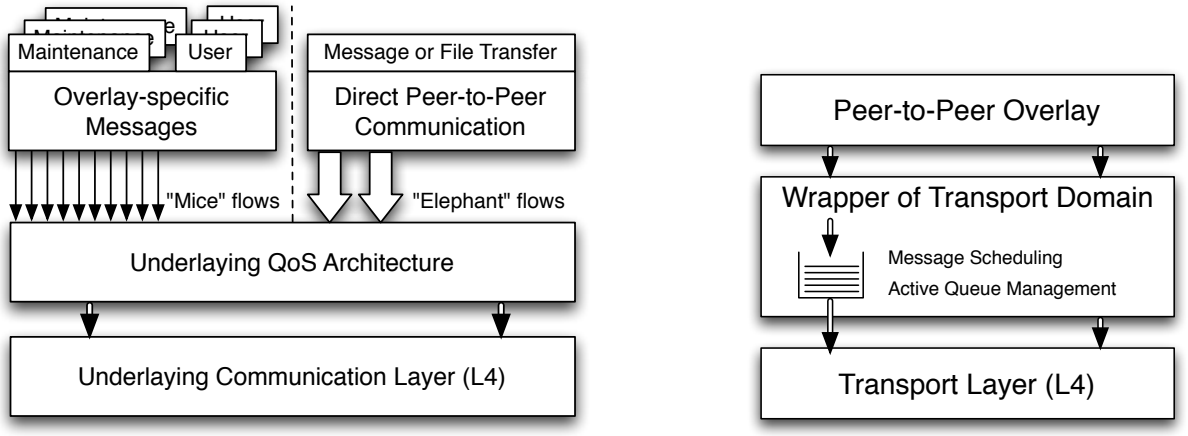
6.2.1 *Configurable Quality in Routing through Prioritization*

In our first investigation, we analyze the various traditional quality of service approaches for the overlay routing functionality in a structured p2p overlay. In this example, we investigate the possible approaches to manage the metrics of a single mechanism (here a p2p overlay) in a systematic manner using only local knowledge. A proposed mechanism should both be able to consider changing quality of service requirements stated from the p2p application, as well as the dynamics of the p2p system.

Thus, we want to adapt and manage the quality of the lookup and routing functionality of a structure p2p overlay regarding the two metrics *delay* and *loss*. In order to provide dedicated, configurable quality for individual overlay operations, such as lookup, join, leave, keep alive, we focus on the routing of the corresponding overlay messages. Delay and message loss are typical quality of service metrics considered on the network layer of the ISO/OSI layer model as well. As a p2p network is manifested in a p2p overlay on top of a network infrastructure, similar concepts may be used.

An overlay is created and maintained by periodically exchanging messages with other peers in the network. Common overlay operations are lookup, search and store operations. However, once a communication partner is identified in the network using the overlay operations, direct p2p communications are initiated. In the case of limited bandwidth capacity, various strategies can be applied in order to limit the negative effects of traffic peaks, including messages loss and delayed message delivery.

Bandwidth limitations are likely to occur, due to two reasons. Download bandwidth is often larger than upload bandwidth, so that peers tend to retrieve more data than they can send out. Furthermore, the computing power of network devices is assumed to be sufficient, and incoming traffic can be processed fast enough. Thus, the output link remains a bottleneck and congestion may occur. Figure 75a



(a) Quality of Service for P2P Flows

(b) The Wrapper of Transport Domain Layer for Overlay Bandwidth Management

Figure 75: Flows in P2P Overlays Managed by a WTD Layer

depicts the various flows in a p2p system: overlay-specific flows, direct p2p flows and underlying network flows.

Direct p2p communication is used for long-term transmission (*elephant* flows), for example, of file transfer. In contrast to this, p2p overlays use only short-term flows (*mice* flows), with only few messages exchanged with every contact. Esten et al. affirm this observation in [EV03] through measurements of p2p traffic. In a first attempt, we investigate the characteristics of corresponding flows and propose an overlay bandwidth management approach with regard to delay and loss for specific quality classes.

TRADITIONAL QUALITY OF SERVICE APPROACHES

The traditional quality of service approaches on the network layer are:

- **Overprovisioning:** By adding more resources to the system, the congestion in the network is reduced and thus the delay and loss in the system is decreased.
- **Price-controlled Best Effort:** By introducing quality classes for the flows with individual pricing, higher quality channels are presumably used less, leading to less congestion.
- **Differentiated Services:** Various quality classes for the flows are introduced with individual processing effort. Higher quality classes are processed with a higher priority on the cost of lower priority classes.
- **Integrated Services:** Each individual flow is setup with a previously negotiated quality class. This approach results in a fine granular quality assignment for each flow on the costs of an increased flow establishment overhead.

In the context of p2p systems, overprovisioning is very challenging, as it involves adding peers to the p2p system in a magnitude related to the number of peers in the system. This results in very high operational costs for a system provider and thus is considered unfeasible. Additionally, by adding more peers to the system, the routing of messages is delayed as an increased number of peers also results in an increased hop count, which is typically $O(\log(N))$.

Price-controlled Best Effort inside of p2p systems is also very challenging, as in p2p systems peers are participating voluntarily without paying for the infrastructure. The infrastructure is provided by the peers themselves and thus it is unclear who should benefit from the earnings through a higher quality channel. More importantly, a basic assumption for p2p systems is that the maintenance and operation of the p2p system is provided by all peers and all peers should benefit in the same manner. Thus, approaches relying on the Price-Controlled Best Effort approach are not viable in the context of p2p systems.

Differentiated and Integrated Services both provide dedicated quality of service for individual flows. The question that arises and we focus on next is whether the overhead for Integrated Services is justified for flows in the p2p system or whether Differentiated Services should be adopted.

Elephant flows are few and large in a p2p system in relation to the number of peers. For them, it seems that Integrated Services are feasible. Moreover, as they occur mainly for direct connections between peers, the peers involved may negotiate a specific quality of service for the elephant flows and use it for a long time. The flow establishment overhead is very short in comparison to the long usage of the flow. Several approaches exist to provide dedicated quality of service for such large flows, such as in [Karoo]. The resource reservation protocol (RSVP) [FDB+07] may be used for controlled quality. Also, the lower than best effort service [BNW03] or alternative best effort [HBTK] service may be used, if no specific quality of service requirements are stated.

MOTIVATION FOR OVERLAY BANDWIDTH MANAGEMENT

The provision of quality of service for overlay-related messages, the *mice* flows, is more challenging. Their relevance for the p2p system is diverse and may be individually set by a p2p system provider. A lookup, for example, may have tighter delay constraints than keepalive messages. Additionally, these requirements may be changed by the provider over time. This may happen in certain cases, such as when the overlay is at risk of falling apart or partitioning. In that case, keepalive and further maintenance messages should be handled with higher priority than lookup messages. Lookup messages, on the other hand should have higher priority in well-connected overlays.

The management of the quality of service of overlay messaging flows involves the following challenges:

- Quality of service requirements are diverse for the overlay-specific flows.
- An approach providing managed quality of service for various overlay message types should cope with the dynamics in the p2p overlay in form of churn and bandwidth heterogeneity.
- Peers may rely on local observations of the quality of service of their lookup operations
- The peers follow the coordinated aim to provide the desired quality of service for various overlay message types

In order to decide whether Integrated or Differentiated Services are the better choice for providing quality of service for overlay operations, we first have to look at the potential overhead related to both approaches.

Characteristics of Flows in the P2P Overlays

In order to provide quality of service for flows in a p2p overlay, we investigate the characteristics of flows in structured p2p overlays and whether a flow-based (Integrated Services) or quality class-based (Differentiated Services) mechanism should be applied. In the research field of queue management, the term flow describes periodically occurring events or messages initiated by a known instance that needs to be processed. For CPU cycle scheduling and message scheduling between known endpoints it is easy to define flows. In CPU cycle scheduling, jobs related to a single process are defined as a flow. In packet scheduling, packets related to a communication path between specific end points are defined as flows. In contrast to this, it is challenging to define a flow in structured p2p overlays, as events related to a flow need to be periodically occurring. However, in the context of structured p2p overlays the destination points of lookup queries, join and leave messages and routing point discovery messages are rarely known.

Consequently, we state the hypothesis that the number of requester-replier pairs in the overlay is large and that a specific combination is not reoccurring periodically. This hypothesis is also manifested in the term mice-flows, which states that the flows are short and of high quantity.

In order to investigate the characteristics of flows in structured p2p overlays, we simulated Kademlia [MM02] with 10,000 peers for two simulation hours. As the main metrics for the flow characteristics we consider the number of *contacts per peer* and the *number of messages per contact*. A contact is an endpoint that is addressed by a peer by sending a message to this contact. The quantity of this metric describes the number of connections a peer needs to establish in the overlay. The number of messages

per contact gives us the "size" of the flow or, in other words, how long it is in use. As the time and message overhead for establishing a quality-assured flow in the sense of Integrate Services does not vary, it is relevant to know the ratio for the overhead for establishing a flow to the traffic generated by the flow. Having only short flows of few messages makes the Integrated Services approach inefficient.

In Figure 76, we examine the number of messages per contact in relation to the number of contacts each peer in Kademlia has. This figure shows us how many peers a single peer receives messages and how many messages are received on average. In our simulations, a peer in a network with 10.000 nodes has on average 1437 contacts in two hours simulation time. The average number of contacts is huge in comparison to the total number of nodes. Further, we see that a peer approximately receives only 1.4 messages per contact. These simulation results match the traffic measurements of real systems presented in [EV03]. Thus, messages from the same contact are too sporadic too be grouped as "flows" according to their initiator.

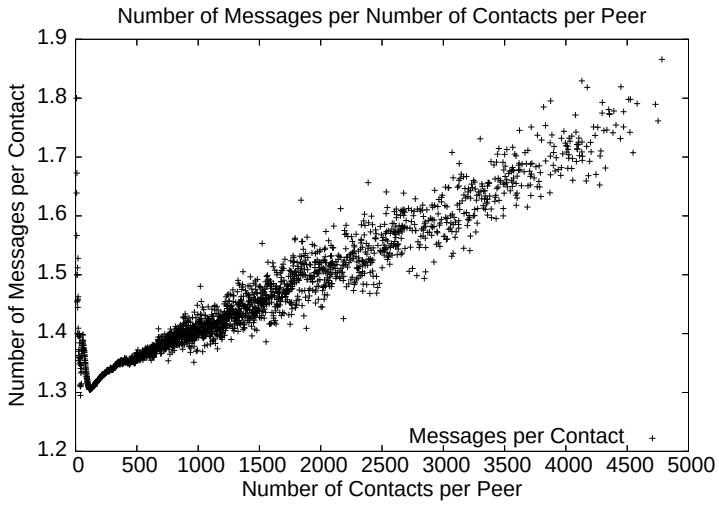


Figure 76: Characteristics of Overlay Flows: Messages per Contact per Peer

Considering these two observations with regard to the applicability of per-flow mechanisms in the context of *mice* flows, lead to the conclusion that it is unfeasible to maintain a status for all "flows" identified by the source peer. Typically, source-destination pairs are used to group messages to flows. Using the source-destination pair of a message is even less feasible in our case, as we get more flows ($O(N^2)$) per peer, with even less messages by each peer. Thus, the characteristics of flows in the network layer and the p2p overlay are fundamentally different. This leads to the infeasibility of providing Integrated Services for flows in the p2p overlay.

However, regarding the feasibility of Differentiated Services, we see that a set of quality classes for the various message types may be used. The number of message types in a p2p overlay is limited. Message types may be lookup queries and replies, keepalive messages and several other maintenance messages. These message types may be mapped to quality classes, for which a dedicated quality of service is provided by the p2p overlay. In our approach, we use multi-dimensional priority classes, and take into consideration delay and loss requirements.

Quality of service for p2p overlay flows with regard to delay and loss needs to be provided in the presence of resource limitations. Considering a case with sufficient bandwidth in all peers for forwarding and processing messages does not lead to quality limitations as every message is processed instantly. In the case where a peer does not have sufficient resources, such as bandwidth, we identified two problems that have to be addressed. When congestion occurs, messages cannot be sent and are stored in the buffer of the congested peer. When bandwidth is available again, the peer can choose which message to transmit next, and this process is called *scheduling*.

The second problem that needs a solution is given by the limitation of the buffer size in each peer. If the transmission rate of a peer is smaller than the arrival rate of new messages, the size of the buffer increases constantly. Due to the limited buffer size in reality the peer has to choose which packets to drop in case of buffer overflow. This problem is addressed by *active queue management* (AQM). In

Figure 77 we sketch the principle of scheduling and AQM, with message priorities 1,2,3 for scheduling and A,B,C for the AQM mechanism. In the following section, we introduce multi-dimensional message priorities and show that priority-based schedulers and AQM mechanisms can give guarantees with respect to transmission delays and loss avoidance.

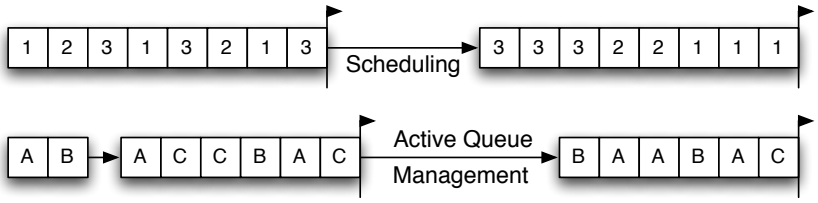


Figure 77: Principle of Scheduling and Active Queue Management

APPROACH: DELAY AND LOSS AWARE OVERLAY BANDWIDTH MANAGEMENT

Next, we present our solution for the scheduling and AQM problem in structured p2p overlays. We introduce a static priority scheduler and AQM mechanism called HiPNOS.KOM that provides guarantees in the context of delay and loss for routing messages in the p2p overlay. First, we discuss the requirements for a solution and the placement in a layered model.

Requirements for a Solution

Message priorities are motivated by the diversity of applications that may use a p2p overlay. Numerous message types exist in the implementation of an overlay, and their relevance for the functionality of the system differs. In addition, the orders of the user may have differing relevance as well. In order to model the relevance of a message with respect to delay and loss, a solution has to take multi-dimensional message priorities into account. We use the term *message class* to describe all messages with the same priority. We state in the following requirements the design goals of our solution.

The average queue delay $D_Q^{avg}(i)$ for messages of priority class i should be smaller than the average queue delay $D_Q^{avg}(j)$ of lower prioritized message classes j of the set of delay priorities P_D:

$$\forall i, j \in P_D \; with \; i > j \; : \; D_Q^{avg}(i) < D_Q^{avg}(j) \tag{6.2}$$

Another requirement is that starvation of lower prioritized messages does not occur. Every message has to be processed in a reasonable time.

For the decision about which message to drop out of a full queue we postulate that no message of importance should be dropped if a less important message exists in the queue:

$$\forall i, j \in P_L \; with \; i > j \; : \; Loss_{avg}(i) < Loss_{avg}(j) \tag{6.3}$$

Additionally, we state the requirement that a solution should be independent of a specific structured overlay. This goal aims for the re-usability of the approach.

Placement on Layer Model

In order to apply an approach for providing configurable quality of service for overlay message types for various structured p2p overlays, we propose a general layer below the p2p overlay, termed the *Wrapper of Transport Domain (WTD)*. Scheduling and AQM mechanisms for overlay messages are applied in this layer, as it interacts tightly with the transport layer. The content of messages is not relevant to the WTD layer as only quality class information regarding scheduling and AQM are of interest for this layer. This quality class information could either be passed via header information or derived from a configuration defined by the system provider. In Figure 75b we show the layer model containing the overlay and WTD layer.

HiPNOS.KOM - High Priority First, No Starvation

In order to investigate whether Differentiated Services for the processing of overlay messages are feasible, we propose a simple scheduler and AQM mechanism for the WTD layer. Our approach, termed *HiPNOS.KOM*, is placed in the WTD layer, managing the messages passed from the layers above for transmission using the layers beneath. Each peer is expected to have a buffer B to store messages for the case that the transmission channel is in use. Furthermore we assume that a peer p can estimate its own available upload capacity U_p. There exist feasible mechanisms to achieve this task, and several approaches are presented in [SKK03].

Messages in the p2p system are marked with a two-dimensional priority value $(P_D; P_L)$, characterizing their criticality regarding delay and loss. These priorities may either be set by upper layers or derived, based on the message type, from a preset configuration of the system provider. It is assumed that these packets are marked according to their relevance for the system. This assumption is valid, as higher layers or the system provider know the semantics of the packets and are able to give additional information on it on their relevance to the lower layers.

As a message $m_{P_D(m),P_L(m)}$ is passed from the higher layers to the WTD layer, HiPNOS.KOM follows the following strategy:

- Only if the buffer B is empty and bandwidth is available is $m_{P_D(m),P_L(m)}$ transmitted. Otherwise $m_{P_D(m),P_L(m)}$ is stored in the buffer.
- If the buffer contains messages and upload bandwidth is available the following steps are processed:
  1. The message with the highest delay priority $P_D(\cdot)$ is chosen.
  2. As a tie-breaking rule the buffer-insertion time is considered in order to choose the message that is longer in the buffer.
- In order to avoid starvation of lower prioritized messages, the delay priority of all messages in B is increased with every time unit Δt.

For active queue management, HiPNOS.KOM uses a simple priority-based mechanism. If the size of B reaches a predefined threshold value and a new message arrives, a message m^* is dropped. In order to calculate m^* let $T_{arrival}(m)$ be the arrival time of message m in the buffer. We define the subset of messages B_{min} in the buffer with minimal loss priority as $B_{min} = m \in B$ with $P_L(m) = min\{P_L(x) \mid x \in B\}$. The message m^* that is dropped is then determined by $m^* = m \in B_{min}$ with $T_{arrival}(m^*) = min\{T_{arrival}(x) \mid x \in B_{min}\}$.

Higher prioritized messages are always considered more relevant than lower prioritized ones. One may argue that in a congested scenario this may lead to the case where messages of specific message types with low loss priority are never processed. This case only leads to problems in the network when the relevance of the messages is estimated incorrectly. HiPNOS.KOM supports dynamic priority changes of messages types. Higher layers are assumed to modify the priority setting of messages that are passed to the WTD layer in order to adapt to the network characteristics. The WTD layer itself is incapable to determine the relevance of the semantics of messages, due to the limited scope of the view regarding the quality of service of the p2p system.

HiPNOS.KOM is designed to provide service guarantees on message transmission to higher layers. The functionality of the overlay layer should not be thwarted by the incapability of the layers below. This aspect is very important if a p2p network contains numerous devices with low bandwidth capabilities.

Regarding the complexity, HiPNOS.KOM can be implemented demanding $O(1)$ processing time using hashmaps (to identify the queue per priority class) and calendar queues (for enqueue and dequeue operations regarding the arrival time of the messages). The storage demand of HiPNOS.KOM is $O(|B| + max\{|P_D|, |P_L|\})$ where $|B|$ is the size of the buffer and P_D and P_L the set of priority classes regarding delay and loss. The buffer size is limited to a predefined threshold. Before presenting the evaluation of HiPNOS.KOM, we briefly discuss related work.

Related Work on Overlay Bandwidth Management, Scheduling and AQM Mechanisms
Now, we discuss the state-of-the-art solution for the scheduling and AQM problem, as well as overlay bandwidth management approaches, and give a brief overview on other solutions presented in literature.

State-of-the-art p2p overlay implementations do not consider the problems occurring from bandwidth congestion. A common approach for implementing p2p overlays is to focus on the overlay layer and to leave details on message transmission for the lower layers. Typically messages are created in the application and passed to the network handler of the operating system. TCP provides congestion avoidance strategies, but does not consider message priorities. In order to provide guarantees in terms of delay and loss, mechanisms have to be implemented that tightly interact with the overlay layer.

As a results of the arguments above, the state-of-the-art mechanisms used for queue management on an overlay layer are *First-In-First-Out* (FIFO) and *Drop-Tail*, we present both in the following.

The FIFO principle processes the incoming messages ordered by their arrival time. Its implementation complexity is O(1). Due to its simplicity, FIFO is the dominant scheduling mechanism currently used in p2p overlay implementations. Messages are passed from the overlay layer to the transport layer and transmitted as soon as bandwidth is available. It is obvious that this strategy does not provide any guarantees on the delay of a message transmission.

The simplest way to handle network congestion is an approach called Drop-Tail. New packets are queued as long as there is place for them in the queue, which is limited in length. The queue may get full because the sending rate of the output link is smaller than the arrival rate at the input link. In this case, new packets are dropped. Drop-Tail does not provide any guarantees that highly prioritized messages are transmitted.

In contrast to these two simple mechanisms, more technically mature mechanisms also exist. In [GLS07], we analyzed several scheduling mechanisms discussed frequently in the literature. Our investigation shows that the majority of the scheduling approaches assume the existence of message flows defined by sender-destination pairs. As we have shown earlier, flows in this sense cannot be assumed in p2p networks. In addition to the taxonomy on scheduling mechanisms, we analyzed in [GPLS07] several common AQM approaches. Our taxonomy on these AQM mechanisms reveals that many of them rely on flows as well.

Based on the taxonomies stated in [GLS07] and [GPLS07], we present in the following section the scheduling and AQM solutions that are independent of sender-destination pair based flows.

For ease of presentation the following overview on existing solutions adopts the terminology to p2p overlays, although the original papers had been proposed for another field of application. In general we changed the term *flow* to *message class*, which describes the set of existing message types or message priorities in the system.

Nagle proposed in 1987 in [Nag87] Fair Queuing (Round Robin), a simple scheduling mechanism for packet switches whereby each flow is assigned to a queue of its own. When adapted to p2p overlays this means that each message (priority) class has a queue of its own assigned. Messages are transmitted using the Round Robin principle to choose the next queue to be serviced. This approach can be implemented very efficiently, as no further computation is needed. Round Robin is a feasible solution for providing fairness among several message classes. However, if message priorities have to be considered, additional control parameters, like weights, need to be introduced.

Classical Round Robin provides an equal share of service to all message classes in the system. Weighted Round Robin (WRR), presented in [KSC91] by Katevenis et al., introduces for each class i weights w_i, which define the amount of share they receive. The round robin share for each class i is $\frac{w_i}{\sum_{j \in F} w_j}$; this is also the fraction of the total service provided for message class i. This approach is capable of considering prioritized messages by adopting the share of service a message class i receives to its priority. Congestion may occur, but each message class receives a certain share of bandwidth. The drawback of this approach is that it only controls the maximum rate of service a specific message class receives. WRR controls how much throughput is guaranteed to a class. This is independent to the requirements of having low delays.

Core-Stateless Fair Queuing (CSFQ) is introduced in [SSZ98] by Stoica, Shenker and Zhang. Their main goal is an efficient fair queuing algorithm with strong complexity reduction. This is achieved by introducing two types of devices: edge and core routers. Core routers are surrounded by edge routers

so that all traffic coming from the rest of the network has to pass an edge router before coming to a core router. Edge routers estimate the traffic at the edge of this network island and label packets with the rate of their flows. Core routers use these labels to calculate a minimum service rate for all flows. Upon congestion packets that exceed a specific threshold above the minimum service rate are dropped.

An extension to CSFQ is Weighted CSFQ [SSZ98]. Each flow i is assigned a weight w_i that has impact on the share the message class receives. The higher the weight of a class, the lower the probability that packets of this class are dropped. Flow i with weight w_i receives in the time interval $[t_1, t_2]$ not more share than $w_i \cdot \alpha \cdot (t_2 - t_1)$, where α is calculated dynamically as the maximum service share for all classes.

A further improvement of CSFQ, presented in [SZS02], is Self-Verifying Core-Stateless Fair Queuing (SV-CSFQ). The authors argue that the concept of having edge and core routers is not applicable, because it is unfeasible to isolate an island of core routers by surrounding them with edge routers. Therefore, they suggest using only one kind of routers, which periodically checks the validity of packet labels. In the case of inappropriate labels being adopted, packets are relabeled and the service rate is adapted. Please note that in p2p systems it is possible to have an island of core routers. Each peer is an core router as it has to forward messages to other nodes, but is also an edge router as it may initiate overlay-specific actions.

Sally Floyd and Van Jacobson present in [FJ93] an AQM mechanism called Random Early Detection (RED) that aims to avoid congestion. Their work is motivated by the goal to keep average queue sizes in routers small. This is done by dropping or marking packets with a probability related to the position of the messages exceeding a certain threshold in the queue. System-wide parameters Q_{min} and Q_{max} define the threshold boundaries of the queue size. Q_{min} defines the minimum queue length where no packets are dropped, as the weighted average queue size Q_{avg} exceeds Q_{min} the dropping probability increases with increasing Q_{avg} and number of packets since the last dropped packet of the same flow up to a maximum dropping probability.

There exists a wide range of AQM mechanisms based on RED. ATM-RED [RBL99] takes the characteristics of ATM networks into account. Adaptive-RED [FGS01] adapts the target queue length to meet delay and throughput requirements. Stabilized-RED [OLW99] considers the bandwidth share of the flows in order to increase the diversity of flows in the queue. Fair-RED [LM97] measures the utilization of bandwidth per flow in order to impose on each flow a loss rate that is related to its bandwidth utilization. RED with Preferential Dropping [MFW01] maintains a dropping history in order to identify flows that utilize bandwidth excessively. Flows with a high number of previously dropped packets are preferred for dropping. The main idea of "Choose and Keep Packets from Responsive Flows" (CHOKe) [PPP00] is to compare an arriving packet with n random packets in the queue. All randomly picked packets having the same flow identifier like the arriving packet are dropped. If they differ, a strategy similar to RED is used. Exponential-RED [LBS05] uses an exponentially increasing dropping probability. This is done by using a primal-dual algorithm, known from optimization theory, in order to compute the optimal dropping parameters for RED.

Having discussed scheduling and AQM approaches in literature, according to [Sti09], we point out their relevance for the economical operation of telecommunication networks. Next, we discuss approaches on managing bandwidth in p2p networks. Hoßfeld et al. observe p2p systems in networks with limited bandwidth capabilities like UMTS [HTA05a] and GSM with GPRS [HTA05b]. They focus mainly on UMTS- and GPRS-specific issues and not on issues arising for p2p networks resulting from peers with limited bandwidth. In [MHS+09] and [MHGS09], Mogre and Hollick extend this view and discuss the scheduling of bandwidth IEEE 802.16 mesh networks.

Some investigations on bandwidth issues in p2p overlay multicast trees have been presented in [BRP+05], [RO03], and [YP05]. The focus of these papers is on scheduling of multimedia streams in p2p networks. P2P multimedia streaming uses scheduling to decide which peer will receive the next chunk of data. These data distribution strategies are applied on top of the overlay layer. They differ from the assumptions and requirements stated for the WTD layer. Therefore, the listed approaches cannot be applied for our problem statement. General end-to-end mechanisms for rate-adaptive multicast streaming are discussed in [Rim05].

Chawathe et al. present in [CRB+03] a flow-control mechanism for Gnutella based on tokens. Each peer should generate tokens according to the rate it can process query messages. These tokens are

propagated to the peer's neighbor nodes. Each query that comes from such a peer requires the sending of a token as well in order to be processed. By adapting the rate at which tokens are generated a peer can control the number of queries it has to process. However, the solution provides only a mechanism to reduce the incoming traffic, but no further differentiation on the priority of incoming messages. Our problem statement and solution focuses on the control of the outgoing traffic in order to provide control for the delay and loss metrics in the overlay routing.

EVALUATION

We evaluated HiPNOS.KOM in PeerfactSim.KOM [KKM+07], as it supports the simulation of layered p2p systems. We extended the simulator with the WTD layer that manages the bandwidth management and message transmissions, and various metrics described next.

The goal of the evaluation is to investigate the effects of our approach, HiPNOS.KOM, which provides differentiated, configurable quality of service to various quality classes for routing in the p2p overlay. We compare the quality of HiPNOS.KOM to the reference strategies of nowadays: the FIFO scheduler and the Drop-Tail AQM mechanism. We used a Kademlia implementation according to [MM02] as p2p overlay in order to compare the effects of bandwidth strategies used in the WTD layer.

Next, we present the simulated scenario and setup. These metrics were used to measure the quality of the p2p overlay routing functionality:

- The metric *average delay per message priority (delay)* shows how the WTD layer supports the processing of relevant messages. Delay is measured from a lookup initiator to a lookup resolver in the overlay.
- The metric *average loss rate per message priority (loss)* shows which message classes are dropped, when the transmission channel of the peer is congested.

Our simulation setup consists of 10,000 peers with heterogeneous bandwidth capabilities. In [GDS+03] Gummadi et al. give a measurement study on the bandwidth capacities of peers in p2p overlay networks. We use the bandwidth distribution presented in Table 17 based on their work:

Fraction	Download capacity	Upload capacity
10%	64 kbps	64 kbps
15%	784 kbps	128 kbps
15%	2048 kbps	304 kbps
30%	3076 kbps	1024 kbps
20%	10240 kbps	2048 kbps
10%	20480 kbps	10240 kbps

Table 17: Considered Upload/Download Capacity Distribution

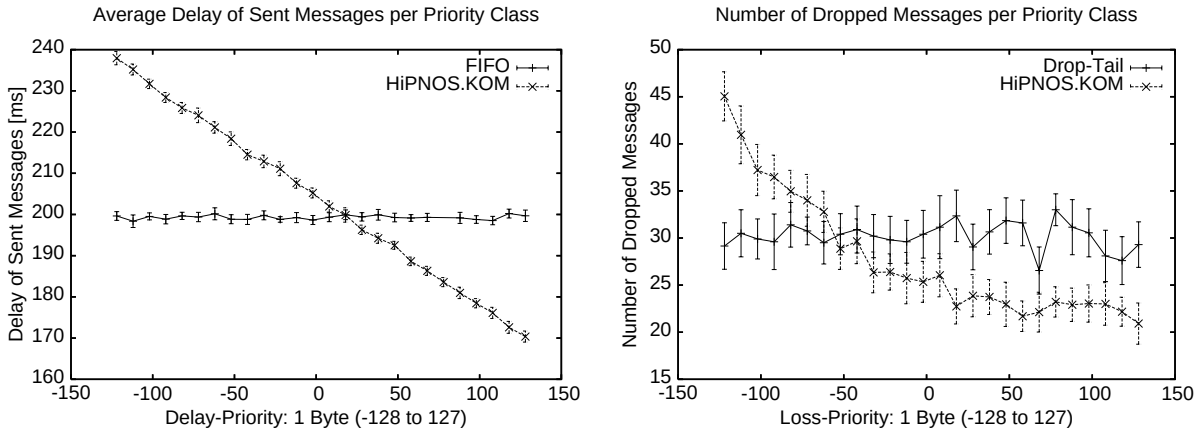
We limit the size of a peer's queue to 10 messages in order to investigate the effects of strategies handling congestion. All peers join at the beginning of the simulation. The joining phase is long enough to give each peer enough time to join. After joining each peer performs several store and lookup operations for randomly chosen objects. It is taken into account during the simulation time that peers may fail and churn exists. Each overlay comes with a set of message types. We do not define a specific priority for each of them, but we give random priorities to each message individually. We do this in order to have messages with a wide range of priorities, so that the effects of the strategies implemented in the WTD layer can be analyzed in more detail.

We vary the bandwidth management strategies using FIFO with Drop-Tail and HiPNOS.KOM. Each scenario is simulated 20 times so that we can use a confidence interval of 95%.

Evaluation Results

Next, we present the results of the simulations. We show that HiPNOS.KOM provides better service for higher prioritized messages in the context of both delay and loss. Figure 78a shows the performance of FIFO in combination with Drop-Tail and HiPNOS.KOM regarding delay in a p2p network with high traffic load. We use the bandwidth distribution presented in Table 17. Figure 78a shows the

average end-to-end delay of different priority classes. We used one byte per priority (delay and loss), so that the range is from -128 to 127. The higher the number, the higher the priority. As FIFO and Drop-Tail do not consider priorities, the graphs corresponding to them are predominantly constant. The average delay of the messages processed with HiPNOS.KOM decreases linearly as the delay priority of the messages increases. In total, both approaches provide a similar overall average delay, but HiPNOS.KOM guarantees a faster processing of messages that are more relevant to higher layers. Here again, HiPNOS.KOM enables an additional functionality by fulfilling the delay related Equation 6.2 as defined in the requirements.



(a) Delay in Routing with various Quality Classes

(b) Loss in Routing with various Quality Classes

Figure 78: Delay and Loss in Routing in Kademlia with various Quality Classes

In Figure 78b we examine the number of dropped messages in relation to the loss-priority of the messages. The high traffic load leads to the case that peers are not able to process every message, as there are more messages incoming or generated than messages can be transmitted. In our scenario approximately 3% of the messages are dropped. The Drop-Tail strategy treats every message class equally and drops loss-critical messages in the same amount as non-loss-critical messages. HiPNOS.KOM in contrast drops the messages according to their loss priorities. The number of messages dropped decreases with the increase of the loss-priority value. We observe that HiPNOS.KOM approximates the loss related Equation 6.3 presented in Section 6.2.1, which models the ideal case. As we mentioned, this aspect is very relevant in real p2p systems. In critical situations like in the join process messages should not be dropped. Using HiPNOS.KOM the WTD layer can provide guarantees that highly important messages are only dropped, when there is no other way.

Additionally, we compared the routing delay in a p2p overlay using FIFO and HiPNOS.KOM. We assigned the highest delay and loss priority to all value lookup and reply messages in the system. The other messages were marked with low priorities. Using FIFO, the total delay for user initiated actions, like value lookup operations was 1.684 seconds. In contrast to this, the average delay using HiPNOS.KOM was 1.282 seconds. By applying scheduling and AQM mechanisms on the application layer the performance of the system could be improved by 24% with minimal additional costs.

CONCLUSIONS

In this subsection, we discussed how to influence the quality of service in the routing of structured p2p systems. For that, we first showed that message flows as in the network layer do not exist in the overlay and therefore common scheduling and AQM approaches cannot be applied directly. However, providing Differentiated Services in routing in the p2p overlay is feasible in the form of prioritized messages classes.

We identified that the term *flow* needs to be adopted for p2p overlays, as it is inapplicable in its common sense. We introduced message priorities and priority classes to model the requirements of messages in terms of delay and loss. HiPNOS.KOM is a simple scheduling and AQM mechanism that

considers the characteristics of p2p overlays and provides differentiated service for the priority classes. Due to the separation of the network layers from the overlay layer by introducing the WTD layer, the solution presented is applicable for any structured p2p overlay. HiPNOS.KOM provides a mechanism to provide quality of service to several message types according to the priority settings.

To conclude, a mechanism that is configurable in order to follow the quality of service goals of the p2p system is manageable without adaptation overhead. We may generalize that through (re-)configuration of mechanisms in p2p systems the quality of service of the p2p system can be adapted to a great extent.

6.2.2 *Adaptable System-wide Quality Goals through Monitoring*

In the previous subsection, we have identified that configurable mechanisms, like the scheduling and AQM mechanism HiPNOS.KOM, provide the opportunity to affect the quality of service in the p2p system according to the current needs. However, we also identified two subsequent limitations. The question that arises now is how to set the priorities for the various message types. Which message type has what relevance for the p2p system? More generally, *which configuration for a mechanism is most suitable in order to match the current quality of service requirements*? The second limitation lies in the scope of metrics able to be influenced. Global, system-wide metrics like load balancing cannot be addressed with an localized view of the peers. Peers need to gather information about the system state, such as the average load on all peers, in order to derive local strategies to influences these global system-wide metrics in a coordinated manner. In the following, we extend the scope of a peer's view of its network and investigate how to influence metrics related to all peers, such as load balancing.

We analyze these two aspects in the application scenario of multimedia streaming in heterogeneous p2p systems. Streaming of multimedia content states strict requirements on the quality of service provided by the system, as described in [MCBM], as it requires the contribution of various resources, ranging from bandwidth to online time. Load balancing is an important design goal in creating a distributed multimedia streaming platform, which relies on the contribution of the participating nodes. Once multimedia content is published in a distributed network, users consume and redistribute the content. Having various streaming providers for the same content leads to the question of how to maintain the information of the providing peers in a distributed system and how to allocate the requests of content consumers to content providers. The redistribution of the content should be balanced on the participating peers so that the costs for the system, from which all participants benefit, are shared. Taking the heterogeneity of the nodes into account can result in a load balanced system which does not stress participants excessively but fulfills the quality of service requirements stated by the provider of the multimedia streaming system.

We identify four conflicting quality of service goals for the allocation of streaming providers to streaming consumers:

- **Load balancing:** The deviation in load distribution of all peer contributions. Peers aim at providing the same share as the other peers.
- **Download speed:** The average download time for a stream over all peers in the network. Peers aim at maximizing their download speed.
- **Incentive to stay online:** The average online time in the system. Peers that stay online support the network with their contribution.
- **Support for heterogeneity:** Weak peers should be spared and peers with more available resources should contribute more.

With this scenario, we address the two limitations that we had in the scenario of overlay bandwidth management in Subsection 6.2.1. Our scenario provides these additional challenges:

- **System-wide metrics:** Load balancing cannot be addressed with the approach discussed before. Peers must be informed about the status of all peers in order to decide on the behavioral strategies regarding load balancing.

- **Configurable, conflicting quality of service goals:** The quality of service goals loss and delay in the approach discussed earlier are independent of each other. For the new conflicting goals, we must provide strategies to allow a provider to define the relevance of these quality goals, for example, support for the heterogeneity and load balancing.

MODEL OF THE P2P STREAMING SCENARIO

We briefly describe the scenario more formally and introduce the terminology used. Let C be a streamable multimedia content, which is split up in m blocks: C_1, C_2, ..., C_m. The multimedia content blocks are stored in a network where participants provide each other with the desired content. Let P be the set of participants in the network, then we define the set $W_i \subseteq P$, $i \in \{1, ..., m\}$ for the participants that want to obtain block C_i and the set $H_i \subseteq P$, $i \in \{1, ..., m\}$ for the participants that already have the block C_i. We assume that the p2p network provides the functionality of a Distributed Hash Table, which provides in specific the KBR interface [DZD+03]. To any content block C_i, which represents an object in the DHT, a peer can be identified, which is responsible for this content block. Messages addressed to a specific object identifier are routed in the DHT to the responsible peer.

The question we focus on is what is the best strategy for matching peers from W_i to H_i according to a scoring function. The scoring function should consider the load of the specific peer (both with regard to local and contribution load), its online time and its capabilities. Here we do not focus on overlay-specific routing and characteristics, like in the section before. When applying a cost function in the matching function, the quality of the decision can be optimized. The quality of the matches is measured by the load distribution in the system or in other words how many service requests has been processed by which peer. An optimal solution results in a minimal standard deviation in the load.

We present a load balanced architecture for p2p-based multimedia streaming and a stream provider selection mechanism, which can be applied on any KBR-compliant DHT. Having the multimedia content split up in content blocks, for each block we assign a responsible peer in the DHT. This peer maintains a list of peers providing the specific content block. Requests for this block are assigned by the DHT node to the providing peers by using a scoring function. The scoring function determines the quality of a peer, by considering its capabilities (heterogeneity) and its contribution to the system (load balancing).

A majority of existing multimedia streaming systems apply the client-server paradigm, where only servers or server farms provide the content, which results in scalability issues. An increasing number of requests can only be compensated by extending the amount or capabilities of the servers.

To lessen the burden on the servers, solutions for p2p-enhanced multimedia streaming have been deployed, like Kontiki [Ver], Octoshape [Oct] or BitTorrent DNA [Bit]. In these cases, users that already received the content help to redistribute it. In contrast to these approaches, which assume that the multimedia content is generated only by one participant, upcoming multimedia streaming applications have to face the challenge of all participants in a distributed network creating, providing and consuming multimedia content.

There are two kinds of multimedia streaming applications: live streaming and video-on-demand (VoD) streaming. The main difference is that in a live system the content is generated and consumed within a small time interval. In a VoD system the videos are already pre-encoded and can be played asynchronously on demand (see [Gri00, Zin03]). Therefore the chunks can be distributed in any order and even be pre-loaded and cached in the network. Furthermore in VoD, users can seek forward and backward, which again encourages pre-caching mechanisms. We focus on p2p-based multimedia-on-demand streaming, as the freshness of movie clips or audio streams is rarely of importance in current multimedia streaming platforms like YouTube or Last.FM.

Typical solutions for p2p-based VoD are either push-based application multicast trees or pull-based mesh systems. In a push-based solution (e.g. see [DHT]), the peers are organized in application level multicast trees with the content source as root, which pushes the data towards the leaves. Challenges arise if peers fail in the tree and the corresponding subtree is not longer served with content. Conversely, in a pull-based system [AGG+07] a peer actively requests parts of data from available sources, which typically results in a mesh topology. The main benefit of pull-based solutions are lower costs, as the multicast tree maintenance is expensive, and there is higher flexibility in source selection. Different strategies for source selections can be applied. Next, we advocate and introduce a scoring function for source selection considering the various quality of service goals of the p2p system.

APPROACH: LOAD-BALANCED MULTIMEDIA STREAMING

In the following, we present our quality-aware streaming architecture for multimedia streaming which takes following design goals into account:

- Load balancing: The goal of the architecture is to provide load balancing in the selection of stream providing peers.
- Variable goal settings: Besides load balancing, the architecture should also consider the download speed and privileged service for strong contributors as well as incentives to stay online. The overall quality goal should be configurable.
- Low overhead: The costs for the allocation of streaming providers to consumers, measured in additional traffic, have to be low.
- Easy deployment: The mechanism has to be applicable in a mixed environment with peers either supporting and not supporting the mechanism.
- Overlay (DHT) independence: The solution should be applicable on any DHT providing the KBR functionality.

Our architecture assumes that the underlying multimedia streaming network provides the KBR functionality [DZD+03]. To any content block C_i, which represents an object in the DHT, a peer can be identified that is responsible for this content block. Messages addressed to a specific object identifier are routed in the DHT to the responsible peer. We do not state further requirements at the DHT, which makes our architecture generally applicable on any DHT.

The peer responsible for a specific content block C_i is called R_i, it maintains a list of all peers in H_i. Besides maintaining H_i, the responsible peer R_i also receives requests for the content blockC_i, it decides to which peer in H_i to assign the streaming tasks to. The contact address of this peer in H_i is then replied to the requesting peer in W_i. We present two approaches in the following regarding whether further information on the peers in H_i is maintained by R_i or not. Both approaches could be used in a mixed scenario, which makes the architecture easy to deploy. Using further information enables the monitoring scope of R_i and thus R_i to derive optimized decisions regarding the quality goals. We assume that the providing peers cooperate and announce their content blocks C_i at the corresponding R_i and behave according to the protocol. Thus, R_i has a partial global view on the p2p system, namely H_i.

Consuming peers (in W_i) retrieve the multimedia stream content block by content block (C_i) by requesting the contact information of streaming peers (in H_i) from the corresponding peer R_i. While consuming the multimedia file, the block index i is increased and successive blocks are (pre-)loaded. We depict in Figure 79 the aforementioned terminology and architecture for multi-goal optimizable multimedia streaming. As a next focus of our investigations we look at which information to consider and which optimization goal to use as a means for choosing a streaming provider.

Stream Provider Assignment Using a Scoring Function

Each responsible peer R_i maintains the information of the offered content blocks (C_i). This information contains in a block-centric view the contact addresses of peers in H_i. Additionally, R_i maintains the state and information of all peers in H_i as well. The peers in H_i periodically announce their state at the corresponding R_i. The following information vector IV_p is maintained per peer ($p \in H_i$) in dependency of the time $t \in T$:

- Active Tasks $I_p^{AT}(t)$: Number of tasks already performed for the system. This parameter can be used to optimize the system regarding load balancing.
- Local Tasks $I_p^{LT}(t)$: Estimation of local load (e.g. number of active processes in relation to the computing power). With this parameter, the system can adapt to the heterogeneity of the peers. Assigning fewer tasks to weaker peers keeps the system stable.
- Bandwidth quality $I_p^{Bq}(t)$: This parameter shows the network conditions of the providing peer. Peers with low bandwidth capabilities are identified and the system can adapt to unburden them. Coping with bandwidth heterogeneity is a key question in upcoming mobile streaming applications.

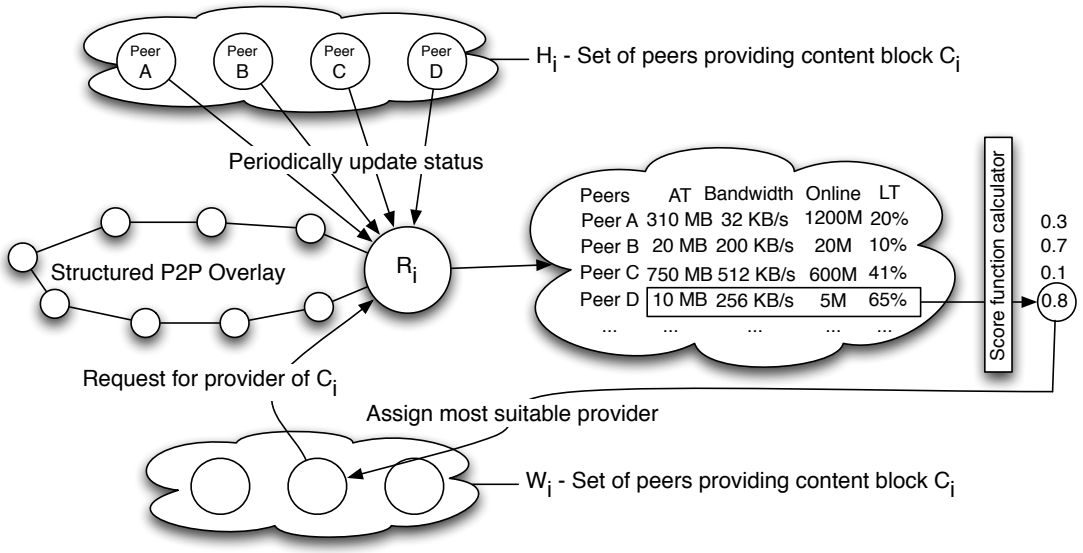


Figure 79: Architecture for Load-Balanced Multimedia Streaming

- Online Time $I_p^{Ot}(t)$: Uptime of the corresponding peer. Considering this parameter, the resources of peers staying only online for a short time can be used more intensively which benefits peers that stay online longer. This incites the peers to remain in the network.

There are two approaches for building a system that relies on distributed information. Information is used either proactively or reactively. In a push-based approach peers update their information in specific intervals proactively at the peers responsible for blocks they provide. This can lead to a high traffic overhead in a system, in which queries are rare. In a pull model peers send their status information reactively on demand. This solutions results in increased query costs, but decreases the update costs. Our solution follows the proactive approach, as the distributed multimedia streaming scenario states a high frequency of queries. The overhead can be adapted by tuning a parameter t_Δ, which is the frequency in which updates are transmitted by peer p to R_i for all $i \in \{1, ..., m\}$ with $p \in H_i$.

We assume that all peers $p \in H_i \subseteq P$ are connected to R_i for all $i \in 1, ..., m$. In a frequency of t_Δ, each peer p transmits its updated information vector IV_p to R_i. When a peer $k \in W_i$ wants to obtain a specific content block C_i it sends a query to R_i asking for a peer providing C_i. The peer R_i determines from all peers in H_i one peer to recommend, using a scoring function $c_s(p,t) : P \times Time \to \mathbb{R}$, which calculates the costs for choosing a specific peer. The scoring/costs of a peer is a value depicting the adequacy of the peer for being recommended. The peer with the lowest costs is considered the most fitting. By taking the peer characteristics into account, the peer that can provide the requested multimedia stream most in line with quality goals can be determined.

We define $c_s(p,t)$ (and s) in the following way:

$$\begin{aligned} c_s(p,t) &:= s(I_p^{AT}(t), I_p^{LT}(t), I_p^{Bq}(t), I_p^{Ot}(t)) \\ &:= \alpha_1 \cdot I_p^{AT}(t) + \alpha_2 \cdot I_p^{LT}(t) + \alpha_3 \cdot I_p^{Bq} + \alpha_4 \cdot I_p^{Ot}(t) \end{aligned} \tag{6.4}$$

The function s defines the scoring weights α_1 to α_4 for the calculation of the fitness of a peer, I_p^* are normalized values. After calculating the costs for each peer providing the desired content, the peer R_i sends a message back to the querying peer, recommending the most suitable multimedia stream provider. The subsequent streaming of multimedia content from a peer in H_i to a peer in W_i generates much more load on the peers (in H_i) than the allocation step by R_i. The balancing of the (higher) streaming load compensates for the dispatching load on the DHT peers. In the evaluation, we investigate the impact of the choice of α_1 to α_4 in the scoring function. The scoring function can be extended by various additional parameters, for example, taking QoS requirements into account as well. In addition, overlay bandwidth management mechanisms [GPK+07] as proposed earlier can be used to further increase the provided quality of service in a coordinated manner.

Stateless Assignment
We present a stateless solution that is similar to a current approach, [SP07], in p2p-based multimedia streaming. This solution assumes the existence of a responsible peer R_i per content block in the DHT as well, which maintains the information about the offered content blocks C_i. The stateless solution stores no additional information on the peer characteristics. Peers requesting a specific content block (C_i) contact R_i using the DHT and request the address of any peer $p \in H_i$. The peer R_i responds with the network address of a peer chosen randomly from the set of the peers offering C_i. After this step, peer R_i updates its internal information on H_i and W_i. This stateless solution regarding the monitoring of the peer conditions results in less traffic overhead for maintaining the lists H_i and W_i, is easy to deploy and provides load balancing due to random load dispatching as well.

Having described the both approaches, next, we summarize the solution and present the evaluation. We presented a DHT-based architecture for multimedia streaming. The multimedia content is split up in content blocks responsible peers in the DHT are assigned to individually. This peer maintains a list of peers providing the specific content block. Providing peers periodically update their status information at these responsible peers. Based on a scoring function, the responsible peer calculates which peer should be utilized to stream a requested content block. As the scoring function considers the status information of the peers, the load of the peers can be balanced and the heterogeneity of the peers taken into account. In the next step, we present the evaluation of our scoring function-based task assignment in comparison to the stateless approach.

EVALUATION

In this step, we present the evaluation of our architecture and the parameters of the scoring function. First, we describe the simulation setup and the used metrics, before we present and discuss the simulation results. Our simulation setup is inspired by the multimedia streaming requirements of today's platforms. In today's content distribution networks like BitTorrent [Coh03] there are only tens to one hundred peers requesting a file [LPY+06] at a time. We therefore focus on the streaming strategy for multimedia content up to 100 participants, the simulation setup consists of 25, 50, 75 and 100 peers. For each request, a peer in the network is chosen, which then states a query for one chunk it is looking for.

We chose the p2p simulator PeerfactSim.KOM [KKM+07] for evaluating our architecture, as it supports the simulation of layer-based p2p systems. We extended the simulator with both solutions and adapted the user layer to define the content preference distribution of the peers. The focus of the evaluation is the balancing of the load on the multimedia content provisioning peers. We investigate the load on the peers resulting from the request allocation strategy.

For the rating of the quality of the solutions, we have chosen metrics focusing on the obtained load balancing and the traffic overhead generated. We measure the *load of providing peers* in the form of the number of requests allocated to them by the peer R_i in the DHT responsible for the content block C_i. The distribution of the allocated requests shows how well the system is balanced in terms of load. We use *the standard deviation of the load distribution* as a metric for fairness in request assignment. Furthermore, we use *the difference between the costs* for using the solution based on the scoring function and the stateless solution as an indicator for the impact of the scoring function parameters α_1 to α_4. In order to investigate the impact of the update frequency t_Δ we measure the *average error rate in relation to the update interval*. We identify the trade-off between the error rate and the traffic overhead for keeping information up-to-date. We define the term *Profit* for the metric $M = c_s(p,t)$ as the ratio of additional costs for the stateless solution (RND) in comparison to the solution using the scoring function (SF):

$$\text{Profit}_M = \frac{M_{RND} - M_{SF}}{M_{RND}} \tag{6.5}$$

By measuring the profit, we identify the quality gain when using the scoring function.

Parameters α_i in the Scoring Function $c_s(\cdot,\cdot)$
In order to evaluate the load distribution of the peers, we first investigated parameters in the scoring function $c_s(p,t)$. We focused on the variation of the parameters concerning load balancing. The impact of these parameters on the function is as important as the impact of the parameters concerning the

heterogeneity of the peers. We therefore varied the impact of the load balancing parameter α_1 and the parameter for heterogeneity support, α_2, which have a sum of 50% in total. The parameters α_3 and α_4 modeling the capacities and online time of the peers are both set to 25%. However, these parameters can be tuned in order to meet the requirements of a given scenario. Table 18 shows five setups for α_i and how they affect the profit of the system according to the scoring/cost function.

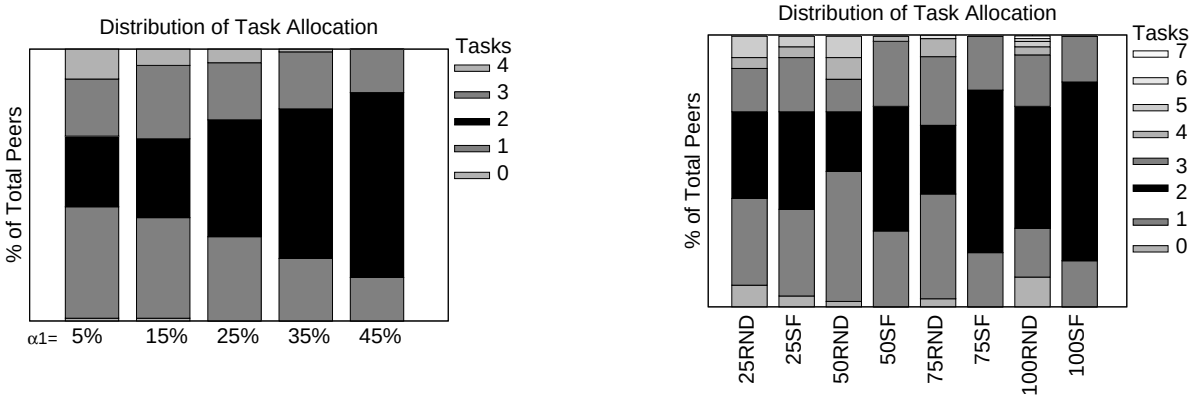
Figure 80a shows the task allocation distribution for a content block using these five setups in a scenario with 100 peers and 100 service requests in total. The Figure shows that with increasing α_1 (i.e. relevance of load balancing) the deviation in the load distribution decreases. The parameter α_1 represents the number of allocated tasks to a peer. By giving more impact on this parameter, load balancing is improved at the expense of the heterogeneity of the peers having less effect on the task allocation.

	setup1	**setup2**	**setup3**	**setup4**	**setup5**
Active Tasks (α_1)	5%	15%	25%	35%	45%
Local Tasks (α_2)	45%	35%	25%	15%	5%
Profit	65.14%	62.29%	56.26%	62.15%	76.88%

Table 18: Impact of α_i in $c_s(p,t)$ on the Profit

Variation in the Number of Peers and Requests

With the variation of α_i, we identified a suitable parameter setting with $\alpha_1 = 45\%$, $\alpha_2 = 5\%$, $\alpha_3 = 25\%$ and $\alpha_4 = 25\%$. Based on these values, we investigated the impact of the number of peers and number of requests in the system. We varied the number of peers from 25, 50, 75 to 100 and investigated the profit in a system with 25 requests (see Table 19). We also investigated the profit of the function using the scoring function in comparison to the stateless solution in a system in which the number of peers and requests are equal, for example, with 50 peers and 50 requests. The profit of our solution (by applying the scoring/cost function to the chosen peers) is depicted in Table 19. The table shows that solution based on the scoring function outperforms the stateless solution by at least 36% with regard to the fitness function $c_s(p,t)$. With increasing number of peers the profit grows to 109.84%, i.e. the decisions resulting from the reference solution cost 109.84% more in relation to the results of our solution. With increasing number of peers and requests, the profit increases as well.



(a) Impact of α_i on Load Balancing

(b) Varying Mechanism and Number of Peers

Figure 80: Effect of Parameter Choice on the Load Distribution

Next, we evaluate the load distribution in the system in relation to an increasing number of peers and requests in the system. In the multimedia streaming scenario, we aimed to take the heterogeneity of the peers into account, but still have a load balanced system. The results of the simulations are shown in Table 20 performed for 25, 50, 75 and 100 peers and the same number of requests per setup. The number of tasks assigned to a peer is also shown in Figure 80b in the rows labeled 0 to 7.

	25 peers	50 peers	75 peers	100 peers
25 resource requests	36.69%	66.25%	94.63%	109.84%
# of requests = # of peers	36.69%	62.89%	74.40%	76.88%

Table 19: Profit with varying Number of Peers

The figure shows that the deviation in the task distribution is smaller using the scoring function (SF) in allocating requests for multimedia content. Assigning randomly (RND) a peer providing the requested multimedia content, leads to more variation. This effect can be best seen in the two right columns of Table 20. We denoted the standard deviation σ in the number of allocated tasks and a metric termed *Load Balancing Saving* LBS. The metric LBS is defined as the profit in the standard deviation

$$\text{LBS} = \frac{\sigma_{\text{RND}} - \sigma_{\text{SF}}}{\sigma_{\text{RND}}} \tag{6.6}$$

The metric LBS represents the ratio by which the deviation in the number of allocated tasks per peer is decreased when using the task assignment based on the scoring solution. The Table 20 shows that the Load Balancing Saving metric increases with increasing number of peers and requests. Due to the increased number of peers the scoring function can take a higher number of candidates into account, thus, better peers can be chosen. The table shows that the Load Balancing Saving can be increased by 53% even better results can be expected in a system and scenario that involves more participating peers.

Peers	Sol.	0	1	2	3	4	5	6	7	σ	LBS
25	RND	2	8	8	4	1	2	0	0	1.32	
25	SF	1	8	9	5	1	1	0	0	1.12	0.15
50	RND	1	24	11	6	4	4	0	0	1.35	
50	SF	0	14	23	12	1	0	0	0	0.78	0.42
75	RND	2	29	19	19	5	0	1	0	1.12	
75	SF	0	15	45	15	0	0	0	0	0.64	0.43
100	RND	11	18	45	19	3	2	1	1	1.24	
100	SF	0	17	66	17	0	0	0	0	0.58	0.53

Table 20: Distribution of Task Assignment with Varying Mechanism and Number of Peers

CONCLUSIONS

We have discussed approaches to address various quality of service goals in a p2p-based multimedia streaming scenario. We presented a maintainer-based approach, in which multimedia content is split in blocks. For each block a responsible peer in the DHT manages the list of block providers and consumers. Using a scoring function on the providers allows the service provisioning to be controlled and facilitates optimization of the system behavior for load balancing and support for peer heterogeneity.

In contrast to the scenario with overlay bandwidth management, here we considered system-wide quality metrics, such as load balancing. In order to address this kind of metric, we used a global view on the set of available stream providing peers per block. This partial-global view is obtained through information that is gathered on the condition of all block providing peers. This monitoring view on the system allows for an optimized matching of providers to consumers regarding configurable quality of service goals.

We generalize the observations in respect to a more general approach for controlling and managing the quality of p2p systems. The following requirements and lessons have been derived from the investigations of the discussed approaches to influence the quality of service of p2p systems:

- Configuration-based influence on the quality of service: Depending on the configuration of a mechanism, the system adapts its strategies in order to address differentiated quality of service goals.
- Monitoring as key to address quality of service goals: Gathering all relevant information from the participating peers enables mechanisms to decide on both optimized strategies and configurations.

- Separation of mechanisms and quality of service management: Mechanisms should be generally configurable to address various quality of service goals. A dedicated quality of service management component should decide on optimized configurations for the mechanisms.

With this insights in mind, we conclude that monitoring and an automated configuration of the system based on the monitoring leads to the desired management of the quality of service in a p2p system.

6.2.3 *Autonomic Computing Approach for Managing P2P Systems*

In this subsection, we transform the conclusions from Subsection 6.2.1 and Subsection 6.2.2 and identify a viable approach. The goal of a mechanism for managing the quality of service of p2p systems is to bring and keep the monitored metrics in the predefined quality intervals. We assume that the application provider using the p2p system defines a set of quality boundaries for the metrics of the p2p system. The p2p system should then automatically adapt to these preset quality of service goals. We depict this idea in Figure 81

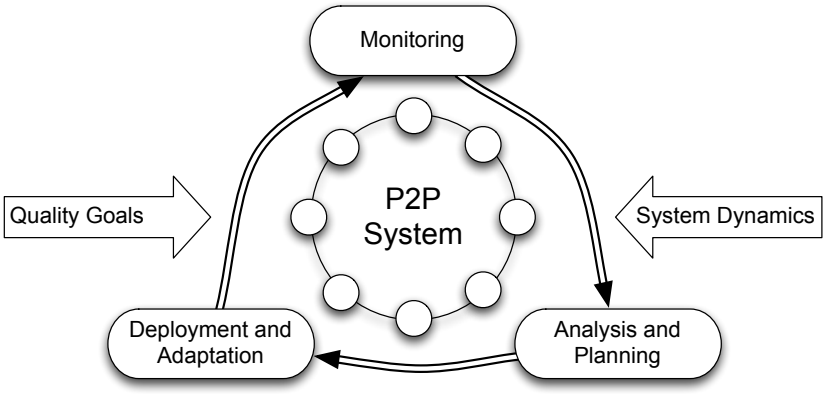


Figure 81: Management of the Quality of Service of Peer-to-Peer Systems

In Section 6.2 we elaborated that through adapting the configuration of p2p mechanisms the quality of service of the p2p system can be modified in a non-intrusive manner. Re-configuration of mechanisms is easy to apply on any p2p mechanism as it does not require the modification of internal processes. In the scenario of overlay bandwidth management we have demonstrated the efficacy of the approach. In the scenario of load balanced multimedia streaming we have shown that in order to manage system-wide metrics, such as load distribution, we need to consider system-wide information like the contribution of all peers. In order to do so, monitoring of the system status is necessary.

In Chapter 3, we presented with SkyEye.KOM an approach to monitor the quality of service of a p2p system in a distributed and lightweight manner. The solution provides us with a precise view on the various metrics measured in the p2p system. It helps to identify the current status of the system and whether the preset quality of service goals are met.

Thus, on the one hand monitoring of the p2p system gives us insights in quality violations and on the other hand re-configuration of p2p mechanisms gives us a tool at hand to coordinate and adapt the quality of service provided by a p2p system. As a bridging module, the monitored information needs to be analyzed and new configuration for the p2p system planned and deployed. This concept of monitoring, analysis, planning and enforcing the plan is described in the concept of autonomic computing, which we apply on p2p systems.

CONCEPT OF AUTONOMIC COMPUTING

The autonomic computing's primary goal is to develop software, hardware and networked systems capable of managing themselves in accordance with high-level policies from their human administrators.

Fundamentally, autonomic computing systems are inspired by strategies used in biological, social and economic systems, which like distributed systems have to deal with similar challenges of scale, complexity, heterogeneity and dynamics. Autonomic systems must be capable of monitoring and evaluating their internal and external states, and subsequently building a self-knowledge about themselves and their embedded environment. Self-knowledge in turn is the starting point for reactive and anticipatory services, such as self-controlled optimization, configuration, and problem solving.

In [Horo1] the terminology for autonomic systems is introduced, stating that a full-fledged autonomic system must possess the following eight characteristics:

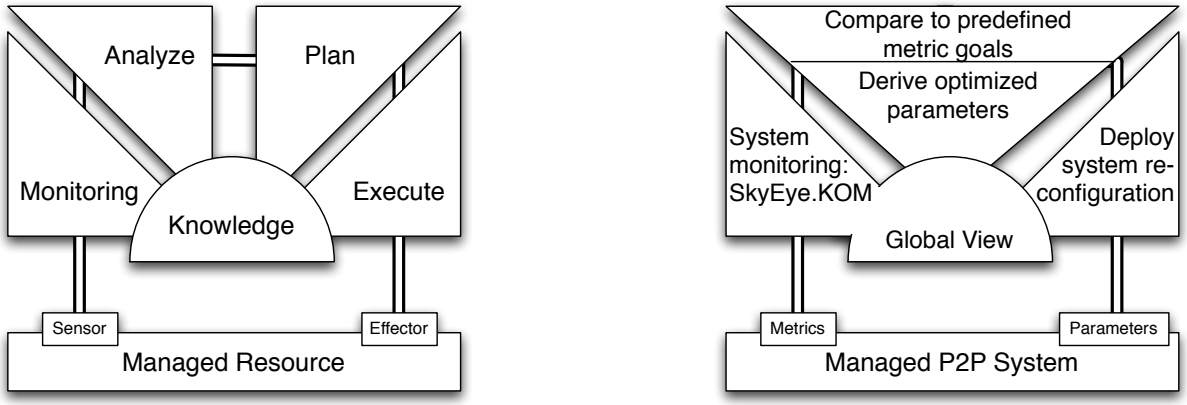
1. Self-Awareness: An autonomic application/system "knows itself" and is aware of its state and its behaviors.
2. Self-Configuring: An autonomic application/system should be able to configure and reconfigure itself under varying and unpredictable conditions.
3. Self-Optimizing: An autonomic application/system should be able to detect suboptimal behaviors and optimize itself to improve its execution.
4. Self-Healing: An autonomic application/system should be able to detect and recover from potential problems and continue to function smoothly.
5. Self-Protecting: An autonomic application/system should be capable of detecting and protecting its resources from both internal and external attacks, as well as maintaining overall system security and integrity.
6. Context-Aware: An autonomic application/system should be aware of its execution environment and be able to react to changes in the environment.
7. Open: An autonomic application/system must function in an heterogeneous world and should be portable across multiple hardware and software architectures. Consequently, it must be built on standardized and open protocols and interfaces.
8. Anticipatory: An autonomic application/system should be able to anticipate to the largest extent possible, its needs and behaviors and those of its context and consequently be able to manage itself proactively.

These eight characteristics form the basis on which autonomic computing can be described. It comprises, for example, that self-managing systems are automatically installing software and hardware components, whenever they are detected as an useful plug-and-play component. Also, they would reactively or proactively install a required software component after detecting that it is missing for a certain (future) operation (self-configuration). Furthermore, autonomic computing systems would procure behavior for restarting a failed element (self-healing), adjusting the workload distribution in accordance to the free resources as well as the self-triggered adaptation to given quality goals (self-optimization), and react with counteractive measures if an intrusion attempt was detected (self-protection).

CONTROL LOOP: MONITOR, ANALYSIS, PLAN, EXECUTE (MAPE)

The eight characteristics conducted in [Horo1] motivate a set of mandatory tasks that must be procured by every autonomic element for enabling self-* properties. Hence, it is necessary for every autonomic system implementation to realize some kind of monitoring-, analyze-, plan-, and execute (MAPE) mechanisms. We sketch these elements in Figure 82. Each of the tasks makes a contribution to the overall system's goal to seek viability and to release the human administrators from unnecessary operating, maintaining and optimization tasks.

As Figure 82 depicts, the MAPE control loop comprises four main steps (monitor, analyze, plan, execute) that are traversed in a cyclic order. All of these individual steps are supported by the fifth element, the knowledge component. The following enumeration will discuss each mandatory step in detail:



(a) General MAPE - Loop

(b) Control Loop in the Context of P2P Systems

Figure 82: Control Loop of an Autonomic System

- **Monitoring:** The main goal of the autonomic system's *monitoring* component is to provide mechanisms for collecting, aggregating, filtering and reporting details about managed resources and/or other autonomic elements. Hence, monitoring is not only a turnkey solution for having insight into the running system, but also has to be seen as a *knowledge-building* tool, delivering the facts on which the future decisions can base. With regard to p2p systems, the monitoring and building of statistics on system-specific metrics, such as lookup delay or peer load, is building the basis for future decisions.

- **Analyze:** The second building block of the MAPE loop is the *analyze* component. Its main purpose is to evaluate the current autonomic element's state against given metrics, which are composed with the help of *policy definition languages*. In general, policies represent predefined and externalized logic that determines the autonomic element's future behavior. Correspondingly, the "overall target" is to fulfill the goal policies, which are defined through a single or a set of desired states. Policies govern the decision making process for autonomic systems, indicate which resources to be monitored and how changes need to be propagated in the system. Hence, policies are static descriptors that encode the generic behavior of the system. As an example, policies can be defined to manage multiple aspects of distributed systems, such as their QoS-preferences, configuration, updating- or monitoring behavior, and auditing. With regard to p2p system goals, policies express desired quality metric goals, for example, in form of valid intervals for p2p system metrics, such as the interval [0ms, 500ms] for the lookup delay.

- **Planning:** Subsequent to the MAPE loop's analysis step, which potentially reaches the conclusion that a system's parameter has to be changed, a request to the *planning* component will be issued. The planning component's purpose is to schedule, decide, and invoke all necessary changes reactively, or even more desirably proactively. In an ideal case, planning components work proactively, having learned from former good or bad decisions, and draw conclusions upon environmental-aware experiences. In general, the planning module is faced with the problem of finding the best possible elements from the space of possible solutions. This space is a set of criteria, expressed with the help of mathematical functions, which might either be obtained through analytical processes or are tied to the service at development stage. However, these *performance functions*, also called *objective functions*, are very likely to have multiple local optima. Nevertheless, autonomic systems are supposed to seek for steady performance improvements, resulting in the fact, that *optimization algorithms* are supposed to find the (only) global optimum. Related to p2p systems, the planning component decides on suitable configurations in the space of possible configurations in relation to a given quality of service goal. As an example, we consider routing table sizes and scheduling priorities as effective parameters to influence the lookup delay of a p2p overlay. The planning component derives suitable and optimized values for these

configuration parameters, which are expected to lead to a lookup delay between, for example, oms and 500ms.

- **Execute:** The *executing* building block of the autonomic system's control loop deals with an effective usage of resources within the autonomic element itself. In the control loop's sequence, the analyzed component retrieves an exceptional circumstance (malfunction, service response-time drop, unnecessary over-provisioning, etc.) and the planning step decides reactively or proactively, the counteractive action, which are invoked in the execution step. With regard to p2p systems, the execution step consists of deploying the configuration parameters derived in the planning step to all relevant peers and employing these locally on all contacted peers.

- **Knowledge:** The autonomic system's knowledge component is a central entity, helping to improve all components' decisions. It unifies policies, restrictions, preferences, problem symptoms and -solving strategies, operating logs, and all other knowledge-building data structures. With the help of the knowledge component, autonomic managers are able to improve their decisions based on a history of earlier successful operations. Accordingly, it provides the means for accessing *self-knowledge*. Without a system's knowledge about its internal structure and functioning, other *self-* properties*, especially self-healing, -configuration, and -management, are not realizable. This means it can be determined if a current behavior is consistent or expected with respect to the environment. The knowledge component is acting as operational center on each peer, providing a container to store the monitoring information in and interfaces to the analysis and planning component operating on the monitoring information.

To conclude, the MAPE loop provides functionality to monitor the system state, analyze this status with regard to a preset quality goal and derive a suitable enforcement policy for the elements applied in the system, so that once deployed and in use by all peers the desired quality of service goal is reached and held. The MAPE loop matches the problem statement for the issue of managing the quality of service of p2p system, which has been motivated in Section 6.1. Next, we describe our approach for implementing the MAPE loop for quality management in structured p2p systems.

6.3 QUALITY MANAGEMENT FRAMEWORK FOR STRUCTURED P2P SYSTEMS

The goal of the quality management framework for structured p2p systems, termed SkyNet.KOM, is to provide a set of mechanisms that enable a p2p system to adapt its quality of service to a given goal defined by the users or a system provider. Our observations regarding the viable approaches to managing the quality of service of p2p systems in a coordinated fashion leads to requirements that are addressed in the MAPE loop. Specifically, we already derived the need for monitoring the quality of service of a p2p system and to create a knowledge base of these observations using SkyEye.KOM. We also described the potential of optimized configurations in p2p system leading to a coordinated control of the quality of service provided by the p2p system. In order to close the loop and to derive optimized configuration parameters based on the observations of the p2p system, the analysis and planning steps of the MAPE loop have to be designed and integrated.

Next, we discuss in Subsection 6.3.1 the design decision, assumptions and components of SkyNet.KOM, our framework for managing the quality of service of p2p systems and then discuss the mechanisms in detail in Subsection 6.3.2.

6.3.1 *Design Decisions*

For the creation of a quality management framework, several mechanisms have to be combined that provide functionality for monitoring, analyzing, planning and enforcing the quality of p2p systems. For these individual functionality, several design decisions and assumptions have to be made. Here, we discuss the design decisions for the individual steps required to implement the MAPE loop in p2p systems. We depict the idea in the corresponding Figure 81. Please note that we do not follow the typical order of the MAPE loop, but proceed in a more suitable order for clarity.

Monitoring
We already discussed the requirements and design decisions for the creation of a monitoring mechanism for system-specific quality of service information in p2p systems in Chapter 3. As result, we designed and evaluated SkyEye.KOM, which is very lightweight and provides a precise global view on the system status.

Execute
Viable ways for the execution of approaches that influence the quality of service were discussed in Section 6.2. We concluded that we can easily and generally influence the quality of service of a p2p system by setting appropriate configuration parameters for the p2p mechanisms used. However, we have to decide how these parameters are set in the enforcement step; either the enforcement policy, as described in the MAPE loop, describes explicitly the parameter settings (e.g. interval of keepalive checks is 20 minutes) or a more general quality goal (e.g. decrease the lookup fail ratio) is demanded. Describing general enforcement policies allows them to be considered in relation to each individual peer. However, this also requires the adaptation of the the mechanism used. On the other hand, setting parameters in the system explicitly allows for a clear analysis of which actions lead to which quality variations in the p2p system.

Thus, we adopted the type of execution policies that explicitly define the parameters each peer has to set locally. In order to derive optimized configuration parameters in the planning step, we also adapted SkyEye.KOM to gather a global view on the parameter settings used in the p2p system. This is done easily, as configuration parameters and quality metrics are both numerical values which can be aggregated in the monitoring tree.

Analysis
The analysis step compares the monitored system status with given quality of service goals in order to detect deviations. For the analysis step, we have to decide which entity should analyze this information and perform the planning step. Closely related to this question is how to communicate the derived enforcement policy. The enforcement policy is an optimized setting for the configuration parameters.

The system status and quality goals may be analyzed by a single peer, by a quorum or by all peers in the network. As the monitoring information of SkyEye.KOM is disseminated to all peers in the network, these peers could also individually analyze the global view and derive an individual configuration setting. In contrast to this, a single peer in the network (e.g. the root of the monitoring tree) could analyze the monitoring information. In order to distributed the load and responsibility, a dedicated set of peers (e.g. a quorum) may be in charge of the analysis and planning task. However, in all of these cases the same monitoring information is taken as an input resulting in the same enforcement policies. From a functional point of view, none of the approaches outperforms the other. In the case of a single decision point (e.g. in the root of the monitoring tree or in each peer) a decision is directly made locally in the peer. The quorum-based approach requires a synchronization and negotiation protocol for performing the tasks.

Once the analysis and planning steps result in an optimized parameter configuration, this decision needs to be communicated among the peers. In the case of distributed analysis and planning in each peer, this step to deploy the decision is not needed. In the second case, the root of the monitoring peer as and analyzing and planning node may use the monitoring tree to disseminate not only the metrics, but also a new optimized parameter configuration. This comes with no additional need for new protocols for deploying the decision as we have already decided to gather a global view on the parameter settings in the p2p system. Instead of providing a global view on the obtained parameter setting to the peers, it is used for the planning step by the root and an optimized parameter setting is disseminated. The quorum-based approach again needs to implement a protocol to communicate its decision either using SkyEye.KOM or another approach.

To conclude, the task of analysis and planning should be performed by the same peer. Whether it is each peer individually or the root of the monitoring tree in SkyEye.KOM, is functionally not relevant. We decided to perform these two tasks in the root of the monitoring tree, in order to allocate the load only to one peer to perform these tasks which may be complex and resource consuming. The root of the

monitoring tree gathers the monitoring information, analyzes it, derives a new enforcement policy and disseminates it with the global view on the p2p system to all peers in the network.

Planning
The task of the planning step is to derive an optimized configuration setting for all peers in the p2p system based on the monitored and planned quality of service, as well as the current configuration of the p2p system. The first design decision for this step is whether to merge this step with the analysis step or not. The analysis step concludes that a quality goal violation exists and the planning step derives a corresponding countermeasure. The analysis step itself is very easy to perform, resulting in a simple check on the monitored information. Thus, we merge these two steps due to the lack of reasonable benefits of alternatives. Once the violation of a quality goal is detected, the planning step derives and distributes a new optimized configuration setting in the p2p system. The planning step could either continuously adapt the system's configuration or only periodically. For this, we recall that a new configuration is not instantly effective. A certain period of time is needed in order to deploy the configuration in the p2p system, for it to take effect and to monitor this effect. A continuous adaptation of the system configuration is ineffective and does not consider the configuration that is currently in deployment. The effects of an older configuration are observable and basis for the analysis and planning. Thus, we activate the planning step only periodically, leaving the system time to adopt between each configuration deployments.

6.3.2 *Overview and Details on the Quality Management Framework*

In order to obtain a global view on the system state, we use SkyEye.KOM as described in Chapter 3. SkyEye.KOM provides the desired functionality to measure each peer's status and to aggregate these individual peer views to a global view on the quality of service provided by the p2p system. This global view is deployed to all peers in the p2p system. We also extend the monitoring view to the configuration parameters of all peers in order to have a knowledge basis to decide which configuration leads to an observed system status. Since all system metrics and parameters are thus propagated to the root, we implement the analysis as well as the planning step as active within the node that is currently the root of the SkyEye.KOM tree. Here, the current system state is evaluated against preset quality intervals for a set of metrics, such as the response time, average bandwidth and variance of bandwidth usage. Within the MAPE loop context, these preset quality intervals manifest the policies. In the subsequent planning step, the root decides which parameters need to be changed. We discuss various approaches regarding how to derive a suitable configuration based on the observations in the following section. As the last step in the control loop, a parameter change request is spread out to all nodes in the network. This is done effectively by attaching the change request descriptors to the metric update acknowledgments. Thus, a very low overhead is induced, as we rely on the pre-established monitoring topology and protocol of SkyEye.KOM. The peers adopt the new configuration parameters and a new cycle in the quality management loop begins. With the proposed quality management framework, SkyNet.KOM, we enable structured p2p systems to reach and hold present quality standards. We depict the interdependencies of the components in Figure 83.

Here, we summarize the assumptions for the given design decisions. For the monitoring solution SkyEye.KOM, we require a structured p2p overlay that is KBR-compatible [DZD+03]. Pastry [RD01] or Chord [SMK+01] are typical examples to name. For the analysis step, we require interfaces that enable quality of service goals may be defined. We require sensors to monitor the system state on each peer for the quality metrics for which valid intervals are defined. The scope of the sensor is in the same space as the metrics relevant for the quality of service goals. For example, to define a valid interval for the lookup delay metric as a quality of service goal, the lookup delay should be monitorable. We additionally assume that the mechanisms in use provide interfaces for monitoring and adapting their configuration parameters. To pick up the example of the overlay Pastry, we assume that parameters like the keepalive intervals or routing table sizes are configurable. Finally, we assume that the peers are well behaving and do not try to cheat or attack the system. With regard to the concept of autonomic computing, we do not aim at self-protection. Next, we discuss details of the framework related to each step in the autonomic computing cycle.

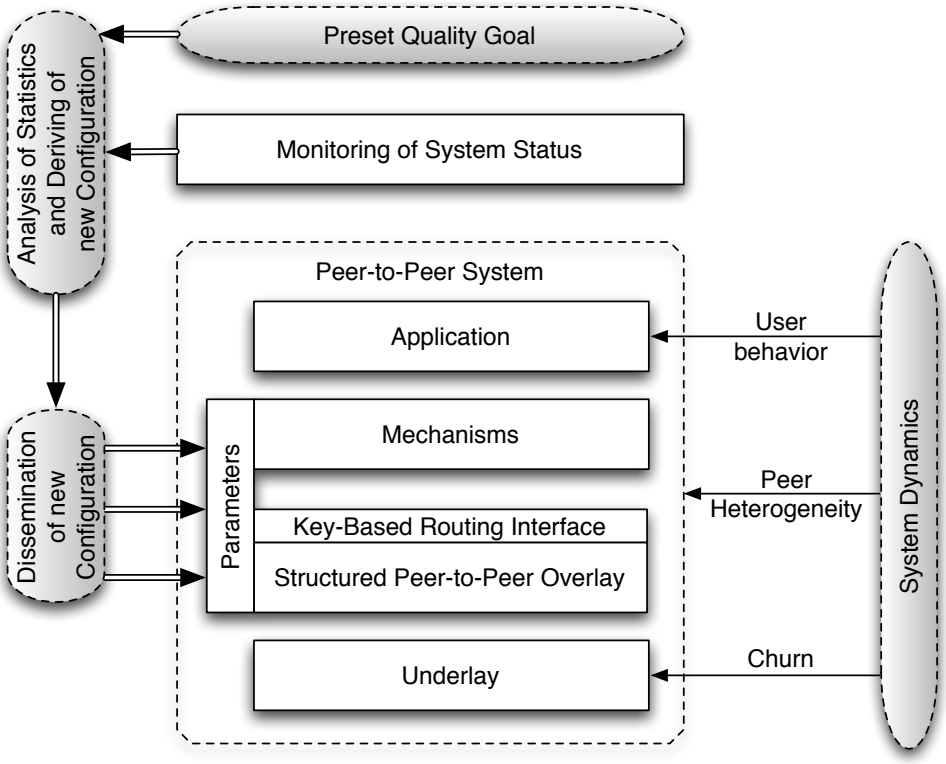
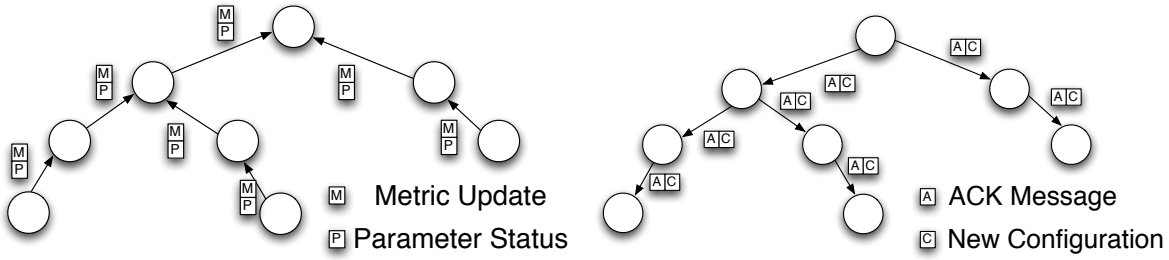


Figure 83: Layer Model of the Framework for Managing the Quality of Service of P2P Systems



(a) Gathering a Global View on the Configuration

(b) Disseminating a new Configuration to all Peers

Figure 84: Gathering and Dissemination of the Configuration of the P2P System

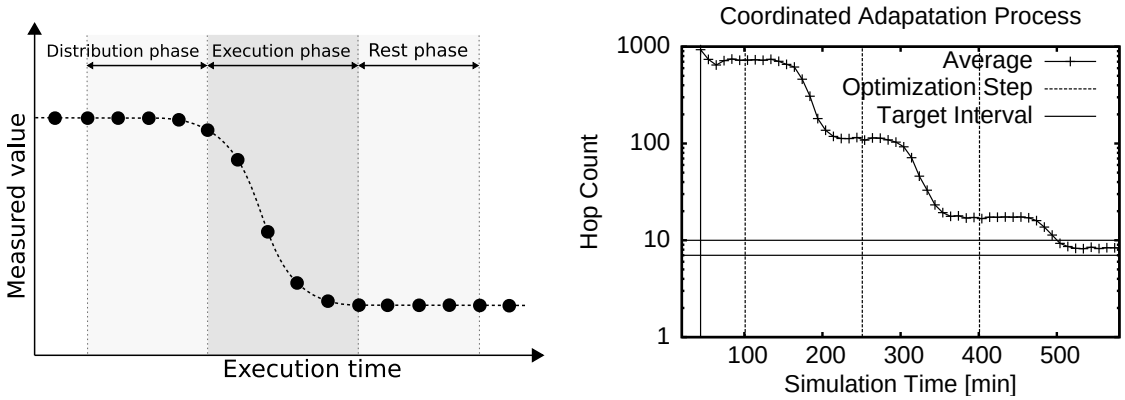
GATHERING AND DISSEMINATING CONFIGURATION PARAMETERS

In order to gather and disseminate current and planned configuration parameters in the p2p system, we treat these parameters like metrics. Thus, they are aggregated in SkyEye.KOM like the metrics before and a statistical view is generated in the root of the tree. Although information on the sum, standard deviation and other aspects in the statistical view are not of relevance for the planning component, it is easier and induces less overhead to generate this statistical information instead of applying a coordination and adaptation on SkyEye.KOM.

In contrast to the approach regarding the system metrics, this global view on the configuration of the peers is not disseminated in the p2p system. The root decides on a new parameter setting and disseminates this over the tree to all peers. Eventually, the peers in the network adopt this new configuration setting, leading to a homogeneous parameter configuration which is monitored. In Figure 84, we depict the gathering and dissemination of parameter configurations in SkyEye.KOM.

PLANNING OF STABILIZATION CONDITIONS

The planning step is operated in the root node of the SkyEye.KOM tree. We introduce a stabilization



(a) Controlled Stabilization (b) Coordinated Adaptation

Figure 85: Controlled Adaptation and Stabilization in SkyNet.KOM

phase, as depicted in Figure 85a, to give the system time to adopt the changes before initiating new changes. A stabilization phase in the planning step is used to avoid the planning of a new configuration while the current is deployed and not yet active. The stabilization phase is characterized by the slope of the measurement history. If the slope is below a certain threshold, we assume that previous changes took effect. Thus, a coordinated system adaption process is implemented, as depicted in Figure 85b.

Once the root derives an optimized parameter configuration, it disseminates it to the peers. In order for the new configuration to take effect, time passes for distributing the configuration to all peers, adopting it, taking effect in the mechanism and influencing a change in the quality of service of the p2p system, which additionally needs time to be monitored. Thus, in order to give the system time to stabilize, we introduce three phases for the adaptation of new configurations: a distribution, execution and rest phases. The execution and rest phases are differentiated by the dynamism of the relevant metric that the optimization is aiming for.

The slope of this metric defines the beginning and end of an execution phase in which the reconfiguration takes effect. The planning component observes the slope of the relevant metric and initiates new re-configuration commands only if the slope falls below a certain threshold, an event which also characterizes the entry of the rest phase. Thus, the rest phase may directly lead over to a distribution phase.

The introduction of these phases is also motivated by early evaluations in which the quality management was over-taking the desired quality ranges by inducing drastic configuration changes as the effects of mild changes were considered insufficient due to the non-instant effects.

ADAPTABLE PARAMETERS IN CHORD AND KADEMLIA

Next, we briefly describe the configurable parameters of the structured overlays Chord [SMK+01] and Kademlia [MM02]. They show potential as candidates for the quality management framework to adapt in order to affect the quality of service of the p2p system.

Chord

In Chord [SMK+01], every peer maintains a list of contacts, so-called *fingers*, which are used for routing. Typically, $O(\log(N))$ fingers are established with exponentially growing distance in the ID space of the establishing peer, leading to a lookup hop count of $O(\log(N))$ as well. In addition to these fingers, each peers maintains it predecessor and successor in the ID space, forming a ring.

We identified the following parameters, inspired by [LSM+05a], that may be configured by the quality management framework.

- **Interval of fix_finger, check_predecessor, check_successor:** Time between two checks on the liveness of a contact. More frequent checks induce traffic load, but also a quick identification of

failed peers and the new contact. With the modification of this parameters, the lookup success ratio of a p2p overlay may be modified.

- **Size of the finger table:** Number of contacts in the routing table. A high number of contacts per peer allows for faster lookups in the p2p overlay as more "shortcuts" exist. However, more maintenance overhead is induced. In [SMK+01], the protocol of Chord is explained in detail. With 2^m being the size of the ID space (e.g. 2^{160}) each peer maintains a connection to m fingers. A peer with peer ID p uses as k^{th} finger (with $k \leqslant m$) the peer responsible for the following ID:

$$\mathtt{fingerID}_p(k) = (p + 2^{k-1}) \mathrm{mod} 2^m \tag{6.7}$$

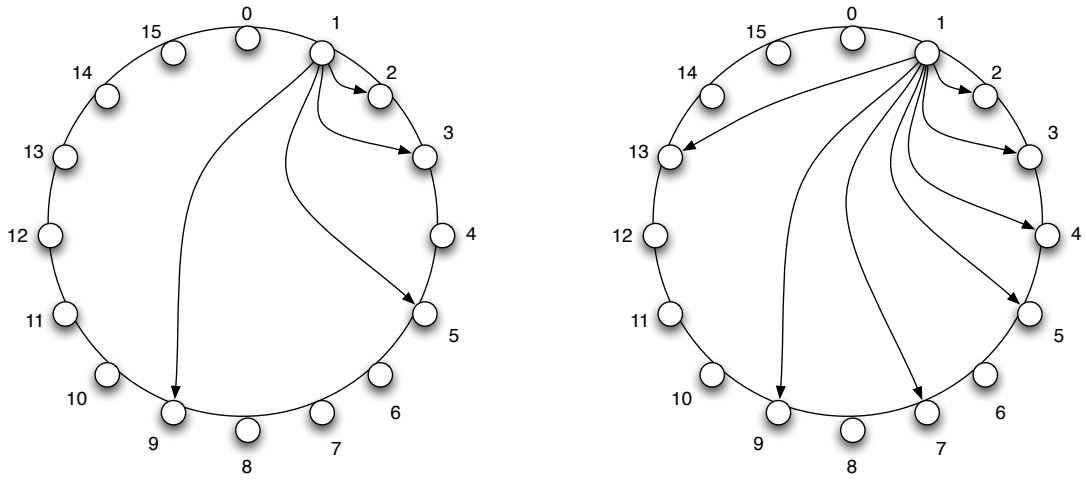
Out of the m fingers several are pointing to the same peers, being responsible for the finger IDs. This results in a reduced set of peers actually having real contacts.

In order to being able to adapt the routing table size in a consistent manner, we propose a configurable function to calculate the finger IDs in Chord that also maintains the exponential distances between finger IDs.

Let k be the index for the finger ID, 2^m the size of the ID space and FT_{size} the parameter to configure the maximum size of the finger table. Then we define a new function to determine the k^{th} finger ID by:

$$\mathtt{fingerID}(k, FT_{size}) = (p + \lceil 2^{\frac{m*(k-1)}{FT_{size}}} \rceil) \bmod 2^m \tag{6.8}$$

We depict a standard finger distribution in a Chord ring of $2^m = 16$ nodes in Figure 86a. In contrast to this, we show in Figure 86b a finger distribution with an increased value for FT_{size}.



(a) Standard FT Size ($FT_{size} = \log_2(16) = 4$)

(b) Extended FT Size ($FT_{size} = 8$)

Figure 86: Finger Distribution in Chord: Standard and Extended

Kademlia

Kademlia is a DHT which differs to Chord in design decisions regarding routing and contact maintenance. It uses an XOR metric to determine the distance between two IDs. A routing table in Kademlia is organized in k-buckets, with each peer having m of the k-buckets in the case that the ID space consists of 2^m IDs. The i^{th} k-bucket of a peer p contains up to k peers within a distance between 2^i and 2^{i+1} to the ID of peer p. The routing tables are used to implement prefix-based routing.

In order to perform a lookup for a key $\mathtt{id}_{key}$, a peer p sends α parallel queries to the peers in the k-bucket with the lowest distances to $\mathtt{id}_{key}$. These α peers reply each with n contacts from their routing table closest to $\mathtt{id}_{key}$. Out of these $n \cdot \alpha$ new contacts, the closest α nodes are picked as next destinations for a lookup message for $\mathtt{id}_{key}$. The lookup terminates when no new contacts are delivered.

We identified the following parameters:

- **Interval of keepalive messages:** Defines the period in which a peer checks the liveness of its contacts and drops them from the routing table in case of absence. A short period keeps the routing table fresh and avoids delays in lookup processing due to timeouts. However, due to the parallelism of the queries, having one slow query path does not influence the overall lookup delay to a high extent.
- **Parameter** k**:** Maximum number of contacts in a bucket which contains contacts for a distance interval. The higher the value of k, the more robust and fast the routing is as the probability to pick a good contact is increased. Additionally, the maintenance overhead is also increased.
- **Parameter** n**:** Number of contacts in a reply. A high value for n results in lower lookup hops, as more contacts are considered as next lookup destinations. In addition, the traffic induced for replies is increased.
- **Parameter** α**:** Number of parallel lookups. A high value results in lower lookup delays as parallel lookups avoid the issue of slow responding peers. However, the traffic costs for a lookup are also multiplied with the factor α.
- **Parameter** m**:** Number of buckets in the routing table. The higher the value for m, the smaller the granularity of the prefix according to which contacts are stored in a bucket. This leads to being able to pick contacts for the forwarding of lookup queries that match the prefix of the queried ID in a higher range. Thus the lookup query reaches its destination with less hops. However, the number of contacts in the routing table rises and more maintenance overhead is induced.

We see some similarities; for example, k should be always greater than n and α. The parameter m is similar to the finger table size FT_{size} in Chord. In the following, we focus on Chord.

DERIVING OPTIMIZED PARAMETER CONFIGURATIONS

The process of the planning component is depicted in Figure 87. It takes current metrics (current system statistics and environmental information), parameters (adjustable preferences) and preset valid metric intervals (quality of service goals) as input and delivers an optimized parameter configuration as output.

Figure 87: Planning Component for Optimized Parameter Settings

The simplest approach puts the system provider in charge of reacting to quality of service violations and deriving a new configuration for the parameters in the p2p system. However, this depends on the quality of expertise of the persons performing the planning step. A more advanced approach assumes rules for the planning step derived by experts that automatically change the system configuration upon detection of a quality goal violation. Here, for example, typical interdependencies may be modeled and described in adaptation rules. Finally, the most advanced approach for planning a new parameter configuration for the p2p system is by letting the planning component identify interdependencies between the quality metrics and parameter settings and to autonomously derive an optimized configuration setting. These approaches are reflected in maturity levels of autonomic systems depicted in Figure 88

However, the steps presented occur consecutively and need to be investigated individually one after another. In this dissertation, we demonstrate the feasibility of closing the control loop cycle for p2p systems with static and adaptive rules derived from experts. These rules are automatically applied, leading to an automated adaptation of the system quality.

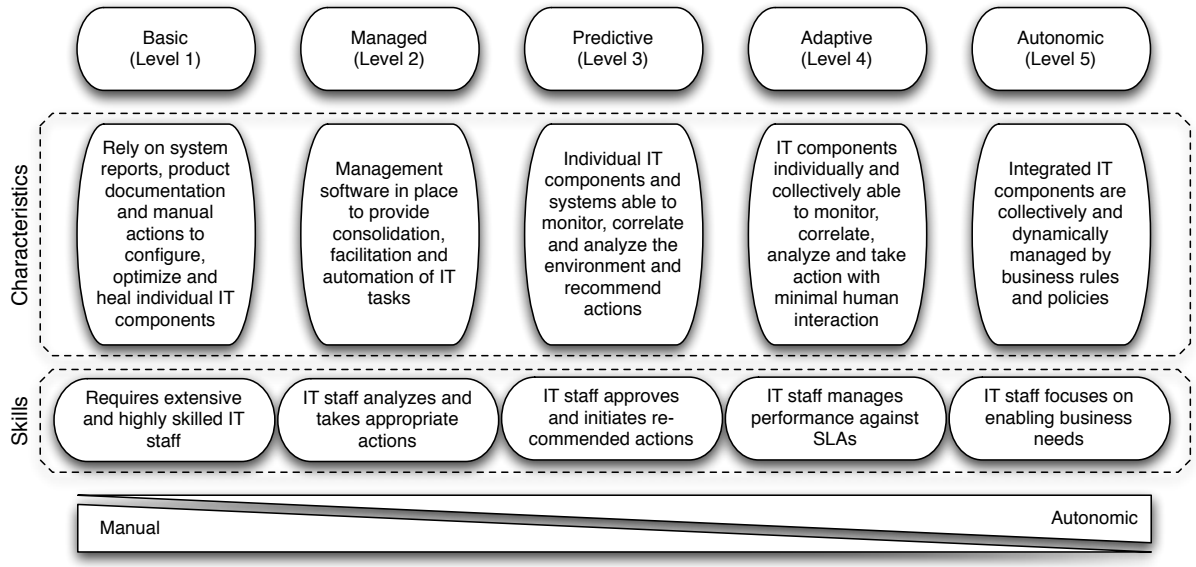


Figure 88: Autonomic System Maturity Levels

One of the possible application scenarios is, for example, to monitor the required average lookup delay or hop count in the p2p overlay. If it takes too long in relation to a preset valid interval, a new configuration is derived that forces the peers to enlarge their routing table size so that more alternative peers may be chosen for closer contact to the queried ID.

Next, we recall that quality metric goals are described as valid intervals for specific metrics. Planning rules may be adapted in order to reach these quality metric goals:

- Static rules: With the violation of a metric goal, one or a set of parameters are increased or decreased independently on the severity of the violation. For each metric goal, the influencing parameters are known and adapted by either constant (+20) or relative (+20%) factors.
- Adaptive rules: Here, the increase or decrease of the violation, for example (e.g. the difference Δ of monitored quality and quality goal) is used as a factor to adapt predetermined parameters (e.g. $+\Delta \cdot 5\%$).
- Automated rules: Here, the interdependencies between the metrics and parameters do not have to be known in advance. The system learns these interdependencies using Machine Learning on its observations and adapts the most influencing parameters accordingly.

Having discussed the details in SkyNet.KOM for implementing the autonomic computing cycle in p2p systems, in the following section we present the evaluation of SkyNet.KOM.

6.4 EVALUATION

We conclude from the overview on the parameters and metrics in Chord that the presented parameters are configurable by SkyNet.KOM, allowing for a coordinated approach to influence the lookup delay and hop count, traffic overhead and lookup success ratio. These metrics characterize the performance and costs of a structured p2p overlay.

In the scenario for demonstrating the feasibility of the approach, we use Chord and focus on the metrics hop count; we define valid quality of service intervals (e.g. $[7, 10]$) for the hop count. Regarding the parameters of Chord, only the finger table size is expected to have influence on the lookup delay. The frequency of maintenance messages influences on the other hand the lookup success ratio which is not in focus in the demonstration. In order to reach the preset goal interval of $[7, 10]$ with the monitored hop count hc_m, we adapt both static and adaptive planning rules for deriving a new finger table size FT'_{size}.

We used the following static rules:

- Case of a too large hop count, $hc_m > 10 \longrightarrow FT'_{size} = FT_{size} + 100\%$
- Case of a too small hop count, $hc_m < 7 \longrightarrow FT'_{size} = FT_{size} - 10\%$

The adaptive rules are related to the distance of the hop count metric hc_m to the desired quality state of $hc_m \in [7, 10]$. Let Δ be the difference:

$$\Delta = \lceil min(min(0, hc_m - 10), min(0, 7 - hc_m)) \rceil \tag{6.9}$$

We defined the following adaptive rules:

- Case of a too large hop count, $hc_m > 10 \longrightarrow FT'_{size} = FT_{size} + \Delta \cdot 200\%$
- Case of a too small hop count, $hc_m < 7 \longrightarrow FT'_{size} = FT_{size} - \Delta \cdot 30\%$

We evaluate the resulting quality management implementation of Chord and present its evaluation next. In particular, we are interested in the stability and reliability of the approach, as well as the convergence speed of the observed quality of service metric towards a desired quality of service interval.

6.4.1 *Simulation Setup and Workload*

The goal of the evaluation is to demonstrate the feasibility, performance and costs for the proposed quality management framework. We have chosen a scenario that is able to show the effects of managing the quality of service of a p2p system. We implemented the proposed mechanism in PeerfactSim.KOM [KKM+07], which was also used to evaluate our monitoring solution SkyEye.KOM as discussed in Chapter 4. The simulator allows for the simulation of layered p2p systems on a large scale. Thus, we simulated following layers:

- **Application layer**: We simulated a simple application that performs lookup and storage operations in order to generate workload for the p2p system.
- **Quality management framework**: We implemented SkyNet.KOM and added an interface for setting quality of service goals regarding the metric "hop count".
- **Monitoring layer**: The monitoring solution SkyEye.KOM, as described in Chapter 3, provides a fresh and precise view on the quality of service of the system and is part of the MAPE loop.
- **Structured p2p overlay**: We implemented Chord [SMK+01] as structured p2p overlay, as the most cited structured p2p overlay. We modified Chord to be able to adapt its finger table size during the running system.
- **Network layer**: For the purpose of modeling the underlying Internet and to have valid latency, loss and churn models we adopted the global network positioning model [NZ02].
- **Simulation layer**: We use the event-based simulation framework of PeerfactSim.KOM, which was first published in [KKH+06] and systematically improved since then.

The lists shows that we simulate six different layers in the p2p system. The goal of the evaluation was to show the feasibility of the proposed quality management framework by enabling the p2p system to autonomously adopt to a given quality goal.

METRICS

In the evaluation of this scenario, we focus on metrics and parameters of interest for the feasibility study:

- **Metric - average number of hops for a lookup:** We defined valid intervals for this metric in the range of [7,10].
- **Metric - average lookup time:** This metric is observable by the application user and is closely related to the number of hops.
- **Parameter - finger table size in Chord:** As the main influencing parameter, we observe its effects on the metric to manage.

SIMULATION SETUP

We have two initial setups for the Chord scenario with the goal to reach the preset quality interval of [7,10] for the average hop count. One setup starts with a configuration that results in a hop count below the desired valid interval and one starts with a configuration resulting in a hop count above the desired valid interval. We evaluated these setups while using 1000 and 10000 peers. We chose the simulation setup as described in Table 21.

General		**Monitoring**	
Simulation time:	900min	Update-Interval	30s
Number of nodes:	1000,10000	Tree degree	4
Chord		**Management**	
FixFinger Interval	10s	Hop Count Goal	[7,10]
Stabilize Interval	10s	History size	10
CheckPredecessor Interval	30s	Maximum Slope	0.2
Static Rule Approach		**Adaptive Rule Approach**	
Metric too small	$FT_{size} - 10\%$	Metric too small by $\Delta\%$	$FT_{size} - \Delta * 30\%$
Metric too large	$FT_{size} + 100\%$	Metric too small by $\Delta\%$	$FT_{size} + \Delta * 200\%$

Table 21: Simulation Setup for the Evaluation of SkyNet.KOM

We investigated a static scenario without churn the performance of the management framework with an initial configuration setup that leads to an undesired metric value. This setup shows that both static and adaptive rules act upon a detection of a quality violation. The desired quality goals are reached and held, which is observable in the static scenario. The storyline of the simulations outlined next, both for the setups with 1000 and 10000 peers. Between minute 1 and 90 all peers join and the p2p overlay stabilizes between minute 90 and 110. In this time the fingers are established in Chord. Starting from minute 110, the applications on the peers initiate lookups in the overlay using the KBR interface. This is done to generate overlay traffic so that the average hop count per lookup request can be estimated.

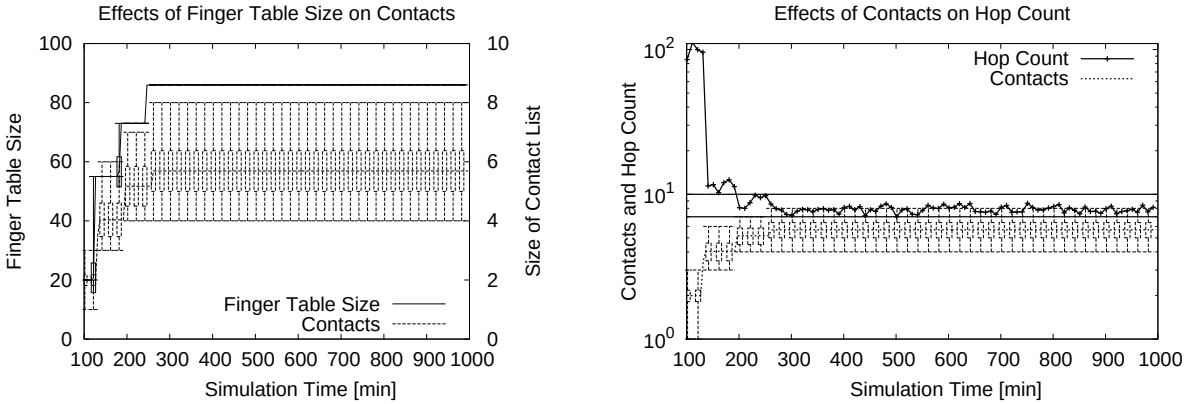
6.4.2 *Evaluation Results*

Next, we present the evaluation results in detail and discuss the observations. For each simulation setup, we present the metrics of interest over the simulation time (i.e. hop count, finger table size, lookup time and load in the system). In each simulation, we aimed to manage the p2p system to bring and keep the hop count metric within the quality interval of [7,10]. To achieve this, the system adapts automatically, through the implementation of the MAPE loop, the finger table size.

Please note that the actual number of contacts is smaller than the finger table size, as various contacts are responsible for multiple fingers in the finger table. SkyNet.KOM adapts the finger table size and influences with this the number of unique contacts in the finger table. In Figure 89a, we show this effect in a network with 1000 peers, an initial finger table size of 20 and using the adaptive rule. In the figure we observe that the adaptation of the finger table size reflects in a logarithmic relation to the number of unique contacts. The corresponding relation between the hop count and the number of unique contacts in the finger table is presented in Figure 89b. With increasing average number of unique contacts in the finger table in the p2p system, the average hop count drops.

This two interdependencies result in the coordinated management of the hop count metric through the adaptation of the configurable finger table size. Next, we discuss the evaluation results regarding both the static and adaptive rules used in SkyNet.KOM.

In Figures 90a and 90b, we depict the results for the network with 1000 peers in which we started with a too small finger table size of 20, which led to a hop count of 100. After two consecutive increases of the finger table size to 80, the hop count reached a value between 7.5 and 8.5, thus, it was in the range of the valid interval for the hop count. It can be noted that the hop count does not vary, once a suitable configuration is met, and therefore no further optimizations are initiated.



(a) Effect of Finger Table Size on Contacts

(b) Effect of Contact List Size on the Hop Count

Figure 89: Effect of the Finger Table Size on Contacts and Hop Count, N=1000, FT=20

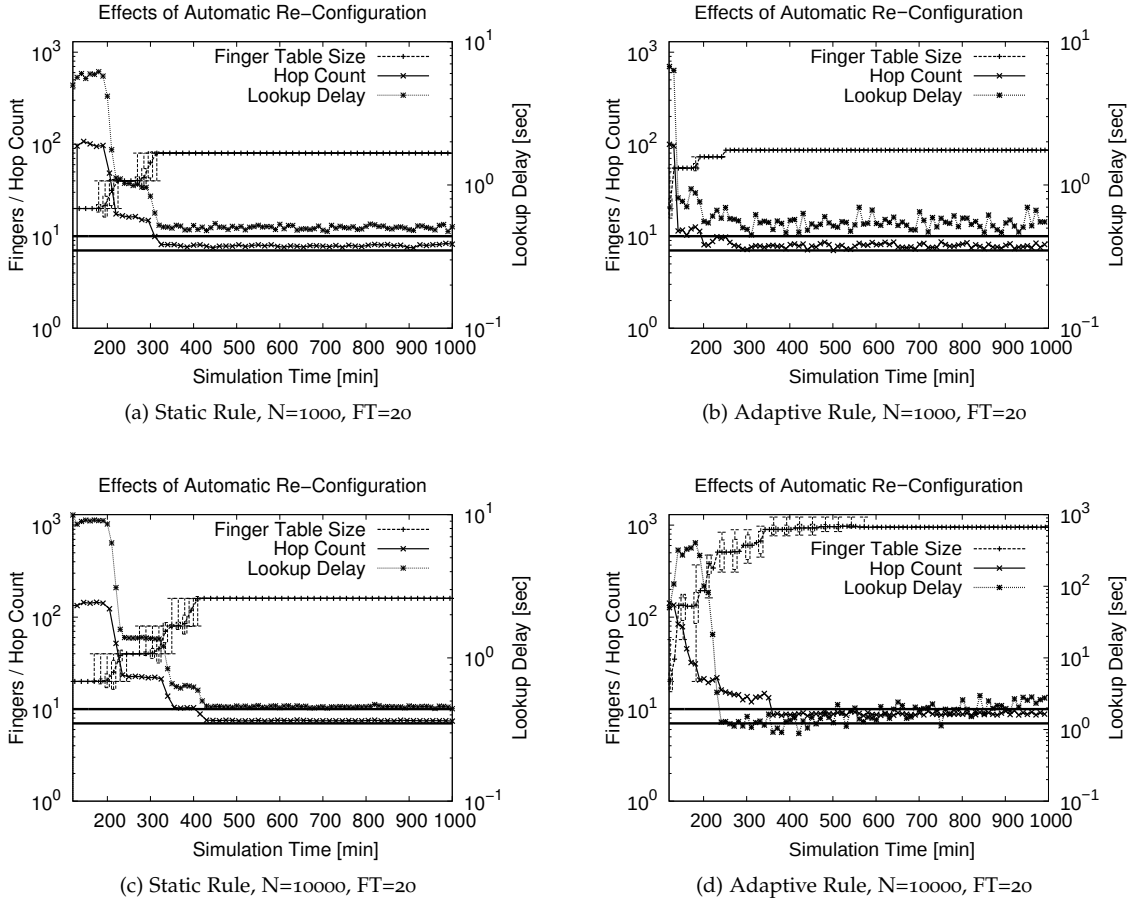
In Figures 90c and 90d, we show the results for the simulations with 10000 peers and an initial finger table size of 20 that was too small to meet the desired interval of [7,10] for the average hop count in the system. In this case, the average hop count in the system was between 130 and 140. Through three adaptation steps with each leading to an increase of 100% of the finger table size, the desired hop count of 7.5 was reached with a finger table size of 160.

In Figures 91a and 91b, we depict the results for the simulations with 1000 peers and an initial finger table size of 160 that led to a too small hop count (5.7) in relation to the desired goal of [7,10]. Through a consecutive automated decrease of the finger table size to 117 in three steps, the hop count grew to slightly over 7 and remained at this level. Thus, the desired quality goal was reached and kept.

In Figures 91c and 91d, we depict the results for the case of starting with a too small hop count in a network with 10000 peers. We used an initial finger table size of 320, which was adopted in 5 steps to 76. After this adaptation of the finger table size, the average hop count in the system was at 9.5 to 10, thus the desired quality of service interval of [7,10] was reached and held.

SkyNet.KOM adapts the desired quality in the reliable and scalable manner and allows to manage the quality of service of p2p systems. Regarding the traffic overhead, we refer to the fact, that SkyNet.KOM relies in large on SkyEye.KOM and uses its topology to derive a current snapshot of the system metrics and parameters. It further uses SkyEye.KOM to disseminate new parameter configurations to the peers in the p2p system. As a result, SkyNet.KOM does not induce additional traffic but extends the current monitoring view with additional metrics to gather, aggregate and disseminate. Thus, it increases the list of observed aggregatable values by a few, namely the parameter for configuring the finger table size and the number of unique contacts in the finger table. In SkyEye.KOM, the list of metrics and parameter to monitor is extendible and thus the evaluation of SkyEye.KOM covers the evaluation of SkyNet.KOM with regard to the induced costs.

In both simulation setups with 1000 and 10000 peers the desired quality of service intervals for the average hop count were reached and held. We evaluated both cases, and increased the hop count and decreased the hop count automatically to reach the desired quality interval. In all cases, the monitoring component derived a valid status of the observed metric and through an analysis and planning step in the root of the tree, a new adapted parameter configuration was derived in the case of violation of the quality goals. SkyNet.KOM adapted the finger table size in Chord automatically and observed the resulting changes of the hop count. Once the slope of the hop count dropped below the threshold of 0.2, the system is assumed to have stabilized. In this case and in the instance of quality of service goals not being achieved, further adaptation steps are initiated. The approach adapts the desired metric in a reliable and scalable manner and allows the quality of service of p2p systems to be managed.



(a) Static Rule, N=1000, FT=20

(b) Adaptive Rule, N=1000, FT=20

(c) Static Rule, N=10000, FT=20

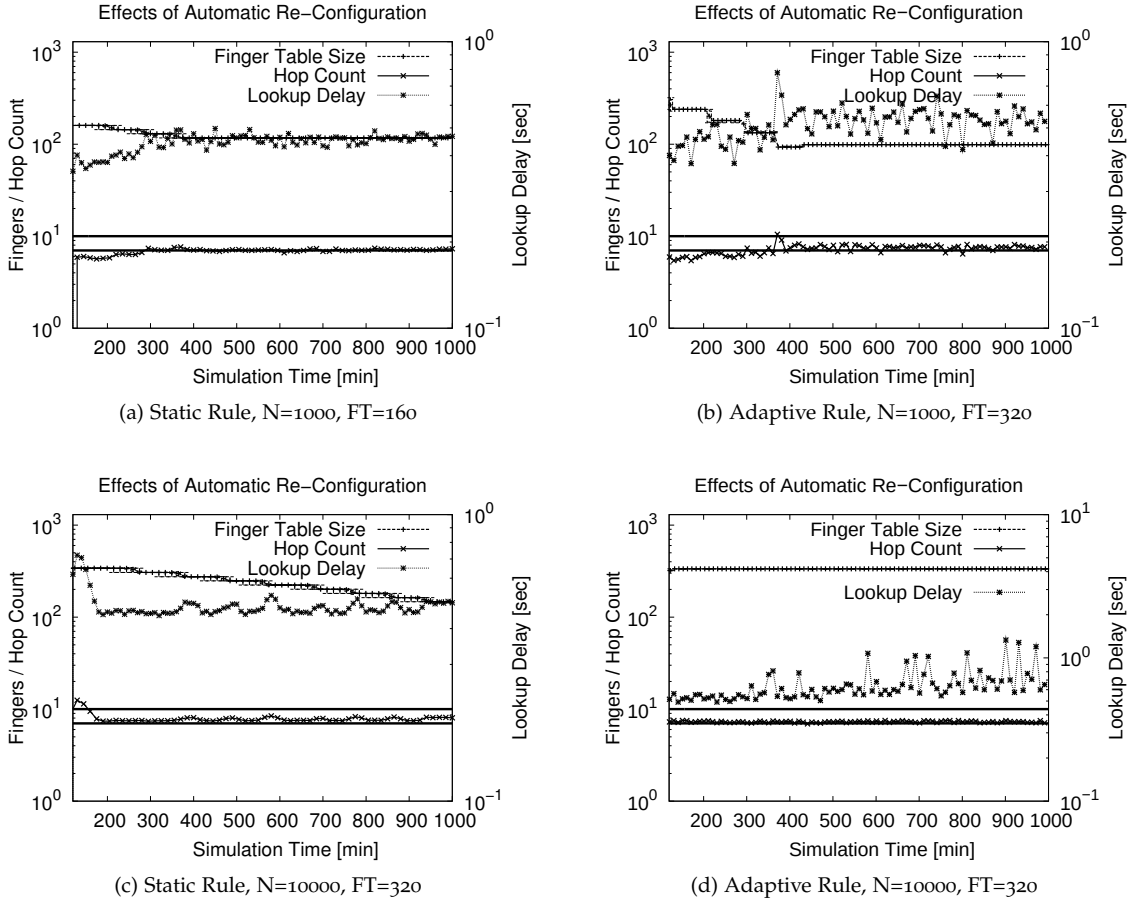
(d) Adaptive Rule, N=10000, FT=20

Figure 90: Automated Decrease of the Metric "Hop Count"

6.5 RELATED WORK

In this section, we discuss related work on the management of p2p systems. Historically, the simple network management protocol [CFSD90] was published as RFC in 1988 and has been continuously improved. It presents an approach for the centralized monitoring of network devices. Various commercial application and enterprise solutions exist for the management of server infrastructure (e.g. [CN95, LL95, WT97, RVK04, UVGS04]). They require a dedicated monitoring and management server that is informed with reports and initiates action upon the violation of infrastructural of service quality goals. In service oriented architectures, quality of service is a main topic and is addressed by Berbner et al. in [Ber07] with regard to workflow optimization, by Repp et al. in [Rep09] using an approach for monitoring and alignment of the services composition, and by Eckert et al. in [Eck09] through an intermediary which acts as intelligent aggregator and composer of services between provider and consumer. However, quality of service [GGRVA05] and quality of experience [MYSG10] are both essential goals of network technologies. Current journals (e.g. [HSS10]) pick up this trend and address network management issues also in relation to p2p systems.

For the management of large-scale distributed networks using the p2p paradigm, several approaches have been presented. The authors of [CFL+06] use a model-driven approach, while the authors of [TSS+06] focus on an integrated engineering environment for the quality-controlled creation of distributed systems. In [dMT02], the authors present an early approach for application-layer network management; the paper was published in 2002 and does not consider the characteristics of current p2p systems. In [CWS01], an overview is given of the characterization of quality-oriented management approaches in distributed systems. In recent years p2p file sharing and streaming applications gained interest in the context of the traffic load induced on the Internet. Several approaches



(a) Static Rule, N=1000, FT=160

(b) Adaptive Rule, N=1000, FT=320

(c) Static Rule, N=10000, FT=320

(d) Adaptive Rule, N=10000, FT=320

Figure 91: Automated Increase of the Metric "Hop Count"

have been presented for a p2p-based support of for the management of the underlying network, as in [BTo5, BTadG+05, PdRM+06, FSB09]. Heckmann focused in his dissertation [Hec05] on the efficiency and quality of service of the underlay.

Besides underlay optimization, further fields have been addressed with respect to management using p2p approaches. In [HRVM07, HRVM08], the data localization in p2p overlay is optimized with regard to linguistic measures. For p2p-based networked virtual environments, the authors in [BACH08, NG09] present solutions on the games state management. For the management of p2p networks, the authors of [PdRM+06] propose a two-tier architecture with top-level and mid-level management peers; however, no further details are given regarding how the mid-level managers are organized.

Aiming at the autonomic management of large-scale distributed systems, the concept of autonomic computing was proposed in [Hor01] and described in Section 6.3. In the field of service oriented architectures, several approaches have been proposed and implemented, for example, in [DPTS06, MdLH+06, MMP06]. The application of autonomic computing in this field arises from commercial needs of automated workflow optimization. Surveys on autonomic computing are given in [KHKS09, HM08]. Although, the surveys have been published recently, these papers do not cite work on autonomic computing approaches in p2p systems. In our opinion, the lack of autonomic computing approaches in the field of p2p systems arises from the fact, that successful p2p applications like file sharing and video streaming integrate optimization techniques for their specific functionality, and consequently a general view is not addressed. We believe that with the rise of general purpose p2p platforms, general management approaches will be needed and autonomic computing techniques should be applied. In this dissertation, we follow this approach.

The project FOCALE [SAL06] proposes an architecture that is closely related to the original autonomic computing vision of enhancing computer systems with self-* properties. As an enhancement,

FOCALE actively tackles the problem of dealing with complexity, derived from accommodating legacy hardware and software. The project aims at applying the idea of autonomic computing beyond several manufacturers' limits and enable interoperability. AutoI (Autonomic Internet) was an European Union funded research project aiming at "the creation of a management resource overlay with autonomic characteristics for the purposes of easy, fast and guaranteed service delivery" [ABB+08] for the future Internet. This aim is addressed through virtualization and a resource overlay that abstracts away from the current Internet infrastructure. However, the project finished in 2009, and it is arguable whether it will be put in place.

We described in Section 6.3 that the MAPE loop in autonomic computing consists of monitoring, analysis, planning and execution. The monitoring component provides a global view on the system and offers this knowledge in the knowledge plane. An overview on both monitoring and knowledge is given in Section 3.6. In the following, we discuss approaches for the implementation of an autonomic computing cycle in distributed systems, but not specifically for p2p systems.

Analyze

Deploying new policies in a self-managing system begins with obtaining a policy or goal statement. In the current state of research, policy descriptors have to be written manually by domain-aware administrators for nearly all aspects of the "autonomic" computation. This statement holds especially for the current commercially available autonomic computing systems, such as IBM Tivoli [IBM] or Fujitsu Triole [Fuj]. This is a drawback, because the idea of autonomic computing envisions a system of self-configuring and self-adapting elements that are acting in concert to reach a certain high-level goal. Research efforts, such as the policy continuum approach [vDD+06], are trying to bridge this gap, by allowing an automated transformation of high-level, mostly business-related policies towards low-level (e.g. device-dependent) goals.

The second step for introducing a new policy to an autonomic system is policy deployment. The policy deployment consists of two basic sub-components: policy ratification and policy storage. The policy ratification module certifies policies by examining the relationships with other established policies in the system, before the policy is activated or "ratified". An overview on policy checks can be found in [AKWL05]. In a full-fledged autonomic system, goal policies are provided by administrators in order to guide the system as a whole, rather than every single resource on its own. In our solution, we defined metric intervals as quality goals for a p2p system. Knowing the dependencies of the system, configuration parameters on the metric help to quickly influence the critical metrics. In order to analyze a resource's performance measurement using mathematical methods, a definition of *performance* is required. From a mathematical point of view, autonomic systems are confronted with finding an optimal solution for the performance function, which takes input values from the discrete domains of the parameters and maps them to the domains of the performance metrics. The function expresses the functional influence of parameters on the metrics as proposed in [Exn07].

In terms of performance metrics, we monitor a wide set of metrics in a p2p system with our monitoring solution SkyEye.KOM. Regarding the parameters, a simple approach is to iterate through either all or a set of possible parameter candidates and choose the one with the best performance increase. Brake et al. [BCD+08] aim at identifying tuning parameters and to provide automatic architecture of the source code to improve the quality of the resulting software architecture. The project already delivered a java-based prototype implementation. Related work and tools for software design recovery can be found in Rigi [SWM97] or CrocoPat [Bey06], who use relational programming to extract and infer model graphs and use pattern matching algorithms for analysis. Finding the dependency or performance function, which shows the impact of a parameter change, is mostly a pre-deployment step. Rabinovitch et al. [RW07] propose the development of a resource dependency model, allowing precise optimization and decision support at run-time based on best practices. In their studies they face the problem of self-optimization in database management systems suited for the academic research, since they expose more than one hundred configurable parameters and a vast number of corresponding metrics.

Planning

Subsequent to the MAPE loop's analysis step, which potentially reaches the conclusion that a system's parameter must be changed, a request to the planning component will be issued. The planning

components purpose is to schedule, decide, and invoke all necessary changes reactively, or even more desirably, makes these changes proactively. In general, the planning module is faced with the problem of finding the best possible parameters from the space of possible solutions for the performance function. These functions are very likely to have multiple local optima. Next, we highlight the most promising approaches for autonomic planning in distributed and centralized systems and give insight into the wide variety of optimization and configuration problems arising in distributed systems. An overview on the general topic of optimization algorithms for distributed algorithms can be found in [WSZG08].

To iteratively improve the system's configuration, evolutionary algorithms may be used. Evolutionary algorithms are confronted with a given population of individuals and environmental pressure, resulting in natural selection of desirable traits, which in our case are good parameter settings. Evolutionary algorithms are generic, population-based, meta-heuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, and natural selection to accomplish a randomly generated reconfiguration of the system. This may lead to an increased performance. The early work of Goldberg et al. [Gol89] paved the path for genetic algorithms. Subsequent fundamental research was conducted by Holland [Hol92]. In their genetic algorithms, they distinguish between the search space (parameters) and the problem space (metrics). The fitness (i.e. quality) of a configuration setting can be estimated, and through cross-over and mutation operations of existing configuration settings, new settings can be derived. Since in every step only the two fittest configuration settings are selected, it is expected that the resulting new configuration will obtain the best characteristics from both. Additionally, in every generation step, a mutation may occur according to a certain probability. The fundamental research achievements by Koza et al. [Koz90] laid the foundation for genetic programming, which is a systematic method for automatically solving a predefined problem. Similarly, it is seen as the group of evolutionary algorithms that automatically generate programs, algorithms, and similar constructs. The mechanism starts from a high-level statement of the program goals and automatically creates a computer program to solve the problem. Obviously, these characteristics manifest a remarkable similarity to the autonomic system's requirements for self-optimization. The applicability of genetic algorithms for network topology optimization was showcased by Khuri and Chiu [KC97]. Their goal was to determine an optimal solution for the terminal assignment problem. The objective here is to determine the minimum cost links to connect a given set of nodes (the terminals) to another disjointed set of nodes (the concentrators). Kirkwood et al. [KSS97] proved the applicability of genetic programming to solve routing problems. Their solution is capable of breeding robust routing rules for networks where links are likely to fail. Nakano and Suda present in their work [NS07] a self-managing, multi-agent system capable of adapting to changes in environmental conditions, such as network failures, varying workload conditions, and changing platform costs. In their work, network services are represented by mobile agents capable of executing different behaviors, such as replication, migration and termination. Execution is controlled by a genetic algorithm. In general, genetic algorithms appear to be a promising approach for the autonomic element's planning component. Not only that these genetic algorithms can adapt to the changing environment, but they are suited for optimization problems, where the performance function is likely to change over time.

Inspired by research on real ant colonies, Dorigo et al. [DG97] developed the idea of ant colony optimization based on cooperative agents randomly populating a computer network. In the studies, the researchers found out that these kind of multi-agent algorithms were capable of determining a solution to problems that can be reduced to finding optimal paths in graphs, as we have shown in [MGHS07]. Through stigmergy, ants (or agents) mark successful decisions and attract further ants to choose these decisions. Over time, the fastest solutions will gain more and more attention and lead to time optimized decisions. Remarkable is that ants are flagging the best solutions, while taking previously made solutions into account. This can be seen as a probabilistic multi-agent form of learning that continuously alters probability distribution functions. In [TFNK08], Tamaki et al. propose a pheromone approach for adaptive discovery and optimized creation of a sensor-network topology. In their biologically-inspired approach, the researchers propose a new methodology for constructing adjacent relations in sensor networks by using an ant colony optimization algorithm. In [UE08] and [MGHS07], a quality of service aware ant routing algorithm for wireless mesh networks is supposed. We presented in [GMHS07], [MGHS07] and [MGHS10] a security framework for ant-based routing in wireless mesh networks. In [UE08], the researchers enhance the ant routing pheromone concept with color coding, which

corresponds to a particular class of traffic. By doing so, it is possible to establish routing information, which treats packets according to the application-specific requirements on packet delay, delay jitter and bandwidth. Ant concepts were explored for computing optimal scheduling in distributed computing environments. In the work of Lorpunmanee et al. [LSACI08], the researchers use ant colony optimization for dynamic job scheduling in grid environments.

Execute

The executing building block of the autonomic system's control loop deals with an effective usage of resources within the autonomic element itself. However, since every autonomic element is part of a larger autonomic system construct, system-wide considerations must also be taken into consideration. In the control loop's sequence, the analysis component retrieves an exceptional circumstance and decides reactively or proactively the counteractive measurements, which are invoked in the execution step. We discuss next the application level of the execution step in the p2p system through three prisms: on peer resource level, p2p network level and underlying Internet level.

The peer resource management layer comprises device-specific resources, such as computing power, memory and bandwidth. Applying scheduling and active queue management approaches on these are effective to influence the performance, as we have shown in Section 6.2 and [GPK+07]. We showed the applicability of the approach in the case of emergency calls over p2p networks in [GKWS07]. It was shown that an automated improvement of resource consumptions on the node level is possible, as long as the optimization goal is known to every peer. In the special case of an emergency call, message priorities can be set per default, but an autonomic re-evaluation of message priorities is only possible if the current network situation is known to every peer in the network. Emergency-specific p2p solutions were also addressed in [BKM08a] and [BKM08b].

Another example of resource efficiency improvement for p2p systems is the work of Lo et al. [LZZ+04], who are using the peers' idle time to calculate an optimal scheduling for incidental tasks. The so-called Wave Scheduler exploits large blocks of idle time at night to provide higher QoS for deadline-driven tasks. Taking advantage of a geographic-based overlay, which includes peer separation into timezones, the researchers are capable of improving the workload distribution to a near optimum. Furthermore, Eger and Killat propose in [EK07] a distributed resource allocation algorithm in which peers control their own service rates. The algorithm is based on the congestion pricing principle known from IP networks. Every time a service is requested, a price bid for the service allocation has to be attached. Based on the offered price(-s), a service provider allocates its resources to the highest bidding customer. As a result, a spot-market for rare service capacities emerges, which takes fairness problems into account. A peer-centric view of the p2p system allows, however, only for a limited control on the behavior of the p2p system; optimization goals on the resource consumption should be globally adaptable.

Regarding the quality influencing approaches on a general p2p system level, a broader view about the behavior of the p2p system is needed. In [GKP+08] and Section 6.2, we presented an approach for quality controlled multimedia streaming with configurable optimization goals. We applied a distributed monitoring approach that gathered information on chunk providers in the p2p streaming swarm. The quality of the peers is judged based on their capabilities and contributions to the system as well as the given priority weights, which decide what metric to optimize for. In [WDÖ08], Wang et al. focus on the optimized data placement in unstructured p2p infrastructures. The researchers propose the positioning of related data chunks into p2p neighborhood groups, leading to a popularity-aware prefetch caching.

On the underlying Internet level, execution parameters (e.g. the choice of peer neighborhoods) can influence the p2p network's effectiveness, performance, and also economical aspects. However, in general an optimal network topology for p2p overlays does not exist, since the performance function is tied to the provided services' requirements. Obviously, a VoIP p2p network can have a significantly different optimal structure compared to a content delivery network. The choice of network topology configurations can be influenced by factors such as network position proximity (i.e. according to a geographic metric), the network delay, same-interest neighborhoods, the bandwidth to the neighbors or even the Internet Service Provider (ISP) affiliation. For example, Zhang et al. [ZLL08] are proposing a routing mechanism for the choice of neighborhoods in a p2p network that embraces information about the next hops' capabilities, features, reputation, and even affiliations of administrative domains into the routing decision. To achieve their goal, the researchers introduce Grouped Tapestry (GTap), a

novel Tapestry-based [ZKJ02] DHT that supports the organization into groups of participating nodes. However, a more general approach whereby other overlays may benefit as well is more desirable. From an economical point of view, the Internet layer optimization is also important for ISPs. Their optimization goal is to minimize interdomain traffic to competitors. Unfortunately, p2p networks have the biggest share in the amount of cross-domain traffic. To tackle this problem, Bindal et al. [BCC+06] propose a biased neighbor selection algorithm, which enables BitTorrent peers to choose the majority of its neighbors within the same ISP domain. As a result of the traffic locality, ISPs are no longer forced to throttle the BitTorrent traffic to control their costs. ISP-friendly neighbor selections are also addressed by P4P [XYK+08] and Ono [CB08]. Besides the capacity-specific and ISP-friendly neighbor selection, several work have also been published on location- and delay-aware neighbor selection, for example, in [KLS07] and [KLKP08].

Overall, the execution component has to ensure that invoked tasks are completed in comprehension with all restrictions and user expectations. Several strategies for neighbor selection in p2p overlays show the potential and need for a configurable neighbor selection mechanism that enables the p2p overlay to be tune with its optimization goals. For example in critical situations low delay and in general low ISP costs could be the aim. In general, the approach of autonomic computing aims to derive strategies and system configurations optimized for the current need of the p2p system.

6.6 CONCLUSIONS

In this chapter, we have motivated and presented SkyNet.KOM, a quality management framework for structured p2p systems. First, we advocated through two example scenarios that the configuration of p2p systems offers an effective interface for influencing the quality of service of a p2p system. The scenarios differed in the available knowledge on the overall p2p system (i.e. local and partial global view) and thus in the resulting viable approaches to manage the system quality. As a result we advocate that monitoring of the quality of service of a p2p system is the key way to identify quality violations and potential new configurations for the p2p system.

From the management point of view, we provided a framework that enables a structured p2p system to reach and hold preset quality intervals. Our proposed solution, SkyNet.KOM, follows the principles of autonomic computing, which consists of monitoring, analysis, planning and execution steps based on a common knowledge. The monitoring information is used to detect divergences from preset quality intervals. The proposed framework analyzes the divergences, decides on actions to be taken and sends an execution policy to all peers using the monitoring tree of SkyEye.KOM. The peers implement a corresponding strategy, for example, adapting the routing table size to a new size, in a coordinated manner. The system waits until the changes take effect and initiates further actions if the effects are not sufficient. Thus a preset quality interval, for example, for the average hop count, is reached and held.

In Chapter 3, we proposed a monitoring mechanism, SkyEye.KOM, which is applicable on any KBR-compliant DHT overlay as it reuses core functionality of the KBR interface. The monitoring mechanism provides a fresh and detailed view on the quality of the system, which is presented as a statistical summary on an expandable number of metrics. SkyNet.KOM uses SkyEye.KOM and fulfills the requirements for a monitoring and management framework for structured p2p systems in a fast and cost effective way. This framework allows to manage p2p systems, as an suitable configuration for quality goals in any given scenario can be adapted automatically.

In the next chapter, we present an application scenario, pointing out the benefits of the proposed monitoring mechanism SkyEye.KOM both for the users and system providers.

Part IV

APPLICATIONS AND CONCLUSIONS

In this part, we present in Chapter 7 an application scenario for the proposed monitoring mechanism SkyEye.KOM presented in Chapter 3. We motivate that future p2p applications and platforms must face the trend towards large-scale social applications as well as app-based application composition. For this scenario, we present a p2p platform for app-based applications in general consisting of several components for distributed data structures, reliable app-to-app and user-to-user communication, and integrated monitoring and management functionality. It provides a clear platform interface for the creation of app-based applications which is monitored in its performance. As a proof of concept, we implemented the application of online social networks on top of the p2p platform proving a wide set of rich functionality. Through a testbed evaluation we show the benefits for the users and system providers to have a tool at hand to monitor the quality of service of the p2p-based application. In Chapter 8, we draw conclusions on the dissertation and present an outlook.

7 APPLICATION SCENARIO: A PEER-TO-PEER PLATFORM FOR ONLINE SOCIAL NETWORKS

Tell me, I forget, show me, I remember, involve me, I understand.
- Carl Orff
He who hath many friends hath none.
- Aristotle

In the previous chapters, we have presented the mechanism for monitoring system- and peer-specific information as well as mechanisms for managing the quality of service of p2p systems and for reliable resource reservations. However, in order to describe the applicability of the approaches and to show the benefits for future p2p applications, we describe a use case which is both challenging as well as it is demonstrating the feasibility and potential of the proposed monitoring mechanism. In this chapter, we first advocate and engineer a general platform for p2p applications with integrated monitoring and design then in specific the application scenario of online social networks. We believe that p2p-based online social networks have the potential to gain high impact in the Internet. In order to be successful, the quality of service of such p2p platforms must be monitored and controlled. We present LifeSocial.KOM, a component-based p2p-based platform for application components (*apps*) in general and online social networks in specific. It offers a wide set of the functionality of today's web-based online social networks. Using SkyEye.KOM, we monitor the proposed platform for social online networks in a live testbed and discuss its performance and costs.

7.1 MOTIVATION

Current application providers are facing a trend towards the need for app-based platforms. Apps are small applications with limited functionality on top of a platform, such as a small game, a chat application or a reminder service. Starting with the app store of Apple, which provides a marketplace for various apps for the IPhone to Facebook with currently over 500,000 active apps on the Facebook platform, we see the trend towards reliable, extendible platforms for versatile applications. However, in the field of p2p applications, still traditional single use-case-specific applications are dominant. The overview in Section 2.1.1 has shown that besides traditional file sharing applications, applications for voice-over-IP, video live streaming and video on demand, also some applications for social file-hosting exist.

Current p2p applications typically focus just on a single functionality and use optimized mechanisms to support this single application. For example, Skype does not offer data-centric services, filesharing applications rarely support user-to-user communication, and streaming applications do not support reliable user management. Having a look at the time line and the quality of service requirements of these applications, we see that with time the need for quality of service in the applications rose.

However, in order to support a wide variety of apps and diverse quality of service needs, mechanisms should not be optimized for one single application. Next generation p2p platforms must face this need for extensibility in order to support rapid development of applications, the reusability of already established mechanisms, as well as the need for controlled quality of service provided by the p2p system and the application. We address these requirements by motivating and designing a component-based p2p platform with integrated monitoring and management modules. It combines established and evaluated individual mechanisms to a framework and provides an interface for the creation of p2p-based applications. On top of this interface, we implement in several apps the application of online social networks demonstrating the feasibility and benefits of a general p2p platform with integrated monitoring and management functionality.

We demonstrate that

- a p2p platform based on the combination of well evaluated mechanisms allows for high performance and flexible applications
- the components for monitoring and management of p2p system play a crucial role for monitoring the quality of the whole p2p system and its individual components
- through a component-based approach, p2p applications can be built quickly and of flexible functionality

REQUIREMENTS ON THE P2P APPLICATION

For the creation of the p2p platform, we aim at providing general components and interfaces that enable the creation of a wide set of applications. Nevertheless, we focus also on one specific application, which we implement, in order to show the feasibility and variety of possible functionality implementations. We state following requirements on the p2p application:

- **Relevance for the Internet community**
  The application should address a user group in the Internet which is large enough so that sufficient interest is expected. Our approach is to analyze today's popular Internet applications and to identify an application field which is suitable to be ported to a p2p-based platform.

  Additionally, not only the user view should be considered, but also the view of the application providers. The goal of the application scenario is to identify an Internet application field which is currently solved solely through client / server approaches and shows potential benefits when used on a p2p-based platform.

- **Component-based software architecture**
  The p2p application should be modular and component-based. With this requirement, individual components interoperate on dedicated interfaces and explicitly declare their dependencies. Any other component (e.g. a p2p overlay) fulfilling these dependencies may interact with the other components in the same manner. This concept enables extendible applications, which should be demonstrated through a wide set of functionality and *plugins*, which are small application components implementing these functionality.

- **Diverse quality of service requirements**
  We aim at a p2p application which not only follows one single quality of service goal (e.g. fast download speed), but diverse quality of service goals, such as data availability, low lookup delay, low bandwidth utilization. A wide set of quality of service requirements is challenging to address with traditional approaches, thus the proposed solution in the dissertation seems more adequate.

- **Automated monitoring and managing the quality of service**
  The application scenario as well as the complexity of the application should require the monitoring and management of the quality of service of the p2p application. The monitored quality is not only interesting for mechanisms adopted in the p2p application, for example to enable load balancing, but also for the users, such as to see how many users are online, as well as the p2p application provider, who may analyze the functionality of the p2p application and identify misleading trends.

- **High usage dynamics**
  In order to provide a foundation for the need for monitoring and managing the quality of service of the p2p platform, the p2p platform should be variable in its offered applications, in the usage scenario and the user behavior. Multimedia applications are very challenging according to [SN95, SN04, Ste00] with regard to quality of service and user impressions. The p2p platform should by able to host a rich set of multimedia-centric plugins.

  The usage scenario for the proposed p2p application should provide sufficient diversity in order show the potential of the proposed monitoring and management mechanisms. Thus, the scenario could, for example, describe Internet-based deployment on heterogeneous devices as well as intra-enterprise usage on high capacity PCs. And lastly, the peer behavior in the given p2p application should be diverse, resulting in moderate to strong churn and heterogeneous user access patterns.

The diverse application range, scenario and user behavior induces dynamics in the behavior of the system, which affects the quality of service of the p2p system. This quality is then to be monitored with monitoring mechanisms, such as SkyEye.KOM as proposed in Chapter 3.

In the following, we discuss online social networks as a potential application scenario. Please note that the software architecture for the p2p platform does not focus on online social networks in specific. We do not use a dedicated overlay optimized for online social network. We follow a more general approach, allowing for any kind of applications. We demonstrate the feasibility of the proposed monitoring and management mechanisms in a p2p platform that support various kinds of functionality in form of apps or plugins.

7.2 BACKGROUND ON ONLINE SOCIAL NETWORKS

Social networks and online communities are very popular nowadays and their growth is astonishing. Rarely somebody, who has not heard about MySpace, Facebook or LinkedIn. According to Alexa [Ale], these websites are among the 20th most popular websites on the world.

Online social networks allow users to create profiles, link to their friends, publish photos and status updates and various forms of user-to-user interaction. Facebook has the largest user community with more than 450 Million profiles. Also a wide set of communities exist, most of them addressing special user groups like students, researchers, musicians or business people.

From the providers point of view, it is challenging to operate a platform for online social network very profitable. Many of them need to be financed from external sources to be kept alive. The main reason for the financial problems are the high platform maintenance costs. Currently most of the online social networks are operated by the massive usage of servers. On the one hand several large databases are needed to handle the profile and multimedia content of the users. On the other hand web servers are needed to generate the page displayed to the user. As the current situation of online social networks show, the server-based IT architecture provides the desired performance.

However, in order to scale with the number of users, servers have to be operated in parallel which implies coordination, cooling and energy provision challenges. Google as a reference, operates millions of servers with a yearly increase of 500,000 servers. According to Jeff Rothschild, member of the board at Facebook, Facebook operated 10,000 web servers and 1,800 database servers in the year 2008. After the announcement of this information, Facebook acquired 100 Million US$ in order to buy up to 50,000 more servers.

Taking the rapid growth of Facebook and other online social networks into account, each user generates both high costs for storage and bandwidth. Some estimates claim that every user costs 1 US$ a year, which is a tremendous amount, leading to unprofitable platforms. These high operational costs threaten the concept of server-based platforms for online social networks and motivate a new IT paradigm lowering the costs while keeping up the performance.

Large scale networks for user interaction, however, also exist in p2p-based user communities. Skype, for example, has more than 21 Million users permanently online and had more than 1 Billion downloads up to Sept. 2008 and 450 Million users in the first quarter of 2009. This vast number of users is similar or even higher than the number of users in social networks. Despite the large number of users, Skype does not announce financial problems. This is due to shifting the maintenance and operational costs to the users by connecting them in a large-scale p2p network. Skype uses a globally decentralized user directory and provides user-to-user communication for free. However, Skype only offers voice-over-IP communication but no data-centric functionality, i.e. reliable data storage.

An example for an area of data-centric large-scale p2p applications is in the field of file-sharing applications. Since the upcoming of Napster, Gnutella, KaZaA and BitTorrent, networks of millions of users have established without paying customers. The amount of traffic generated by these applications makes up to 70 % of the Internet traffic. The efficiency of p2p-based architectures has been demonstrated in various applications both addressing tight quality requirements (Skype) as well as costly, high-traffic demands (file sharing and multimedia streaming).

We believe that the next large application area for the p2p paradigm lies in the area of social online networks, as well as that online social networks need to distribute the load on the user devices in order

to become or remain profitable. Safebook [CMS09] and PeerSoN [BSVD] address p2p-based online social networks, but need yet to provide a functional prototype and evaluation results.

An online social network provides a wide set of functionality. The underlying platform must provide the functionality of reliable storage of user data and providing reliable, quick access to it. The user related functions such as registering, login and profile creation require low delays and a reliable storage as well. The community-based functions like friend lists, groups or shared photo albums require a reliable storage and efficient distributed data structures. And finally, functionality for user-to-user interaction like chatting, voice-over-IP and gaming state real-time constraints. All of these functional parts may be described and modeled as individual components in form of apps and plugins, allowing an easy extension of the p2p application. Furthermore, all of these functionality provide a set of metrics to monitor in order to complete the view on the global status of the p2p system.

Goal of the feasibility study is to create a component-based p2p platform with plugable applications on top providing an application for online social networks. Additionally some of the proposed components should also be monitored in their quality of service. The feasibility study shows the benefits for monitoring and managing the quality of service of p2p applications, both for the users and the application providers.

7.3 LIFESOCIAL.KOM - A P2P-BASED SECURE ONLINE SOCIAL NETWORK

The goal of LifeSocial.KOM is to provide a totally distributed, extendible p2p-based platform for p2p applications providing the functionality of common online social networks with additional user collaboration tools. Common functionality in online social networks are user profiles, friend lists, photo albums, user groups, live chatting and status updates. In addition, we add the opportunity to exchange files, to collaborate in a group on a common whiteboard and to play online games, like Tic Tac Toe. Additional interaction functionality can be added in form of plugins which are either based on already existing plugins, such as a multicast plugin, or create a functionality from the scratch. The resulting application as well as the p2p network are monitored in terms of quality of service, allowing the provider to judge the quality of the p2p system, to debug and improve the platform.

7.3.1 *A General P2P Platform for P2P Applications*

A general p2p framework offers a framework of combinable functional components that may be arranged and activated to meet the current needs of the p2p applications it is used for. A modular setup of the platform allows for rapid extension of the functionality of it. The platform consists of abstract layers, built over each other, such as for routing, storage and data access. Each single layer consists of one or more software modules or components. These components provide clear interfaces and may be used by other components. Each layer provides services for the immediate upper layer and uses services provided by the immediate lower layer. Through the layer idea, the functionality of a platform is structured and allows for reusable and replaceable software components. The components, located on the same layer, can freely use services of each other using the defined components interface. The number of components, a single layer consists of, can be dynamically changed. Components with new functionality can be included on each of the existing layers. In order to extend the platform and enable a richer functionality set, new architecture layers can be added on the top of the architecture. The new architecture layers should provide the same properties in terms of extendability and interface as the existing layers, on top of which they were build on.

In the following, we state the components for a p2p platform for applications in general as depicted in Figure 92, as well as the mechanisms used to address these requirements:

- **Structured p2p overlay**: In order to interconnect all peers and to enable KBR-functionality, we use *FreePastry* [Ric]. FreePastry's main task is to interconnect the peers in a structured p2p overlay and provide reliable, consistent routing.
- **Reliable data storage**: In order to provide a reliable put/get operation for distributed data storage, we use *PAST* [DR01] which can easily be integrated with FreePastry. PAST implements replication strategies to keep data available in the p2p network in the presence of churn.

- **Storage Dispatcher**: In order to manage both local data storage as well as the access to the distributed data storage, we implement a *Storage Dispatcher*. Its main task is to identify where to store data or retrieve data based on the ID of the documents to store or retrieve, either locally or distributedly.
- **Information Cache**: In a distributed data storage environment, the retrieval of documents is delayed once accessing the network. In order to keep documents instantly available and to hide the distributed storage from the applications, we adopt an *Information Cache*.
- **Message Dispatcher**: In order to enable the addressing of inner components or plugins in the p2p application, we integrate a plugin ID-based *Message Dispatcher*, which main task is to forward messages to the destination plugins.
- **Quality monitoring component**: In order to support both the needs of the users, as well as the system providers, the system quality needs to be monitored. Here, we apply SkyEye.KOM as proposed in Chapter 3.
- **Application functionality**: The actual functionality of the application should be extendible and thus implemented in a modular, plugin-based manner. In order to address this requirement, we implement all involved modules in the software architecture based on OSGi declarative services [IOS03]. With this, applications in form of plugins may be loaded and unloaded during the runtime of the application easily.

Please note that the software architecture does not focus on online social networks in specific. A similar architecture was published in [MLS08] in close cooperation with the author, implementing a Wiki with a platform extension for p2p-based version control [MLTS08]. In our platform, the information cache as well as the message dispatcher allow for data-centric applications and applications for user-to-user communication of any kind. In order to show the benefits of this concept for p2p platforms, we address the challenging application field of online social networks. Next, we present the design and implementation of the specific layers and components of LifeSocial.KOM.

STRUCTURED P2P OVERLAY: FREEPASTRY

Pastry [RD01] is a generic routing and location overlay which is supposed to be used in p2p networks. Pastry nodes form a completely decentralized, structured p2p overlay network on top of the Internet.

Every node in the Pastry overlay gets a unique numeric identifier assigned. The identifier is called 'nodeId' and is generated randomly for each node. The values of 'nodeIDs' are uniformly distributed over the numeric space the identifiers are picked from. In Pastry, the 'nodeIDs' are 128bit values.

Except assigning the node identifiers and organizing an overlay network, Pastry offers an efficient routing functionality. Given a numeric value in the 128bit numeric space and a message, Pastry provides the function to efficiently route this message to the network node, which has an identifier numerically closest to the given numeric value. Pastry provides an efficient routing algorithm which guaranties that the message will be delivered to the recipient node in O(log N) steps, where N denotes the number of nodes participating the overlay. The routing algorithm requires that each overly node keeps track of its immediate neighbors. The neighbors are defined as nodes of the overlay network with the identifiers numerically closest to the regarded node's identifier. The applications built on top of Pastry are notified if a neighbor node leaves/joins the network or a network failure occurs.

There exist a number of implementations of Pastry which are currently available. Two of them are FreePastry [Ric] from Rice University and SimPastry/VisPastry [Mic] from Microsoft Research. Besides FreePastry, also BambooDHT [Sea04], JXTA [TAA+04] and OpenChord [LK06] exist. However, FreePastry is most used in the research community which shows its maturity and well understood functionality. Besides, a number of different research projects were started on the basis of Pastry and some useful p2p-based tools like Scribe[RKCD01] and SplitStream [CDK+03] are already implemented on top of FreePastry and available as open source implementations.

We follow a component-based approach, relying on established and evaluated building blocks which can be used by dedicated interfaces. FreePastry provides the KBR-functionality [DZD+03], which was proposed by the same authors, focusing on reliable ID-based routing.

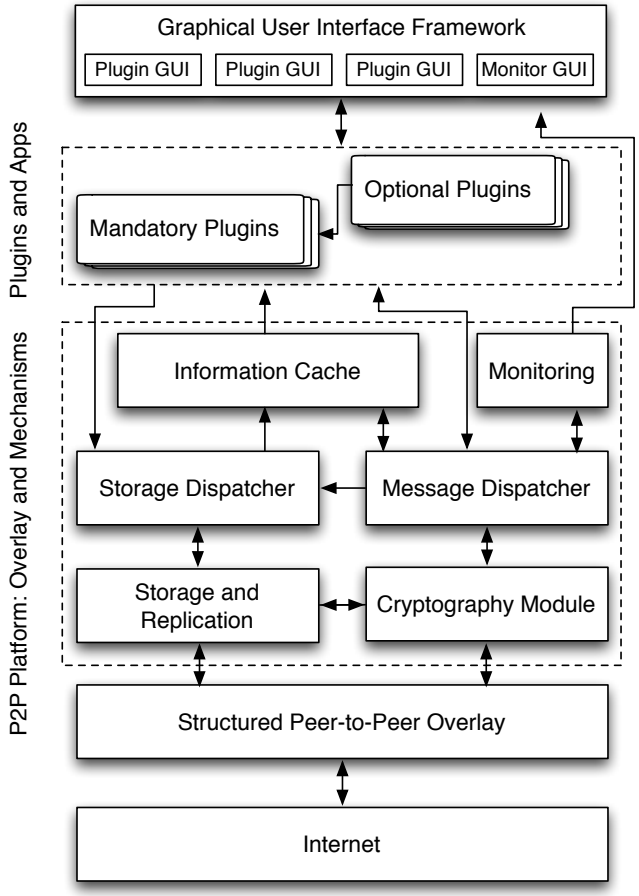


Figure 92: Component-based P2P Platform: LifeSocial.KOM

STORAGE AND REPLICATION LAYER: PAST

PAST [DR01] is a storage management and replication tool to be used in combination with FreePastry [Ric]. It provides persistent storage, caching and replication of data among Pastry nodes. Files in PAST are assigned uniformly distributed identifiers and replicas of a file are stored at nodes whose identifiers are numerically closest to the file's identifier in the ID space.

PAST is used as a storage and replication utility in the component-based p2p platform. Given an object and a storage key, which is associated with this object, PAST provides the functionality of (1) *store(Object, storage key)* and (2) *lookup(storage key)*. The first function causes PAST to store and replicate the object across multiple Pastry nodes, whereas the second function causes PAST to retrieve the object from one of the nodes, where the object is stored at. If one of these nodes leaves the network, a new replication of the data object is created on one of the nodes. The number of replications and the time intervals for replication maintenance procedures can be set by changing PAST's parameters.

We extended PAST with the support to modify data objects in place on the data storing peer. Thus, expensive data transmissions are avoided. We released the strong constraints in PAST for immutable objects and implemented the functions (3) *update(storage key, update information object)* and (4) *remove(storage key)*.

Function (3) modifies the data object associated with the given storage key. The information about the way the data object is to be modified is contained in the update information object. The updates are executed remotely without transferring the data to the initiating node. The function (4) removes the data object associated with the given storage key in the network.

In order to provide user authentication and access control, we further adapted FreePastry and PAST. Here, we briefly summarize the approach based on the terminology used in [Eck04], the specific modifications are described and evaluated in Section 7.3.3 in detail. For providing access control on the stored data, we use cryptographic Public Keys as main user identifiers in the network which are

also the node IDs. This simple step helps to uniquely identify users and to create a root for security, allowing the authentication of users, secure communication and user-based access control on the objects in the network. Any data object that is not public, is first encrypted with a symmetric cryptographic key. This symmetric key is encrypted individually with the Public Key of every read-enabled user (which are their user identifiers) and appended to the data object. The list of encrypted symmetric keys as well as the object itself is signed by the author of the object and stored with PAST. Any other node interested in the object may retrieve it (the encrypted object) from the network and validate the signature using the Public Key of the author, but only the read-enabled users are able to decrypt the symmetric key and thus the content of the object. We mention this approach for secure authentication and communication as well as access control on distributedly stored data for the completeness of the description. A more detailed view is given in Section 7.3.3. However, in the following we exclude the security consideration for the clarity of the description.

STORAGE DISPATCHER AND MESSAGE DISPATCHER

The Storage Dispatcher provides storage services for platform-specific data objects. All data objects and messages in LifeSocial.KOM extend a common class termed *SharedItem*. This class (SharedItem) contains attributes representing storage key and header of the data object and supplies a data object with properties which make it to a storable object. Given a storage key and a data object associated with this key, the Storage Dispatcher performs efficient storage, retrieval, update or removal operations on the data object. When initialized, the Storage Dispatcher reserves a certain amount of hard disk memory on the local machine for the p2p system.

The Message Dispatcher module is responsible for sending and receiving messages between users and is used by plugins such as live chat and messaging to offer direct communication capabilities to the users. The Message Dispatcher allows messages to be stored in the network for later retrieval in case the recipient is not online at the moment the message was sent. Additionally, a history of the messages sent and received by the user is supported.

INFORMATION CACHE

The Information Cache acts as a cache for objects requested by higher layers from the distributed storage. Objects, i.e. data objects or stored messages, which are often requested and do not change in the meantime do not need to be retrieved from the distributed storage every time a requests occurs. They are kept in the cache and are available to the higher layers, so that subsequent requests are served locally, avoiding the generation of extra traffic. The size of the cache is configurable, as well as the applied caching strategy. Here, we use the least recently used strategy.

The Information Cache plays an important role in the p2p-based architecture. It accumulates all the requests for data objects the application states and offers a clear interface for upper layers for accessing the data in the cache and the network. Some plugin-specific data packages can be a combination of a set of single data objects. Single data objects are stored separately in the network, which means that they can be physically located on different network nodes. In order to retrieve such a data package from the storage, all single data objects have to be retrieved. Some of the requested objects can be delayed and arrive later then the others.

With the Information Cache, plugins in the upper layer do not have to handle asynchronous events happening in the p2p network. Plugins can decide what data object they want to have at each point of time and get it from the cache. The data object is either available, already requested or not available. If the data object is not in the cache, the plugin does not do anything more, it needs to request it later on. In the meanwhile, the cache initiates a lookup for the desired object and processes the irregularly incoming data.

MONITORING THE QUALITY OF SERVICE OF LIFESOCIAL.KOM

Our solution for monitoring of p2p systems, SkyEye.KOM, was presented in Chapter 3. It requires a structured p2p overlay as substrate and provides the system providers and users with current statistics about the quality of the system. It generates a statistical view on an extendible list of various metrics, such as the object retrieval delays, storage, bandwidth and CPU consumption, number of nodes and various metrics more regarding the performance and costs of the system. For each of these metrics all

peers are taken into account to create a view on the global average, minimum and maximum values, as well as the sum, standard deviations and variances of the values.

The prototype implementation of SkyEye.KOM, which was evaluated in Chapter 4, was extended to provide also a view of the performance and behavior of the p2p platform. Therefore, new *sensors* were added that monitor LifeSocial.KOM-specific data, in specific the Information Cache, the Storage and Message Dispatcher. The list of new metrics that are monitored is given in Table 22.

We added as metrics the total and current amount of objects processed in the cache, the total number of messages exchanged overall and per plugin as well as the disk space usage by LifeSocial.KOM. The new metrics related to LifeSocial.KOM are listed in Table 22. These new metrics allow the performance of the LifeSocial.KOM platform to be evaluated and helps identifying eventual performance bottlenecks.

A	S	P	Statistic - (A: Analytic Model, S: Simulation View, P: Prototype)	Unit
LifeSocial.KOM-specific - 5 values for all statistics: Count, Min, Max, Sum, Sum of Squares				
		X	Total overhead of all plugins	1/s,KB/s
		X	Per plugin overhead	1/s,KB/s
		X	SharedItems ever stored in cache	#
		X	LifeSocial.KOM objects ever stored in the cache	#
		X	Items currently stored in cache	#
		X	Items ever stored in cache for plugin X	#

Table 22: List of Statistics Monitored in LifeSocial.KOM

APPLICATION-SPECIFIC PLUGINS

Plugins are stand-alone applications or apps on top of the platform which can be mostly used independently of other plugins. However, there are mandatory plugins which belong to the core functionality of the application (e.g. a plugin responsible for user account management) and their correct execution is a prerequisite for the execution of other plugins. All other plugins, which are not necessary for the online social network are optional in this application scenario. For them, dependencies may be defined as well. A plugin sends and receives plugin-specific messages, creates and stores plugin-specific data objects, retrieves data objects from the shared memory. Plugins are expected to interact directly with the user interface. To each plugin a corresponding user interface has been designed. LifeSocial.KOM includes a set of plugins implementing an application for online social networks with rich functionality.

Before presenting some of the plugins for the online social network, we first describe how plugins are used and which functionality is provided from the p2p platform for plugins to communicate and interact with each other.

7.3.2 *General Data Structures and Communication*

With FreePastry and PAST, objects can be stored and retrieved from the network based on their identifier very quickly with low delay. For messaging between peers, FreePastry provides a simple identifier-based routing of messages. In LifeSocial.KOM, we also use a Message Dispatcher which dispatches incoming messages according to the message type to the internal plugins, e.g. assigning arriving chat messages to the chat plugin. In the following, we describe how plugins may communicate with each other in order to implement their functionality.

In general, the p2p platform provides two concepts for the plugins in the p2p system to communicate, over direct messages or over the shared memory. Figure 93 shows the communication concepts for plugins.

PLUGIN COMMUNICATION TYPES

The entire functionality of an online social network is implemented by a set of plugins. A set of mandatory plugins build the core functionality of the application. Every plugin operates on its own plugin-specific data types. Plugin messages are sent and received over the Message Dispatcher. Storing and retrieving data objects is accomplished by the Storage Dispatcher. Since different plugins operate

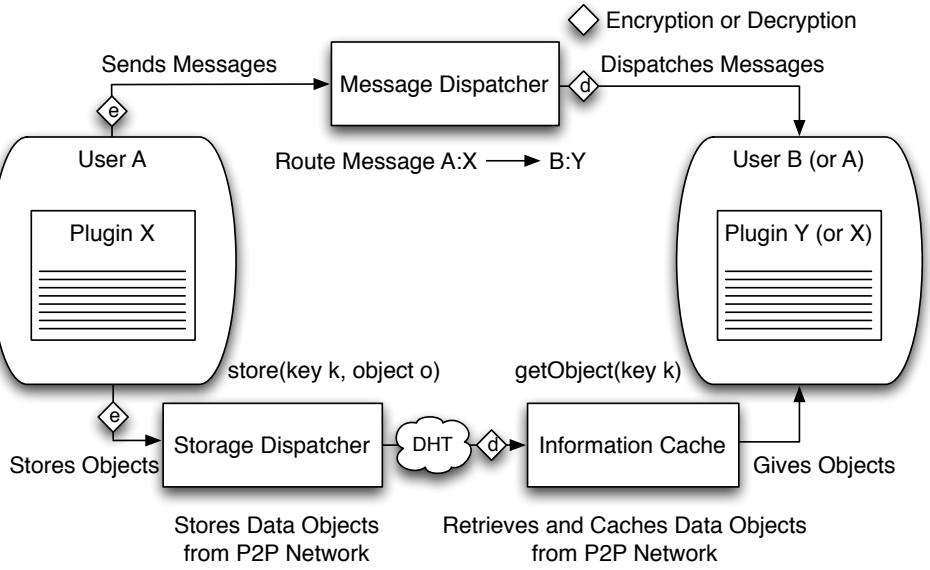


Figure 93: Plugin to Plugin Communication

with different data, we attach header information to the data objects. A header holds the identifier of the plugin to which the data object belongs to and two more fields: the data object identifier and item identifier. These second field is designed for plugin internal usage. It can be used to determine different types of plugin data, data object roles or functions. Additionally, each data object has a unique storage key which is used in the case that messages cannot be delivered and are stored in the p2p network. Both messages and storable data objects implement a common interface which enables them to be stored, termed SharedItem. The concept of *SharedItems* is depicted in Figure 94.

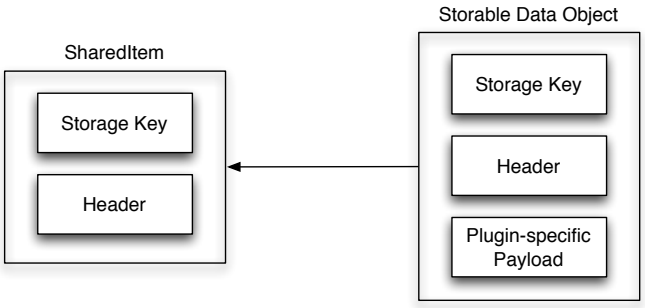


Figure 94: SharedItem and Plugin Data Object

Plugins interact with the Massage Dispatcher to send messages to a dedicated plugin on another peer. This concept is similar to the introduction of ports at the transport layer in contrast to their absence in the network layer. Having the opportunity of plugin-addressable messaging allows to create easily protocols on the plugin level, e.g. for user-to-user chatting, multicasting or gaming.

The SharedItems are also used and created in the Storage Dispatcher which marks data objects with header information related to the data content. The Storage Dispatcher then uses the storage and replication module PAST to store and retrieve SharedItems. One single SharedItem may contain two fold information. One the one hand, the data contained may be a document or immutable information. On the other hand, the SharedItem could also contain links to other SharedItems. Thus, data objects may be built modular as well and allow for more fine granular modifications and complex distributed data structures. In the following, we present a distributed data structure that supports a wide set of application scenarios.

DISTRIBUTED LINKED LISTS

The data structure, we present now, is termed *distributed linked list*. The main idea of a distributed linked list can be stated as follows; a distributed linked list is a directed graph consisting of a set of storable objects (data nodes). All data nodes are classified into two groups, entry data nodes and internal data nodes. An entry data node is a data node whose storage key is well known and can be generated as the need arises (e.g. $hash(username + \text{“\_profile”})$). An internal data node is a data node whose storage key is randomly generated and is previously unknown (e.g. 0x2311F1B). Every data node (both entry data nodes and internal data nodes) may contain a list of storage keys as their payload. Every storage key refers to a data node within the graph, which can hold a list of storage keys and/or other data as its payload. A data node containing no storage keys of other data nodes is called a leaf data node.

Using this concept, different linked data structures can be built and stored in the network. We depict the concept in Figure 95. The structure of distributed linked lists allows to navigate through the shared storage by using the storage keys stored in the objects. Once we built a storage key of an entry data node, we can get all data objects which are reachable from this data node.

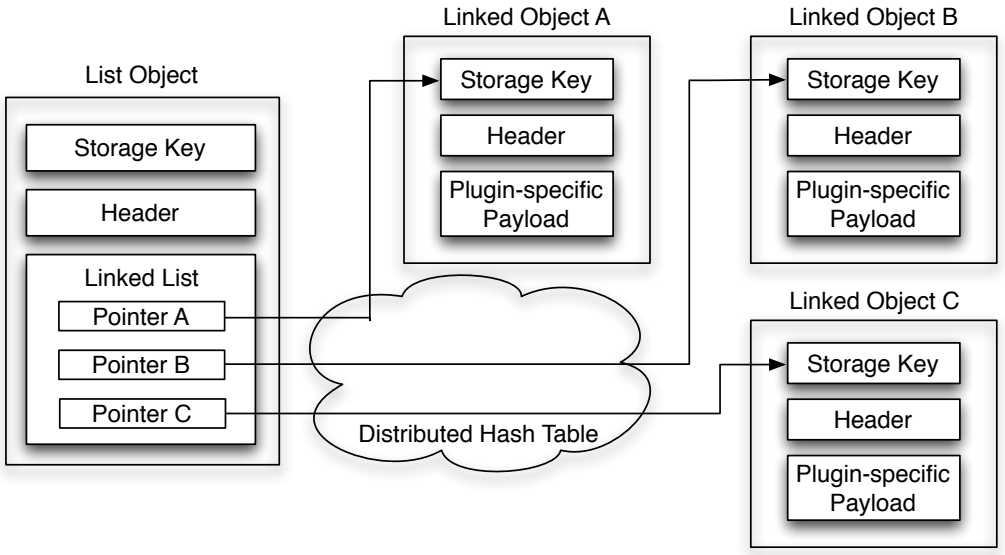


Figure 95: Concept of Distributed Linked List

We take a look on an example of a distributed linked list, which demonstrates a possible application of this data structure. We consider following situation: every user of an online community owns his/her personal images, which he/she would like to publish within the platform to enable other users and friends to view them. All images are sorted into albums. An album holds some information about the image collection it contains along with the image collection itself.

We depict the distributed linked list for this example in Figure 96 and sketch the distributed storage in Figure 97. Every photo album and corresponding image is stored in separate objects under randomly generated unique storage key. The payload of every album object is a list of storage keys which refer to a set of image objects, which are registered to be in this particular album. A dedicated data object holds a list of storage keys which refer to a set of album objects, all of which belong to a single user. The storage key of this dedicated data object is well known and can be build by hashing the string representation of the owner's unique user name (e.g. “alice”) concatenated with a well known String token (e.g. "albums-list") resulting in “hash(alice-albums-list)”. This data object acts as an entry data node. Figure 96 demonstrates this data structure.

Distributed linked list appear as a flexible and simple concept which allows to effectively store and retrieve any kinds of complex and coherent data objects. For example, a data object can hold storage keys of undelivered messages addressed to a certain user. The list can be modified when new undelivered messages appear or when some of the undelivered messages are delivered and can be removed from the memory. Other examples envision friends lists containing pointers to friend profiles, or group lists containing pointers to profiles of group members as well as to group images. In addition, large data

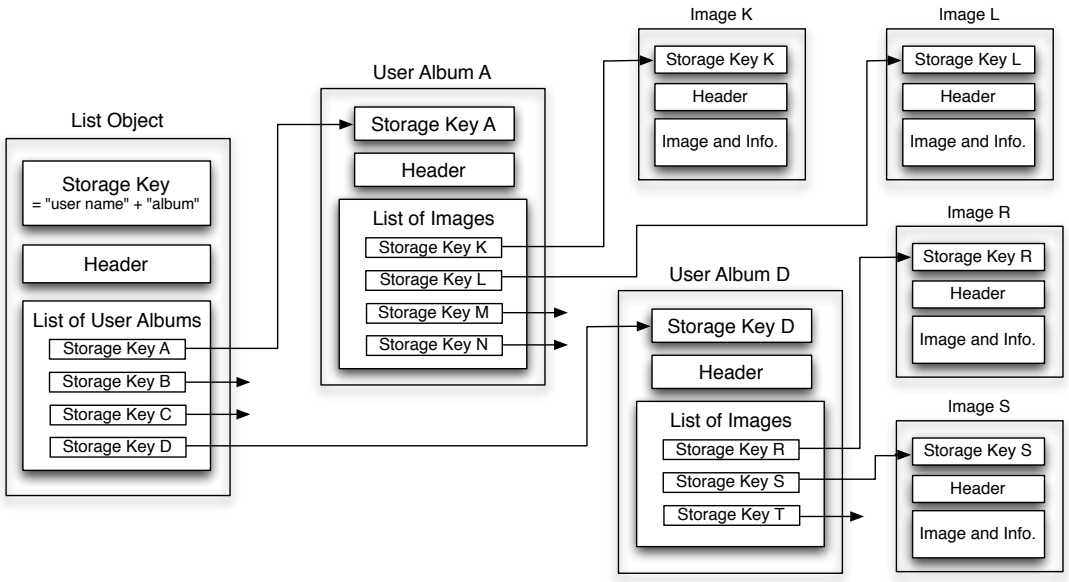


Figure 96: Example of Distributed Linked List: Photo Albums

structures can be stored so distributedly in the p2p network. Here, we point out the benefits of having an Information Cache which manages the data requests as well as composes and offers complex data structures retrieval from the p2p network.

A distributed linked list meets all of the requirements and represents a flexible data structure that allows to maintain the data stored in a p2p in a desired way. The entry points, i.e. storage keys, for a distributed linked list, e.g. the lists of albums of a specific user, have to be either publicly available or easy to construct. We follow the latter approach and construct the storage keys through the concatenation of specific words, as the username and plugins-specific terms like “useralbums” or “friends”.

To conclude, plugins in the p2p platform may either communicate directly with each other or rely on (complex) data structures which are distributedly stored in the p2p network.

DISCUSSION ON EVENT PROPAGATION

Regarding the access on distributedly stored data, we identified two approaches to handle responses to data lookup. We use the term *event propagation* to refer to the process that handles incoming events that have been previously requested. An event can be an arriving data object, a message or any other information coming from the overlay network layer.

Two different approaches were identified for event propagation in a p2p environment with regard to a p2p application. First, events may be stored in an intermediate layer, such as the Information Cache, where they can be accessed by the interested entities. Second, the events may be directly propagated to the receiver entity. A receiver entity can be a plugin or a user interface component for a plugin.

In the first approach all events from the overlay layer are propagated up to the Information Cache layer where they can be accessed by the plugins which track the corresponding event, for example, by knowing the object ID of the requested data object. Plugins which are interested in the result of the event, periodically query the Information Cache for a result. The Information Cache replies either with the result of the event, its pending status or a note on the absence of the looked up data in the p2p overlay after a timeout. This allows plugins to decide what events has to be processed and what events may be not relevant any more. The disadvantage of this approach is the CPU overhead caused by the periodical queries sent by a plugin to the Information Cache as well as a delay of event notification depending on the polling frequency.

The advantages of this approach can be stated as follows. Plugins and user interface components are not forced to react to events propagated from lower layers, they are decoupled from the asynchronous occurrence of the events. This makes the internal plugin data management much easier in comparison

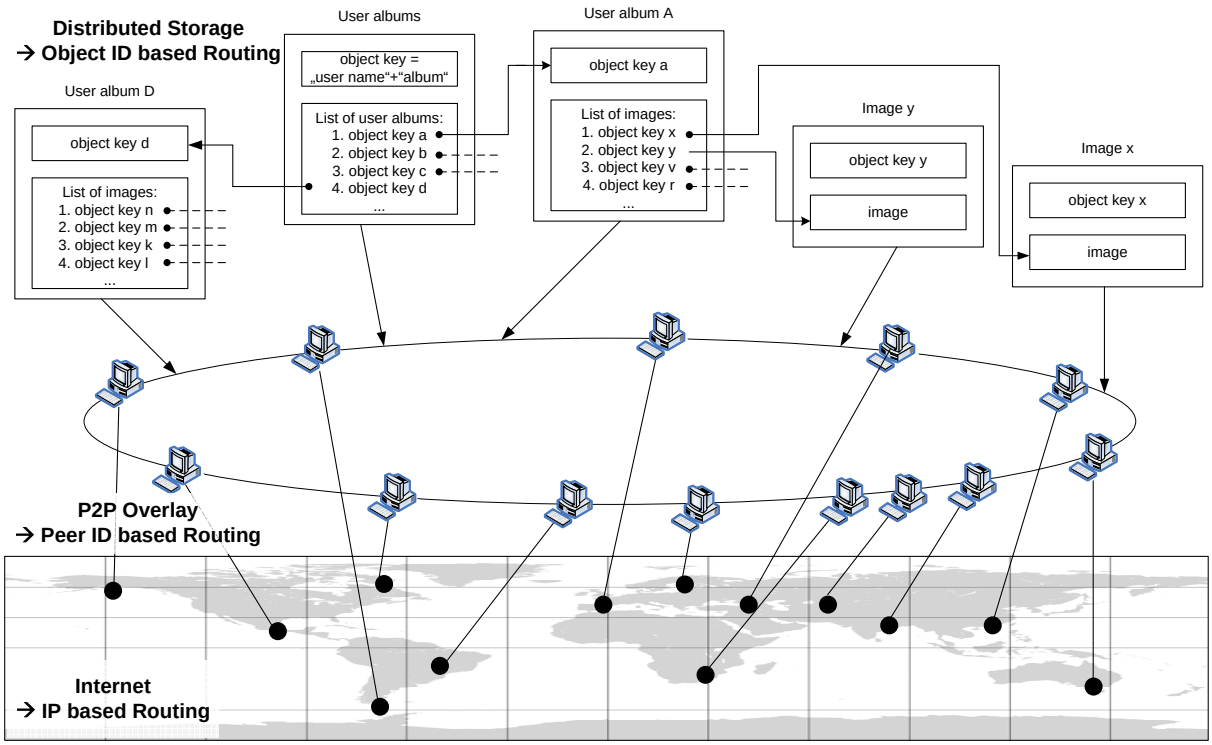


Figure 97: Globally Decentralized Storage of Social Information in Distributed Linked List

to another strategy, where the events from lower layers are forwarded to upper layers. Upper layers can determine the frequency of their queries for related events (e.g. arrived data objects or messages). Some plugins have a higher querying frequency than the others, depending on the character of services the plugin provides. The plugins and user interface components are not triggered by events happening in the lower architecture layers, but by the actions generated by the application user. Lower layers do not have to be aware about the components in the upper layers. If a new component is added, it has no affect on the lower layers.

The second strategy assumes that the events are propagated up to the plugin or user interface layer without delay. This approach avoids that the Information Cache is frequently queried. However, the responsibilities and the development effort for single plugins increase considerably because the plugins have to process the data on their own. The disadvantages of this approach are that the lower architecture layers have to be aware of all components in the upper architecture layer which can be receivers of the data objects and messages provided by the overlay network layer. This makes the coupling of the architecture components less loosely than in the first approach. Plugins and user interface components have to react to all events propagated from the lower layers, even if these events not relevant any more. The plugin has to care about administration of the data objects. If multiple plugins request the same data object from the network, then this data object is loaded twice and two different instances of the same object are held in the memory. The user interface has to react to irrelevant events which is expensive in terms of processing effort. The user interface has to determine whether the arrived object is the one which was requested by the user or it is not relevant. A complicated internal administration of data is required.

With regard to the current paradigm in traditional software architectures, we stick to the first approach with an intermediary layer to store events and provide an interface for all plugins to access these events, namely the Information Cache. Queries from plugins are always answered directly and the distributed storage acts virtually as a database.

To make the advantages of the used event propagation approach more clear, we consider the following example: assume a user requests a data object which consists of multiple parts. Every part of the object is stored separately and there exists a main object which holds all storage keys of these parts. In an overlay network like FreePastry and storage utility like PAST this would mean that all these parts are spread

among a set of network nodes which are currently in the network. If such a data object is requested, the responsible plugin requests all of these parts by initiating the loading of each single part. A subset of these parts may arrive very soon and the rest of them may have a considerable delay. The arrived parts are processed and the user gets the currently available data parts. Not waiting until all parts of the data object arrive the user requests another similar structured data object. The plugin requests all parts which build the latest requested data object. The loading of these parts begins. In the meantime the rest of the previously requested object parts arrive. If all data objects are forwarded directly to the plugin entity, the plugin has to process all arriving data object parts and determine whether they belong to the actually requested data object or not. This would result in redundant plugin activities which would waste processing and memory resources.

Using the event propagation strategy, in which all events are propagated only up to the Information Cache layer, the plugin picks just the relevant data objects from the cache and composes the requested data object, which can be then retrieved by components from the user interface layer or another plugin. The data object parts which were delayed and arrive later on will be stored in the cache. The delayed data parts would not be pushed to the upper layers. In fact, the upper layers (e.g. plugins) would not get notice of arriving of data which is not needed any more. These delayed data objects are then held in the Information Cache and can be accessed by the plugins on demand.

7.3.3 *Data Access Control and Secure Communication in LifeSocial.KOM*

Until now, we presented a p2p platform for online social networks. Here, we discuss the security framework of it, addressing the main concerns and issues related to security in this application field.

REQUIREMENTS FOR A SECURITY FRAMEWORK IN LIFESOCIAL.KOM

First, we describe the requirements and functional goals of a security framework and discuss subsequently our solution.

Registration and Login

A registration phase is needed to grant new users access to the network and to create credentials for the user for later authentication. Users should be able to log on at every peer, i.e. device, in the network, thus login credentials should be purely based on the knowledge of the user. After the registration, the user should be equipped with a valid and unique userID and authentication information. The authentication information, being a pseudonym for the real user, should be stored confidentially, available and with integrity.

The login functionality enables the (pre-registered) user to announce his status in the network. During the login process the user authenticates himself against the authentication information from the registration phase. As a result, the joining of the node is announced in the network, and the node / user can further on be contacted by other nodes.

Access Control

We distinguish between user- and group-based access control, in both cases the security goals are similar. For all documents stored in the network the author should be able to mark *privileged* users, which are authorized to read these documents. Access to selected information of user-specific information (e.g. profile details) or whole documents (e.g. photos) should be controllable. To manage groups with thousands of users, a group-based access control is needed. We call all storable data *SharedItems*.Access rights are dynamic and must be changeable at any time by the author of the document if he or she decides to do so. Access control aims to ensure the integrity, confidentiality and availability of SharedItems inside the community. SharedItems of users or groups must be available with expected service up time (e.g. 99.9 %), thus the security solution must be compatible to common replication mechanisms and caching mechanisms. There must be no restrictions on the peers that store the data.

Secure Communication

During a live chat, all messages are directly sent to the users they are addressed to. For this communication, the sender and receiver must be authenticated, the communication itself must provide

confidentiality and integrity. This wide set of requirements is challenging to solve in p2p systems, due to the peers' unreliability and autonomy.

A SECURITY FRAMEWORK FOR P2P-BASED PLATFORMS FOR ONLINE SOCIAL NETWORKS

In this section, we describe the design of our security framework for LifeSocial.KOM. To summarize the idea, each user creates with his username and passphrase an asymmetric key pair. The public key is used as nodeID and userID in the network. Any communication is encrypted with the public key of the receiver, thus secure and authenticated communication can be provided once the nodeID of the receiver is known. For data storage and access control, we use a hybrid approach. All sensitive data is encrypted with a unique symmetric key, this symmetric key is encrypted with the public keys of the privileged users. The encrypted and signed data and the encrypted keys are stored as a package (CryptedItem) in the p2p network. Any node may retrieve and replicate this data, but only privileged users can decrypt it.

Registration and Login

In the registration process credentials for new users are created in a fully decentralized way. First, the user picks a (unique) user name and passphrase, which is used to generate an asymmetric key pair $\mathtt{Priv}_A$ and $\mathtt{Pub}_A$. The numeric representation of the public key $\mathtt{Pub}_A$ is used as nodeID and userID, identifying both the node and the user.

To join the network, a request containing the node's information (i.e. nodehandle and $\mathtt{Pub}_A$) is send to a bootstrap node. A bootstrap node can be any formerly known node. It looks up the generated nodeID inside the network to prevent any nodeID collisions. If the object exists, the user is already registered, thus next registration steps are skipped. If the object does not exist, the new user creates a minimal public profile, signs it and stores it in the p2p network. Through the signature, the profile is integer. The user is now equipped with a valid userID that will be the basis for later authentication and encryption processes inside the community since the userID is also his public key. Documents or data signed with the user's private key, $\mathtt{Priv}_A$, can now be validated.

For the Login process, user A recreates his key pair by entering his user name and his passphrase within the application. His userID is then derived from the just generated public key. The application sends a login request with the user's userID respectively public key to an available bootstrap node. The bootstrap node answers with information about further nodes to contact. This answer is encrypted, using the public key of the joining peer. The information is crucial to join, the joining peer must decrypt the data with $\mathtt{Priv}_A$, thus authenticate itself by proving the possession of $\mathtt{Priv}_A$.

The presence of a user is depicted by a LoginItem that is stored in the network. This signed object contains the user's nodeID and his IP address. Every time a user logs in, he updates his IP address in the object. The signed LoginItem can be retrieved and verified by any other user. The nodeID/userID is further used to encrypt communication to this node (as it is an asymmetric cryptographic public key). Only the receiving node can decrypt messages that are encrypted in such a way. The concept of using the userID as a public key allows to established a simple PKI without any servers or certificate authorities.

Access Control

A user can read a SharedItem, create a new SharedItem or alter an existing one. In each case he must prove his access rights to do so. We decided to use an Access Control Lists (ACL) based approach instead of Capability Lists, as ACLs can be sticked to data objects and allow an object-specific fine grained control and replication strategy. Each SharedItem that needs access control is encrypted with an object-specific symmetric key. To the SharedItem a data structure (*key list*) is added which holds copies of the encryption key of the SharedItem, wrapped (encrypted) with the public keys of the users who are allowed to access the item. The SharedItem, in addition, with the key list is signed by the author and named CryptedItem. CryptedItems contain all information to enforce access control, they can be replicated and cached. An overview on the SharedItem and CryptedItem is given in Figure 98. Next, we describe the access patterns.

To store a new SharedItem, it is created with a timestamp and signed by the user for later verification of the author. Next, the user defines which other users should be allowed to read this item. If the user

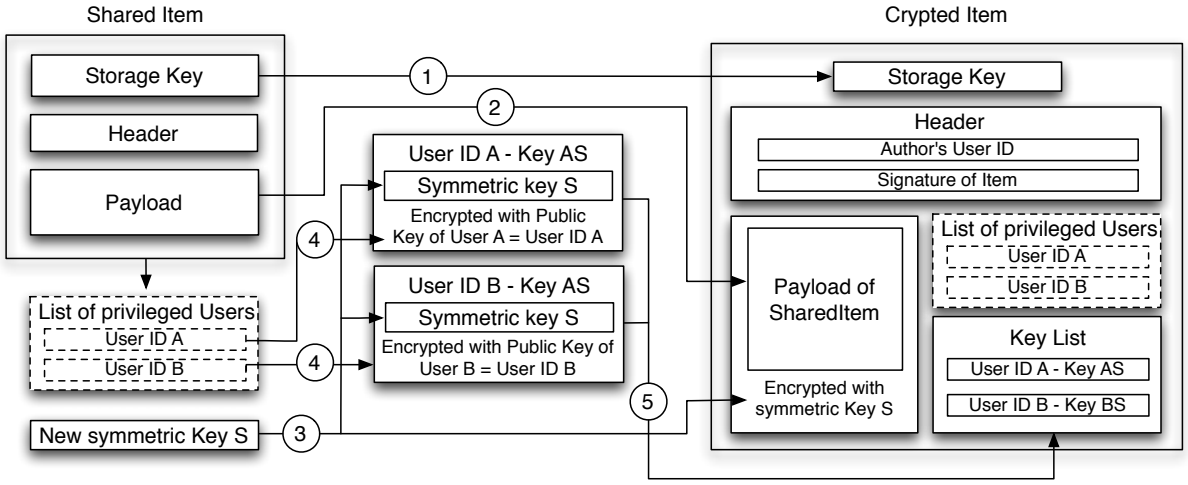


Figure 98: Data-centric Security for P2P-based distributed Data Structures

decides that only a set of privileged users should be allowed to read the item, he creates a symmetric key and encrypts the SharedItem, including its signature, with this key. See therefore Steps 2 and 3 in Figure 98. This symmetric key is then encrypted with the public keys of the privileged users, which leads to n encrypted copies of the symmetric key for n privileged users (Step 4). The encrypted copies of the key are then attached to the SharedItem (Step 5), which is then signed and finally stored as a CryptedItem in the network. Each SharedItem (and thus also CryptedItem) has an objectID, which indicates where in the DHT the object will be stored (Step 1). Using this ID, the object can be retrieved.

To alter an already existing object, it has to be retrieved, modified and stored again. An objectID is created as a hash of the userID and some unchanging properties depending on the type of the SharedItem (e.g. hash(username + albumname)). We refer for more details to Section 7.3.2. As the objectID contains the username as well, any node can check whether a CryptedItem is valid or not using the objectID and the signature of the user related to the objectID. For changing the privileged users of a SharedItem, only the attached keys have to be altered.

Any node can retrieve a CryptedItem from the p2p network. CryptedItems can be replicated and cached using any mechanism. However, only nodes listed in the key list can decrypt the SharedItem. If the retrieving node's ID is in the key list, the symmetric key is decrypted using the private key of the privileged user and the item is decrypted using the symmetric key.

Inside a group, access to documents can be granted for all group members. This allows to use just one symmetric key for all accessible data inside a group. This symmetric key is created by the group founder at the time he establishes the group. At first, the founder creates a key list for his group where he stores the symmetric key encrypted with the public key of the group members. For each new member that joins the group, the administrator just adds a copy of the symmetric key, encrypted with the pursuant public key. This list is stored in the network, signed with the administrators private key to inhibit unauthorized write access. The objectID of this list is a hash of the administrators public key and the name of the group. A user can now store new SharedItems just as described above with the only difference that if he wants to make the item only accessible to group members, he encrypts it with the symmetric key of the group. For read access, a user accesses the key list of the group instead of the key list of a particular item. The protocol for read access is aside from that the same as described above.

Live Chat and Messaging

The live messaging functionality benefits from the design of making the public key of a user also his userID inside the network. User A wants to establish a secure connection to another user B for the purpose of a direct plugin to plugin communication (e.g. a live chat session). We use a hybrid approach for secure communication. User A creates a symmetric session key to encrypt his chat message to user B. User A sends the encrypted message and the symmetric key to user B encrypted with the public key of user B and signed with his own private key. With the signature, both the integrity of the message can

be checked and the sender verified. User B now verifies that the message is really from the sender with the given userID (of User A) by verifying the signature of the message with the public key of User A. If user B wishes to answer, he creates a secret key for the communication himself and encrypts it with the public key of user A. Both users have a secret key for secure end-to-end communication now. Each message sent between the users is signed and encrypted by the sender and verified and decrypted by the receiver.

TESTBED EVALUATION

We implemented the security framework for the p2p platform for social networks, LifeSocial.KOM, which we presented in Chapter 7.3.1. The prototype [Gra] implements the described solution. We used as p2p overlay FreePastry [Ric] and as mechanism for asymmetric cryptographic keys we use RSA [RSA78] with a key length of 1024 bits. To comprise the modulus and exponent in RSA, we enlarged the ID space of FreePastry to 1088 bit identifiers. For the symmetric keys, we use AES [DR02] with 128 bits. A signature is 128 bits in size as well. The described key sizes represent a configuration that provides a desired security level for reasonable costs. All values are averaged over 100 runs on an Intel Core 2 Quad machine with 2.4 GHz and 3 GB of main memory.

The goal of the evaluation is to identify the increase in the message and object sizes related to the security solution. In addition, we analyze the induced time delay in the secure communication due to the security mechanism as well the induced time delay for the encryption and decryption of the CryptedItems.

Data Overhead

Table 23 shows the data overhead on basic messages. We started with an empty message, containing no text but only the header, storage key, receiver ID and an empty payload. Then, we increased the message size by adding larger message text. We observe a nearly constant absolute overhead, smaller than 2 KB per message coming from the duplication of the receiver information and the storage key and from the size of the empty CryptedMessage. Encrypting a basic message and turning it into a byte array does not increase its size perceptibly. The overhead of 2 KB will not affect the traffic speed or the storage space noticeably. Our approach for secure communication is therefore an acceptable solution regarding the data overhead.

Table 24 depicts the data overhead for access control on SharedItems. The size of an item does not affect the data overhead as the signature length and key list size is independent of the object size, therefore we varied the number of privileged users as parameter. The overhead grows with the number of privileged users as for each privileged user, a copy of the secret key is added to the CryptedItem alongside the users' userID. Each additional privileged user causes a data overhead of about 413 bytes. Still, the relative overhead is acceptable even for 200 privileged users. The SharedItem we used, is a PhotoItem which has a standard size of 346 KB. However, any other item of arbitrary size would result in the same absolute overhead, the costs scale with the number of privileged users.

The overhead we must deal with in this case is larger than the message overhead if we have more than one privileged user. However, 200 privileged users for a single object is a turning point of whether individual user-based access control should be replaced by group-based access control. To keep the scenario of a social network in mind, in cases with 200 or more friends, it is recommendable to introduce group-based access. The management of group keys is similar to the management of individual user keys in the CryptedItem, same costs apply.

Time Overhead

We present the encryption and decryption times as an important metric for the costs of a practical security framework in Table 23. The encryption and decryption time for secure communication is around 12 ms and almost independent of the message size. Most of the time is needed for administrative processes like obtaining the encryption keys and building the CryptedMessage.

For the evaluation of the SharedItem regarding the induced delay, we observe that the encryption time rises linear with the number of privileged users. That is because the wrapping of the secret key with the public key of each privileged user takes about 0.36 ms time. Not surprisingly the decryption

Message Size (bytes)	**Encrypted Message Size** (bytes)	**Overhead absolute** (bytes)	**Overhead relative** (%)	**Encryption/Decryption Time** (ms)
895	2794	1899	212,18	10 / 9
995	2906	1911	192,06	10 / 8
1395	3306	1911	136,99	11 / 9
1895	3802	1907	100,63	12 / 9
2895	4794	1899	65,60	14 / 10
3895	5802	1907	48,69	13 / 9
5895	7802	1907	32,35	12 / 8
10895	12794	1899	17,43	11 / 9

Table 23: Message Encryption Data and Time Overhead

Privileged Users	**Item Size** (bytes)	**Encrypted Item Size** (bytes)	**Overhead abs.** (bytes)	**Overhead rel.** (%)	**Encryption/Decryption** (ms)	**Key Wrapping Time** (ms)
1	346697	348159	1462	0,42	15 / 20	1
10	346715	351892	5177	1,49	25 / 21	4
50	346819	368524	21705	6,26	34 / 20	19
100	346969	389318	42349	12,21	54 / 19	37
200	347269	430922	83653	24,09	89 / 20	73

Table 24: SharedItem Encryption Data Overhead

time is constant, as only one key has to be unwrapped in order to decrypt the item with the resulting symmetric key.

Data encryption is distinctly slower than message encryption when we must deal with many privileged users. Still, 89 milliseconds seem applicable for the encryption of items for 200 privileged users. Please note that all used public keys were present in a *buddy keys* list, they were not needed to be retrieved from the network. That applies for the message encryption as well as for the item encryption. However, this is a reasonable step, as user knowing the privileged user(ID) also know the corresponding public key.

Conclusions

P2P-based platforms, like LifeSocial.KOM, face several challenges, among the security requirements which we addressed in section. Our security framework for LifeSocial.KOM includes the support of user registration and a login process which allows further authentication of the users. Any user and application communication is confidential, integer and authenticated. We also presented an access control solution both for user-based access control and group-based access control. The security framework solves the security issues appearing in social networks. We implemented the security framework in our p2p-based platform for social networks, LifeSocial.KOM, demonstrated its applicability and evaluated both its performance and costs. Evaluation shows that all security requirements were solved and the overhead in terms of space and time are low and reasonable in a p2p-based scenario.

7.3.4 *Implementing Online Social Networks in Several Plugins*

Before introducing the implemented plugins, we describe the assumption and properties of the plugin-based design of the application of an online social network.

The p2p platform is assumed to run on every participating peer. Every user picks an unique user name while registering. This user name is persistent and associated with a certain user account. A peer instance as well as a peer ID is generated and assigned every time the user logs in. For security reasons, the peer ID and the user identifier are identical and derived from the user name and corresponding password. This is done to create a trust anchor for authentication and security mechanisms. More

information on the security is described in Section 7.3.3. Every peer creates a data object containing the IP address of the peer (Nodehandle) and stores it in the p2p network, so that other users willing to contact the corresponding user can identify the contact address. Thus, it is only possible for an user to be logged into the p2p network at one single computer at the same time.

The application of online social networks is provided through the combination of various functionality which are implemented in individual plugins. Every plugin included into the system has an unique plugin identifier, which is represented by a sequence of characters, such as "friendsplugin", "chatplugin". All plugins are registered in the local p2p platform and can be accessed using their plugin identifier over OSGi bindings. Using OSGi, plugin dependencies can be specified and individual plugins can be loaded and updated during the runtime of an application. A set of plugins provide the core functionality of the application and must be included into the p2p platform. Plugins which do not belong to the core functionality can be included into the system dynamically while starting and initializing the system if desired.

Two different LifeSocial.KOM clients running on different network nodes can communicate over messages and/or over the shared network memory. All messages are plugin-specific and contain plugin-specific payload. Knowing the type of a plugin message, other plugins can generate and send messages to this plugin. Undelivered plugin messages are stored by the Message Dispatcher and can be retrieved by the recipient at an appropriate point of time. Distributed linked lists are used as a basis data structure for storing data objects.

Knowing the main p2p functional components, which provide secure and reliable storage and communication, we now introduce the main plugins implementing the desired functionality of online social networks. We depict in Figure 99 the functional elements, we designed and implemented. The plugins (or apps) are located on top of the general p2p platform, depicting the general usability of the platform.

We implemented following plugins:

- Login: Registration / login based on cryptographic keys.
- Profile: Presenting a description and image of the user
- Friends: A list, linking the profiles of the user's friends
- Messaging: Email-like inbox, outbox and message composer
- Photo: A list of photo albums, linking to user photos
- Groups: A list of users, joined in common interest groups
- Tweets: List of status updates of an user and its followers
- (Group) Chat: Direct user to user text messaging
- File transfer: Sending files from user to user
- Games for two with spectators: Tic Tac Toe
- Whiteboard: Collaborative graphical editing on a shared canvas
- Schedule and Calendar: Collaborative time schedule
- Multicast: Creation of and publishing to multicast groups

With the plugin-based architecture for p2p applications, LifeSocial.KOM is extendible in terms of functionality. In the following, we describe selected plugins that show the potential of the underlying p2p platform as well as the communication concepts offered to plugins.

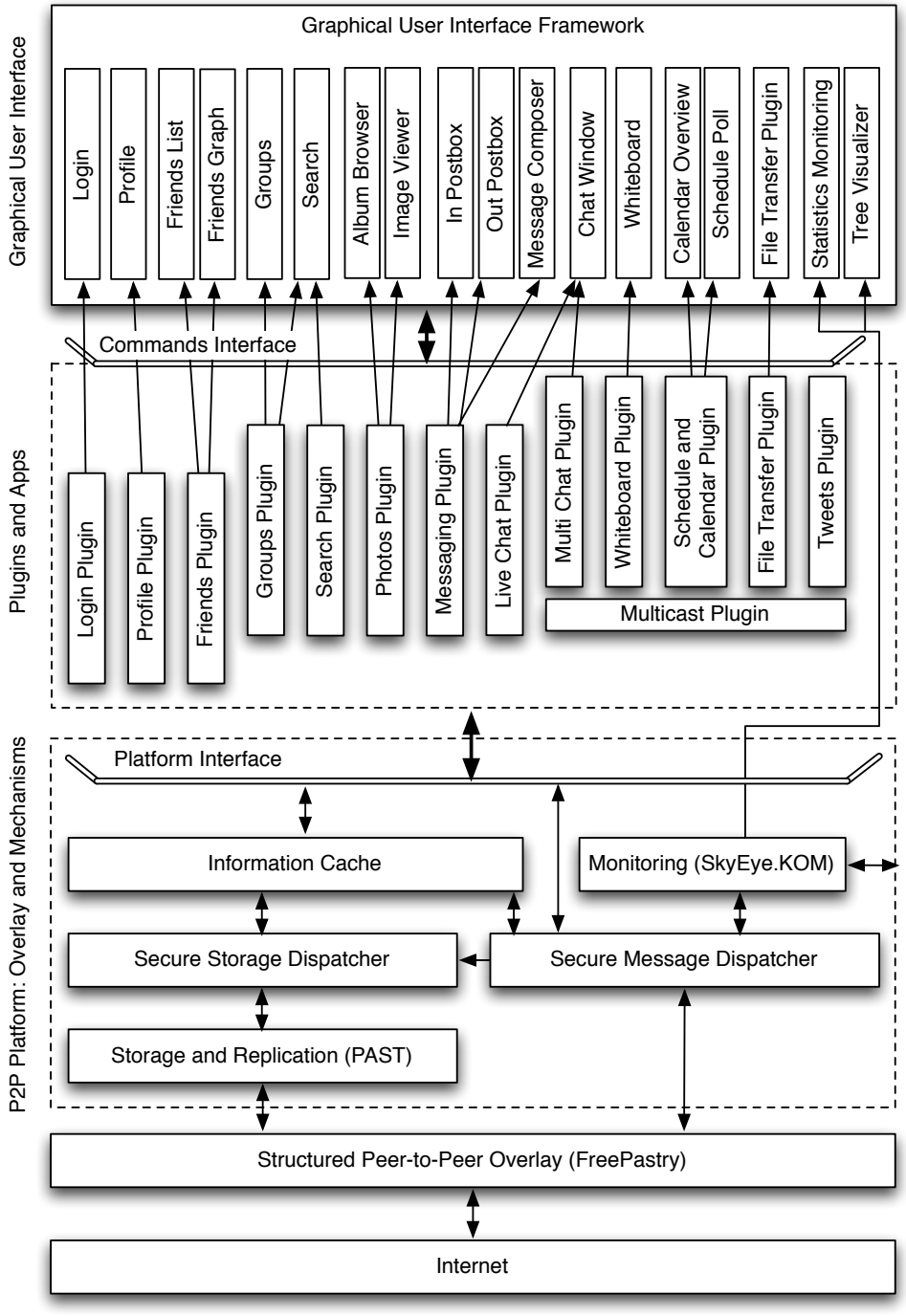


Figure 99: Plugin-based Architecture of the Online Social Network Application

LOGIN PLUGIN

The Login plugin belongs to the core part of LifeSocial.KOM and is responsible for creation and administration of user accounts. It offers the functionality to create a new user account, to log in and to log out. This is the plugin which is initialized first when the system is started. Other plugins are initialized after the user logs into the system by providing his/her user name and the corresponding password. Optional plugins are bound into the p2p platform using OSGi. After the initializing, the functionality of all plugins is available for the user.

The Login plugin uses data objects dedicated for the Login plugin termed *LoginItem* extending the general SharedItem. During the registration of a new user, the user has to pick an username and password resulting in an unique cryptographic key pair. The Public Key is used as user identifier and defines also the peer ID of the user's peer. Each LoginItem stored in the network holds the entire account information for one user account.

The payload of LoginItem contains the information on the user name, his email address, the peer ID, a Nodehandle object as well as a timestamped signature of the user. The Nodehandle object holds the ID and the current IP-address of the peer or is empty if the user is offline. It can be used for sending messages directly from peer to peer. An alternative way to route messages from Pastry node to Pastry node is to use the peer ID. Before an user account is created, the Login plugin checks whether the provided user name and email address are already used by other users. If it is not the case, then the LoginItem is stored and the user becomes automatically logged into the system.

The LoginItem can be (stored and) retrieved using an appropriate storage key which is built using the user name, the plugin identifier and a data type specific descriptor. The storage key for plugin-specific data is typically built as

$$\textit{storage key} = \textit{hash(user name} \circ \textit{plugin identifier} \circ \textit{String token)},$$

where the "user name" denotes the String representation of the user name, "plugin identifier" denotes the String representation of the unique plugin identifier, the "String token" stands for a Login plugin-specific String token, which is used to avoid key collisions and $\circ$ stand for concatenation operations.

An example will demonstrate how the storage key for a LoginItem data object is built. Assume that the user name is "Filiz", the identifier of the Login plugin is equal to "LoginPluginIdentifier" and the specific token is equal to "UserAccountObject".

The resulting storage key would be the following :

$$\textit{hash(Filiz}\circ\textit{LoginPluginIdentifier}\circ\textit{UserAccountObject)}$$

The uniqueness of the user name makes sure that the storage key is also unique within the system.

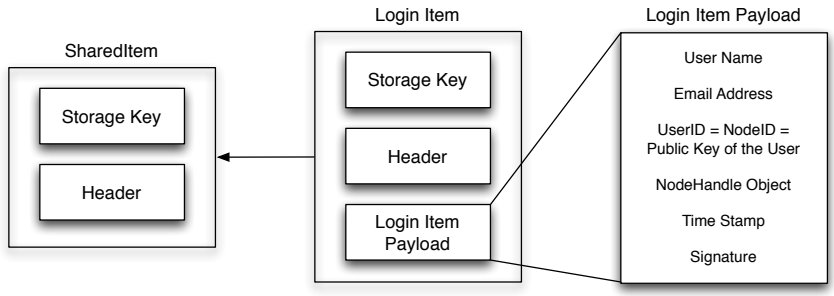
The LoginItem has an important role in the system. It holds the information required for communication between users. Given an user identifier, the storage key of the LoginItem associated with the user account can be derived. Using this storage key, the Nodehandle of the recipient can be retrieved and messages can be routed to this user.

The LoginItem object is expected to stay permanently available in the shared storage. A loss of such data object would cause the loss of all account information associated with the user name held in the LoginItem. Each time an user logs into the system, the associated LoginItem is modified by updating the Nodehandle which holds the current address information about the Pastry peer. Additionally, the signature is updated. The LoginItem is used by the Message Dispatcher to send user-to-user messages, as it contains a valid contact address to the user. Figure 100a shows the structure of a LoginItem object. The structure is similar to the structure of the data object which other plugins operate on. All of them extend the SharedItem class to make the objects storable.

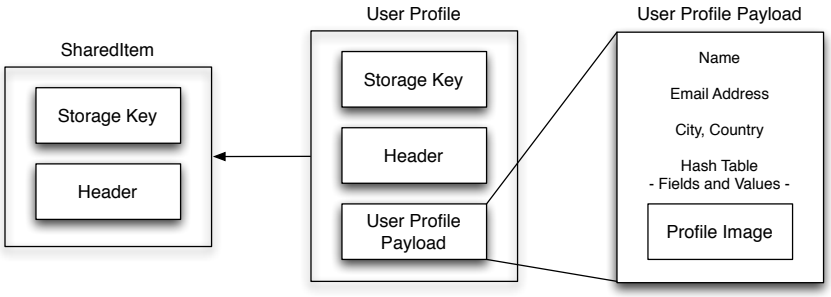
PROFILE PLUGIN

The Profile plugin is responsible for creation and administration of an user's profile data. It is also responsible for finding and retrieving the profile data of other users. The creation of a basic account is immediately initiated by the Login plugin after the registration of the new user account. This fact guarantees that every registered user owns a profile with some basic personal information. Additionally, it shows that plugins may invoke exposed methods of each other. The Profile plugin offers the functionality to create user profiles, retrieve a profile of an user given an user identifier, edit profile fields and store the own profile.

The Profile plugin uses dedicated data objects termed UserProfile. Figure 100b demonstrates the simplified structure of an UserProfile object. Like all data objects the plugins work with, UserProfile extends the concept of SharedItem and results in a storable object. The payload of UserProfile contains a profile image and profile field names with values.



(a) LoginItem Structure



(b) UserProfile Structure

Figure 100: Data Structures of selected Plugin Objects

Profile fields are names of the information the field is associated with. For example, the field name can be "Gender" and the possible values can be "Male" or "Female". Another field name is "Country" and the value associated with this field is "Germany". The number of profile fields can be extended by adding new field names and values.

Directly after the Profile plugin is initialized, the profile of the currently logged in user is retrieved from the shared network memory and made locally available in the Information Cache. Since the owner of a profile is the only one who can edit the profile contents, the version of the profile which is held locally by the plugin remains valid. If the user profile is changed, the profile is stored in the network and updated locally to have an actual version of the data.

A profile of an user can be loaded by the Profile plugin and made available for the user interface. The user interface may instruct the Profile plugin to load a profile for a certain user given his/her user name. The Profile plugin checks whether the needed data object is available in the Information Cache and if not the Information Cache initiates the lookup of the data object from the p2p network using the Storage Dispatcher. During the loading process, the Profile plugin responds the current status of loading and the data loaded so far available for the user interface. The user interface can check whether the requested profile is loaded and get the data from the plugin.

FRIENDS PLUGIN

Every user owns a list of friends which may be empty for unpopular users. This list contains user names of the users which are registered as friends of this certain user in scope of the system. The Friends plugin is responsible for administration of the user's friends list.

When the plugin is initialized, the data object containing the list of friends of the currently logged in user is loaded. This list contains user identifiers, user names and meta information, and storage keys of corresponding ProfileItems. This list remains available within the plugin until the system terminates. The friends lists for all users are stored in the network in dedicated data objects (FriendList) and are

available for every user. The user can request a list of friends of any other registered user. In this case the Friends plugin requests the corresponding data object from the Information Cache. Typically it is not instantly available, thus the Information Cache requests it from the Storage Dispatcher which fetches it from the DHT. Once the data object arrives, it is passed to the Information Cache and made available to the plugins. The next time (and from then on) the Friends plugin asks for this specific friends list, it is instantly available and returned to the Information Cache.

Besides retrieving friend lists, the plugin also offers the function to request and delete friendships as well as to accept or decline friendship requests. In order to implement these exposed functions, internally the Friends plugin is able to load friends list for a certain user, get his own friends list, send friendship request, check for undelivered requests, retrieve new requests, accept or deny friendship request and dismiss a friend.

As friends lists are stored as FriendLists in the network, adding or removing friends can only be done by modifying these data objects. Corresponding storage keys are for example

$$hash(Marko \circ FriendsPluginIdentifier \circ FriendListObject).$$

Requests for friendships are also stored in the network by adding a new list entry in the data object managing the pending friend requests of an user. The list entry contains the user identifier and user name of the requesting peer/user. The Friends plugin periodically checks for new entries, retrieves them and either adds the new user to the FriendList data object or not. In the both cases the pending request is deleted. Already established friends can be dismissed by deleting their list entries from the FriendList.

Here, we see that typical actions in distributed linked lists are the adding or deleting of individual list entries. This function is provided by the Storage Dispatcher, as it allows to modify data objects remotely. Friends lists allow users to navigate through the social network built within the online community. Using the user identifiers, which are contained in the friends lists, different actions can be performed such as using the functions of mandatory or other well known plugins.

LIVE CHAT PLUGIN

In order to demonstrate the benefits of the Message Dispatcher, we present a plugin that states real-time requirements for user-to-user communication. According to [Gö5], Real-time communication is essential for upcoming multimedia applications, such as haptics [ESCK08, IES08]. The main purpose of the Live Chat plugin is to provide an instant messaging service for the users. The messages, exchanged between users, are text messages. However, the concept is not limited to text messages, multimedia streaming, in addition VoIP, can be supported as well. The text messages are delivered directly to the recipient using the Message Dispatcher and addressing a specific user identifier (e.g. user ID of Sandra) and plugin identifier (LiveChatPlugin). Messages arriving at the Message Dispatcher of the user Sandra are directly forwarded to the Live Chat plugin.

We extended this functionality with a support for a chat history. A set of messages exchanged with a certain user is stored in a data object dedicated to the Live Chat plugin in the p2p network. The number of messages included in such a set is restricted. The most recent messages build this set. Each time the Live Chat plugin is initialized, the appropriate data object for the chat history is loaded from the network and made available within the plugin. The Live Chat plugin provides besides the functionality to send messages to a given user, also the function of check if missed chat messages are available, retrieve missed chat messages and get a chat history related to a communication partner. Several ideas regarding this plugin, were discussed and integrated additionally in the p2p platform [MKBS08] as a tool termed *AskMe* in [MKS08]. The authors, P. Mukherjee and A. Kovacevic, of these papers were also closely involved in the creation of LifeSocial.KOM and its security framwork, which is presented in the Section 7.3.3.

PHOTOS PLUGIN

In order to show the potential of distributed linked lists, we present a plugin which deploys a complex distributed data structure. The Photo plugin is responsible for the administration of albums and photos which users post into the system and make available to other users. Every user owns an arbitrary amount of albums (zero or more). Each albums can contain zero or more photos. Every photo is assigned to only one album, but the same photo instance could also be referenced in several albums. The Photo

plugin allows to store own photos and to load and view photos of other users. The corresponding data object is depicted in Figure 94 and organized as follows:

- an 'AlbumKeysHolder' data object holds the storage keys of all albums for a single user
- an album, represented by 'AlbumItem' data object, contains references (storage keys) of photos which belong to this album along with some album information
- a single photo represented by 'PhotoItem' data object holds an image and some information about the image

In order to view photos of an user, an appropriate AlbumKeysHolder object with the storage keys of all albums must be loaded. The storage key is related to the username, the plugin identifier and an object descriptor. For example, it looks like

$$hash(Daniel \circ PhotoPluginIdentifier \circ AlbumKeyHolderObject).$$

The Photo plugin may request the AlbumKeyHolder object of a given other user by building the storage key and asking the Information Cache. When the AlbumKeyHolder arrives, the Photo plugin can extract all storage keys of all user albums from it. Using these storage keys, all albums are loaded from the network. The albums are represented by AlbumItems and contain a list of storage keys of the photos assigned to these albums. The AlbumKeyHolder and AlbumItem object contain only lists of storage keys and hence do not consume much bandwidth while being loaded. Having all AlbumItems of an user, we can see the set of photos which are assigned to any of the albums. The Photo plugin can load a single or all chosen PhotoItem objects from the network, which contain the images. The required photos and albums may be prefetched by the Photo plugin and are stored in the Information Cache for instant access. Here, we also observe the benefit of the Information Cache, as it provides a container for incoming photos. In order to load all albums, represented by AlbumItem objects, the Photo plugin uses a dedicated plugin which is responsible for loading all albums, termed *AlbumLoader*. AlbumLoader gets a collection of storage keys and loads all albums associated with these keys into the Information Cache and thus the Photo plugin. The Photo plugin offers not only the functionality of retrieving albums, but also for creating new photo albums, adding album information and photos, editing album and photo information and removing albums and photos.

Obviously, the Photo plugin uses only the shared network storage and does not use the user-to-user communication. It builds a distributed data structure, which allows for navigating and retrieving complete or partial albums from the network.

7.3.5 *An Extendible User Interface for P2P Applications*

The plugins described in the previous subsection offer dedicated interfaces and commands for each other and provide interfaces for a graphical user interface. In the following, we describe how we create a convenient view on the commands available as well as the interaction with the graphical user interface. The GUI is based on OSGi-technology, like the plugins, and allows to have plugin-based individual GUIs arranged in a flexible GUI framework.

GENERAL VIEW ON THE OFFERED COMMANDS

In order to provide a general view on the offered commands, we implemented an OSGi bundle (*Commands*) that serves as interface between the GUI and the application plugins and make the plugins self contained regarding the commands they provide to the platform. The *Commands* bundle serves as an aggregator of all commands currently available in the platform. When a plugin is activated, it registers its available commands. The Commands bundle then automatically recognizes the new commands and makes them available to other bundles. When a plugin is deactivated, its commands are accordingly unregistered.

In order to not bind a specific implementation of a plugin to a specific command set, we further introduce interfaces for the commands, termed *ICommands*. ICommands define the basic interface for the interaction with the different Command Handlers representing a single command offered by a plugin. The abstract class *AbstractCommands* then implements the mechanism to deal with Handler insertion and

removal. A plugin exposing commands can then extend the abstract *AbstractCommandHandler*, which merely implements mechanisms to obtain the name and help of a command, to add the dependency on the plugin it represents and from there create concrete classes which actually execute the command. The concrete class interacts with the plugin (e.g. the ProfilePluginComponent) to execute the command.

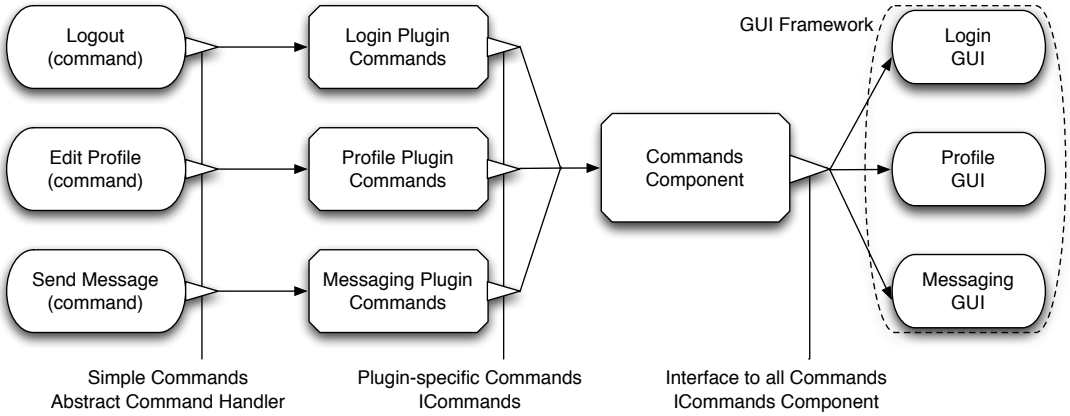


Figure 101: Aggregation of individual Commands in a Commands Component

Figure 101 shows how the GUIs access the commands offered by different plugins. The *CommandsComponent* acts as an interface between the GUIs and the plugins. The GUIs are thus dependent only on the Commands component, which itself takes care of dealing with the registration and deregistration of the different commands. Command names are built by concatenation of the corresponding plugin ID and the command itself, thus being unique in the p2p platform. The return type of commands is a general object, so all LifeSocial.KOM items and message types can be accommodated. This is beneficiary for combining various existing plugins and creating new functionality through new plugins. However, when executing a command it is therefore important to check if the returned object is of the correct type.

EXTENDIBLE GUI FRAMEWORK

The Information Cache, Message Dispatcher and Monitoring components act as interfaces between the p2p platform and the application-specific plugins. The Commands component acts as an interface between the application-specific plugins and a GUI. Through this separation of functionality each part can be monitored, individually extended and improved without interfering with the other two parts.

Every plugin comes with one or several (e.g. for the Friends plugin) graphical user interfaces which are arranged in a GUI framework. The GUI framework allows for managing and placing the individual plugin windows. We implemented a GUI framework based on OSGi as a rich client platform (RCP) using Eclipse [Ecl] as a basis. Eclipse is a popular open-source software development tool and was developed initially by IBM. Its main functionality lies in supporting the software development process with an integrated provision of necessary and convenient tools. These tools, such as a help screen, console or repository browser, are arranged around a central source code editor. The GUI of eclipse integrates for all of these tools individual *views*, which may be switched on and off, rearranged and modified. Eclipse offers various RCP libraries and plugins to extend the GUI of Eclipse or to create GUIs for standalone applications.

We use this convenient approach to create and manage the GUIs of individual plugins and arrange them in a common GUI framework. Every plugin is mapped to an individual GUI which implements the commands offered, for example the request friend list of a specific user, and presents the results of the command invocations, which, for example, depict a friend list. In Figure 99, we depict the dependencies of the GUI framework, the individual plugin-specific GUIs and the plugins themselves.

7.4 EVALUATION AND TESTING

We implemented the p2p platform LifeSocial.KOM with SkyEye.KOM integrated as a Java-based standalone application since 2008. We show the feasibility and practical usability of LifeSocial.KOM in

a testbed evaluation with the proposed p2p platform. The goal of our evaluation is twofold. First, we show that the p2p platform provides the desired functionality to support the rapid development of new p2p applications in form of plugins for LifeSocial.KOM. We implemented a set of plugins providing the application of online social networks, harnessing the resources of the participating peers to create a reliable p2p-based platform for social online networks. The data generated by the users is distributedly stored and replicated among the participating nodes and all load is deployed on the peers.

The second evaluation goal is to show the benefits of an integrated quality monitoring component, namely SkyEye.KOM. Using SkyEye.KOM, we monitored the p2p platform on various layers:

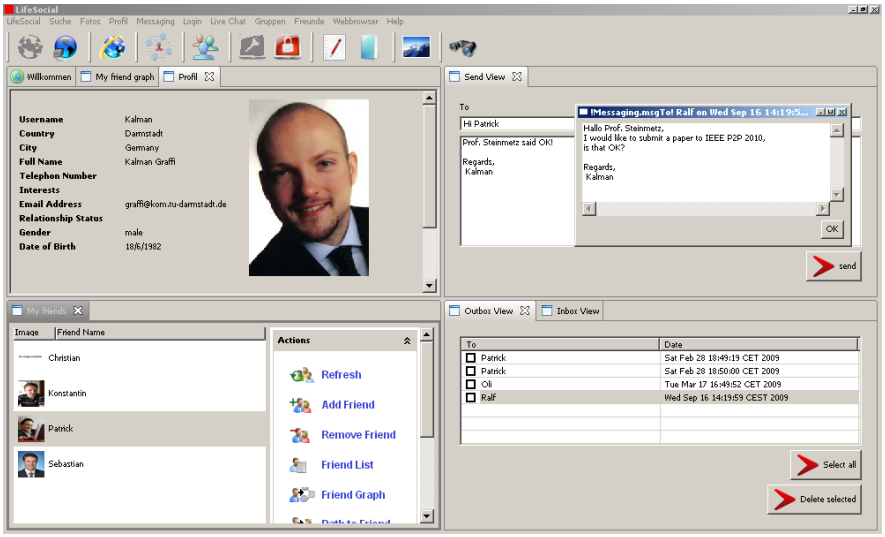
- Device capabilities: we build statistics on the available peer resources, such as the CPU load, bandwidth utilization and available storage space.
- P2P overlay: we measured a large set of statistics related to the p2p overlay, such as the hop count, lookup delay and generated traffic overhead.
- Storage and messaging: the main components between the p2p overlay and p2p application, i.e. the Message Dispatcher, Storage Dispatcher and Information Cache, are monitored and statics related to the overall application are created, e.g. number of stored objects and application-specific traffic overhead and storage space consumption.
- Plugin layer: we further monitor individual plugins and generate statistics related to their individual characteristics.

We extended the list of monitored system statistics (see Tables 2, 3, 8) with a set of new, LifeSocial.KOM-specific statistics that are presented in Table 22. With this broad range of considered statistics in the p2p platform, we demonstrate the applicability of the proposed monitoring mechanisms. SkyEye.KOM generates a global view on any involved layer in the p2p system and enables users to be informed about current statistics and platform providers to see quality trends and general characteristics of the p2p system.

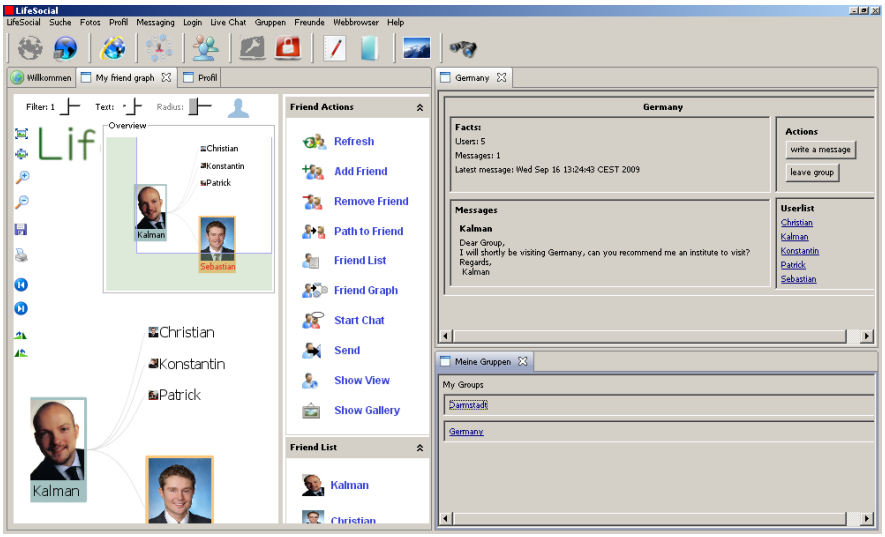
USER AND PROVIDER VIEW ON LIFESOCIAL.KOM

In LifeSocial.KOM, we implemented a wide set of functional plugins. For all of these plugins, we created GUIs and placed them in a GUI framework analogue to the GUI of Eclipse Development Framework. We depict in Figure 102 three screenshots of the actual LifeSocial.KOM application. The screenshots show the variety of the plugins and the option to customize the view positions, sizes and presence in the GUI framework. Figure 102a shows the GUIs for the Profile plugin, the Messaging plugin with an opened message composing window as well as a list-based GUI for the Friends plugin. Figure 102b shows a graph-based GUI for the Friends plugin, emphasizing the extendability of the GUI framework, as well as GUIs group lists and group details. Figure 102c shows the album browsed and the image frame of the Album plugin, which are two separate GUIs for one plugin. The depicted picture was shot by the author at Neko Harbor as sample image. Both the menu entries as well as the depicted icons are dynamically loaded with the corresponding OSGi component, representing the plugins. The GUIs, although made for a specific plugin, are rich in their functionality and may use the interfaces of several plugins. The friends graph GUI, for example, offers also links to establish a direct chat and to show the photo albums of the corresponding friend.

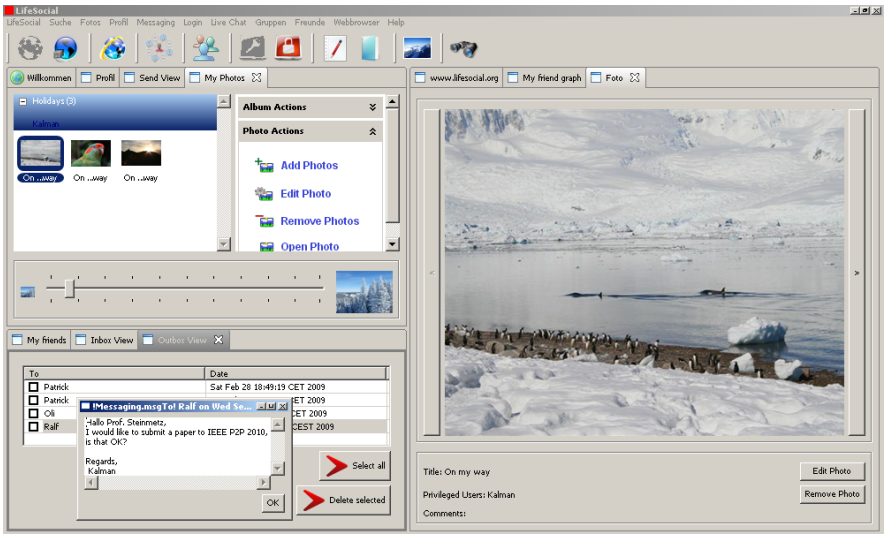
The presented screenshots show the user view. We further implemented GUIs for the monitoring component, SkyEye.KOM. It enables the user and system provider to select among the monitored metrics which to present. It also provides a view on the topology of the SkyEye.KOM tree. In Figure 103a, we show the monitoring GUI on LifeSocial.KOM. In the lower left corner, the metric selector is depicted, which allows to pick among the monitored metrics one to present. Here, the uptime distribution of all peers and the LifeSocial.KOM-specific storage space consumption are depicted in detail with the average value as well as the standard deviation. In Figure 103b, we show the SkyEye.KOM tree visualizer which is based on the peer-specific information that is obtained through monitoring. We used this view to show the tree characteristics in the testbed-based evaluation. In Figure 103c, we show the monitoring tree with 30 nodes maintaining the LifeSocial.KOM platform. More screenshots and videos of LifeSocial.KOM can be found on [Gra].



(a) Profile, Messaging and Friend List GUIs

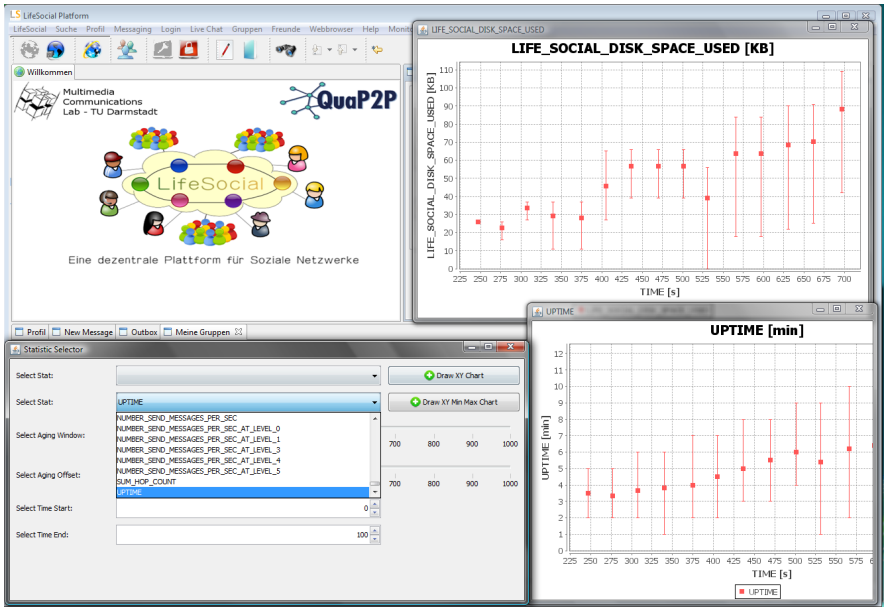


(b) Friend Graph and Groups GUIs

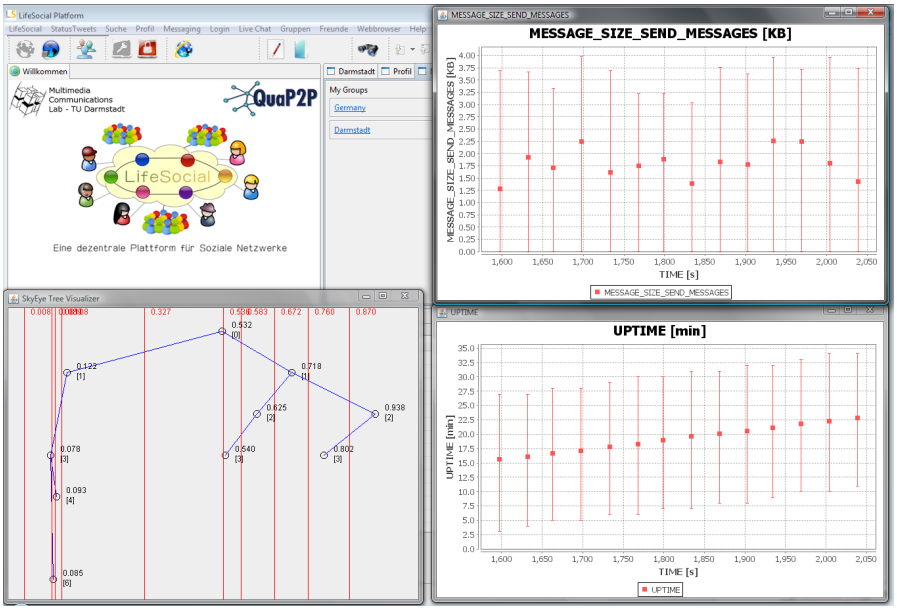


(c) Album Browser, Image Frame and Messaging View

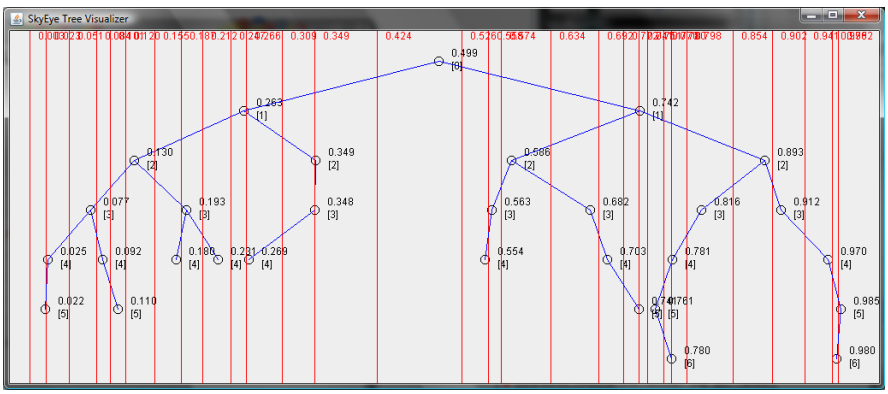
Figure 102: Actual Screenshots of LifeSocial.KOM



(a) Monitoring Metric Selection View



(b) Sample Monitoring Views

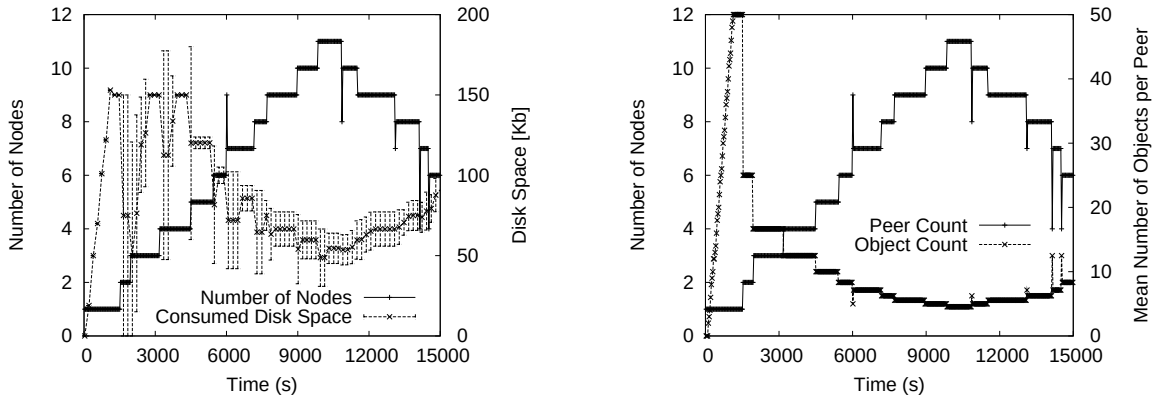


(c) SkyEye.KOM Tree Visualizer

Figure 103: Monitoring View on LifeSocial.KOM

PROOF OF CONCEPT - APPLICABILITY OF SKYEYE.KOM IN LIFESOCIAL.KOM

We demonstrate the applicability of SkyEye.KOM in LifeSocial.KOM and point out the practical use in a general environment. As evaluation methodology, we used a testbed with 11 peers and 30 peers and applied a scenario with joining and leaving peers which perform actions related to online social networks. In both scenarios we used SkyEye.KOM with $\beta = 2$ and $UI = 30s$ both for monitoring the system as well as the peer capacities. The 10 peer setup allows to deeply gain insight on the behavior of data distribution in LifeSocial.KOM. In the second setup with 30 peers, we investigate the impact of churn on traffic overhead and monitoring quality. In the small setup, first, 11 nodes joined one after another through a 15.000 seconds trial. The first node generates a set of profiles and sets up his individual profile and friends list. Once all of them are online, 5 of the nodes leave the network. As main metric, we observe the load on the peers in terms of disk usage of LifeSocial.KOM, the distribution of data objects in the p2p network as well as the number of nodes. These metrics document the distributed data storage of the user-specific information in the p2p network.



(a) Disk Usage

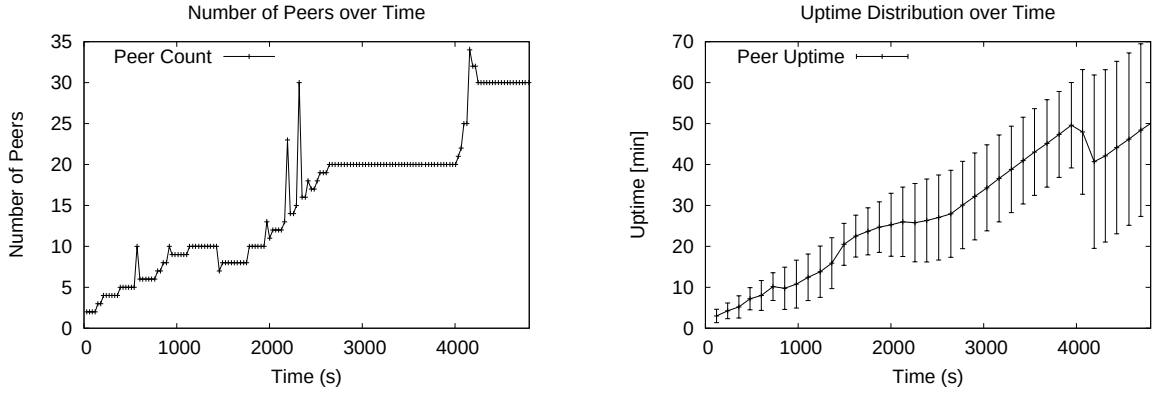
(b) Peer and Object Count over Time

Figure 104: Monitoring Results of LifeSocial.KOM in a Small-Scale Network

Figure 104a shows the mean disk usage in the network in relation to the number of peers in the p2p network over time. The figure shows how the number of peers increases and decreases over time. The mean disk usage per peer denotes that at the beginning the first peer created a set of objects which used 150 KB of disk space. With the second peer coming online the objects are assigned according to their object identifier to the corresponding peer identifiers. The standard deviation in the beginning up to 4 nodes is high, as every document is replicated in order to maintain a high object availability. The average number of objects per peer is depicted in Figure 104b. Every object is replicated 4 times, which is reflected in the mean disk usage of 150 KB even with 4 nodes. With the fifth node joining, no further replication is needed and the mean disk usage drops. With every joining node, the load is shared among the peers, as depicted by the decreased mean disk utilization and the low standard deviation. As the peers start leaving the network, the distributed data objects need to be replicated again and the mean disk usage grows. The evaluation shows that the storage load is totally distributed among the participating nodes while maintaining a high data availability.

In the second testbed setup, we look at 30 peers which join subsequently in blocks of 10 peers. The first 10 peers join in the beginning in 20 minutes, create a profile and stabilize for 5 minutes. From $t = 1500s$ to $t = 2000s$ two of the peers left the network and joined again. Then the next 10 peers joined and the corresponding users created a profile. From $t = 3000s$ to $t = 3500s$ the peers request group and profile lookups. From $t = 4000s$ to $t = 4200s$, 10 more peers join. Having already sketched the proof of concept for the applicability of SkyEye.KOM in LifeSocial.KOM, we focus in this step of the evaluation on giving a general view on the performance and costs of LifeSocial.KOM. In Figure 105, we show an overview on the actions in the testbed evaluation setup of LifeSocial.KOM.

In Figure 105a, we visualize the peer count in the mid scale network. The corresponding peer uptime distribution is depicted in Figure 105b. We depict the tree topology of the SkyEye.KOM tree at $t = 2100s$ in Figure 106a. At time $t = 2194s$ the monitored peer count jumps to 23 which is related to the exchange



(a) Peer Count

(b) Online Time Distribution

Figure 105: Monitoring Peer Count of LifeSocial.KOM in a Mid-Scale Network

of the root. At time t = 2320s the peer count jumps to 30. We present the corresponding tree topology in Figure 106b and Figure 106c. In both cases the root has been exchanged by the reassignment of the corresponding Domain Key (0.5) in the unified ID space. In Figure 106d, we show the tree topology with 30 LifeSocial.KOM nodes.

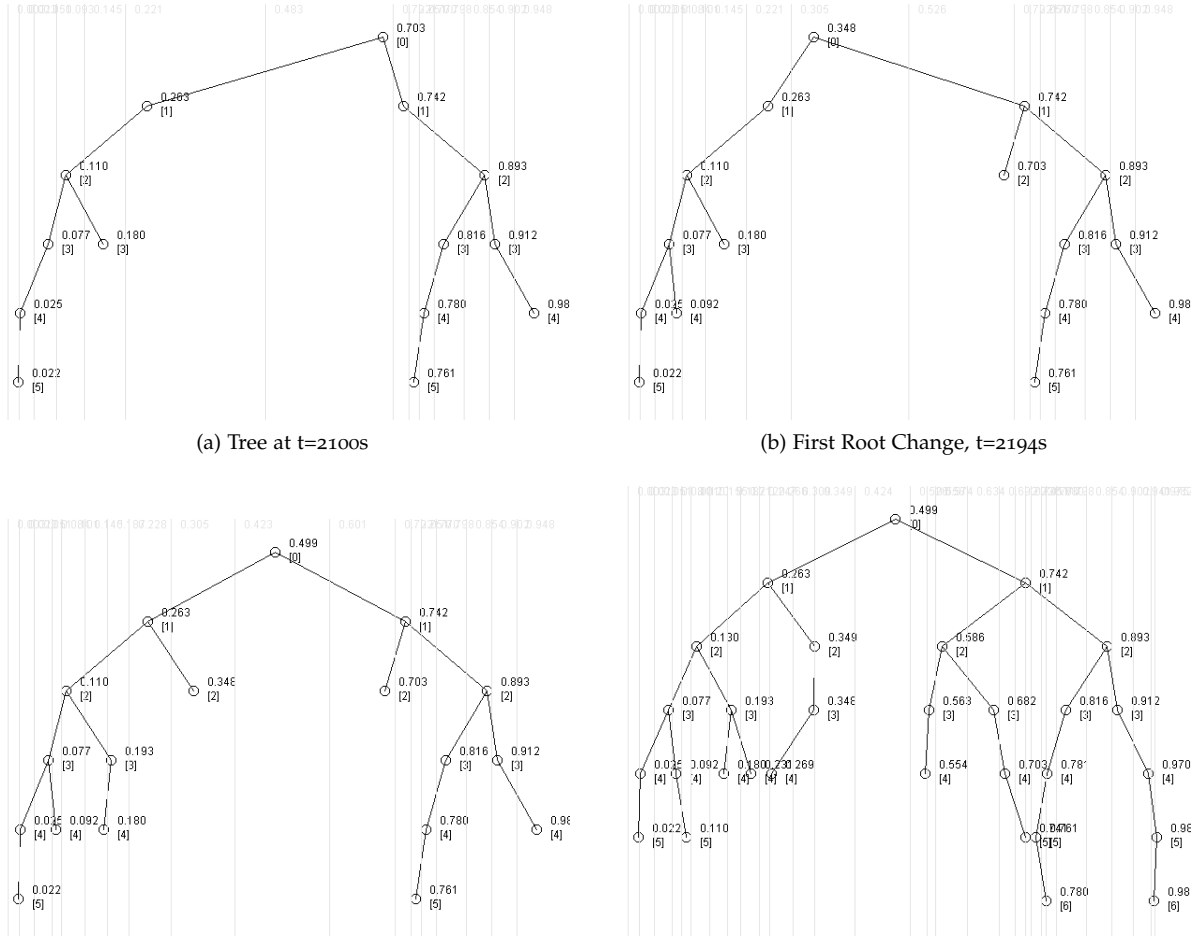
In the following, we take a look on the overhead generated through LifeSocial. An overview on the various overhead types is depicted in Figure 107. The main two metrics are the disk space usage and the bandwidth usage of LifeSocial.KOM, they are depicted in the Figures 107a and 107b. While the disk space usage is around 150KB in the stabilization phase around t = 1200s, it grows in the following and stabilizes in the period from t = 3000s on. Here, we observe the replication efforts that are done to maintain the data availability in LifeSocial.KOM. Through the observation of the disk space usage, an application provider may define suitable initial storage intervals to be announced and confirmed by the user during the installation of LifeSocial.

Traffic, being the scarcest resource is compressed before transmitted. Thus, the bandwidth consumption is relatively small. LifeSocial.KOM needs in this small and mid sized setup only 2.1 to 2.5 KB/s both upload and download bandwidth. With regard to the replication efforts, the bandwidth consumption is expected to grow. As a conclusion, we derive the need for a well engineered structured p2p overlay that is optimized for the needs of LifeSocial.KOM. In particular, FreePastry and PAST fail at supporting the peer heterogeneity in the process of data replication. Peers with larger storage space and greater bandwidth should be used more efficiently and play a dedicated role. In addition, long living peers should also fulfill in the p2p overlay more specific roles that utilize their potential to stay long online.

In Figure 107c, we show the averaged total message overhead in the network, in Figure 107d the corresponding message sizes over the time. One observation is that the peer count has no influence on the message overhead, also the online time of the peers has no influence. All peers send up to 1 message per second and show a larger standard deviation for the received messages than for the sent ones. The total traffic in the network is depicted in Figure 107e and Figure 107f. Here, the effects of compression are visible. This view on the total overhead helps application providers of social networks to evaluate the infrastructure costs to maintain the service. Both the total and the average load help to analyze the stress on the individual peers. Here also the CPU utilization, main memory utilization, used storage space and bandwidth are valuable indicators on the load on the peers. System providers may detect shortcomings in their solution and initiate stabilizing actions in case of obvious problems.

The evaluation shows that a general p2p platform for rapid development of p2p applications is feasible through the composition of well evaluated components like FreePastry, PAST and SkyEye.KOM. With an addition of components like the Storage Dispatcher, Information Cache and Message Dispatcher, we created a platform for plugin-based p2p applications. LifeSocial.KOM provides the functionality of online social networks, in an easily extendible manner.

We show that it is feasible and beneficial to integrate the monitoring component SkyEye.KOM in the p2p platform in order to enable users and platform providers to evaluate the quality of service provided



(a) Tree at t=2100s

(b) First Root Change, t=2194s

(c) Second Root Change, t=2324s

(d) Final Tree Topology, t=4500s

Figure 106: Tree Structure in LifeSocial.KOM in a Mid-Scale Network

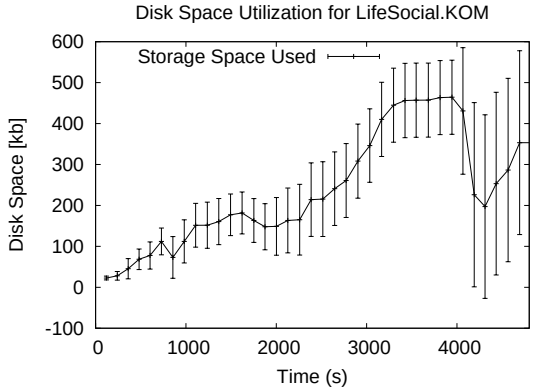
by the p2p system. The observed metric in the evaluation show the flexibility of the approach to monitor metrics on various layers in the p2p system. Thus a quality controlled and easy to extend general p2p platform for p2p applications has been built.

7.5 RELATED WORK

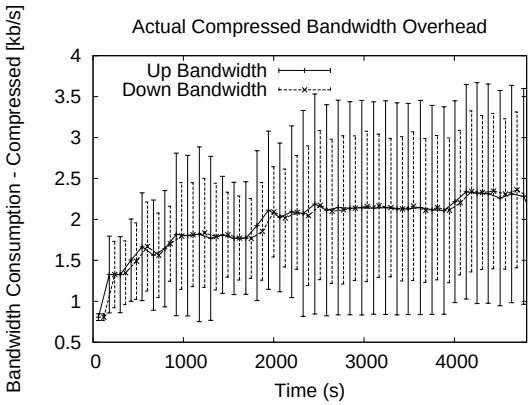
Current platforms for online social networks follow the client-server paradigm, one vendor provides the servers which provide the service to the users. While it is quite convenient for the users, the vendor carries all the costs.

Annual administration expenses are directly linked to the number of users. Some estimates claim costs of 1.05$ per single platform user per year in the case of Facebook. The most of these expenses are server administration costs. For YouTube, LastFM and MySpace the principle is the same. There as well, the most crucial resources, storage space and bandwidth, are provided by servers and the capacities of the clients are unused. A second limitation of client-server-based solutions is the limited innovation progress. Many providers keep the control to add new features and functions to the system, which limits the extendability of the platform, such as in Last.FM. The p2p paradigm can help to drastically lower the costs for a provider, as we have shown in [LPG+07], as administration costs and the service load are shared among the users of the system.

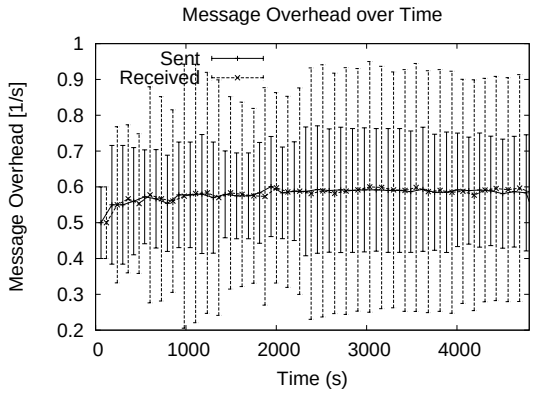
Several distributed storage applications like OceanStore [KBC+00] address the reliable handling of large data objects, but do not offer user interaction capabilities. Wuala [Cal] enhances file-sharing with communication functionality, but does not offer common collaboration functions of online social



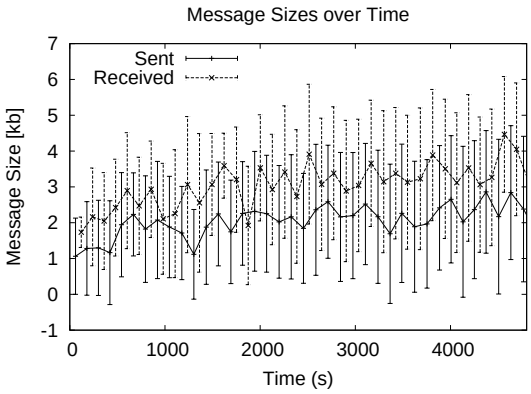
(a) Disk Usage of LifeSocial.KOM



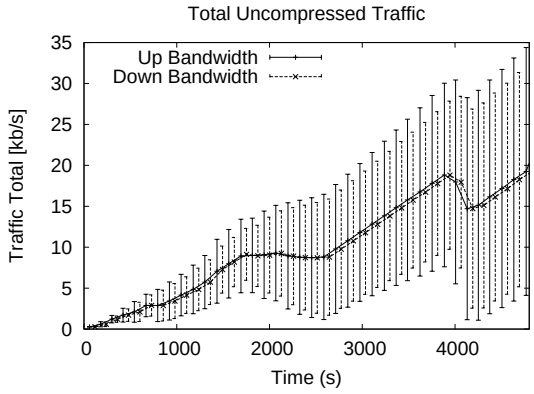
(b) Bandwidth Consumption of Compress Data



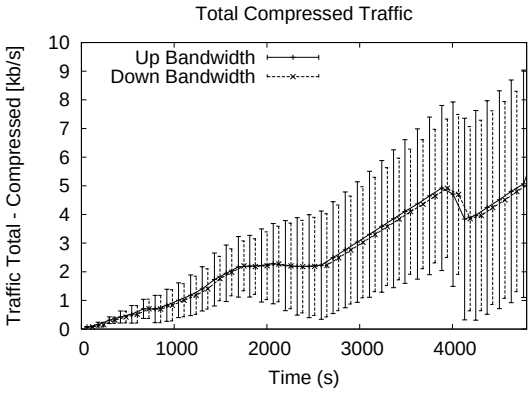
(c) Messaging Overhead



(d) Message Size Distribution



(e) Bandwidth Consumption of Uncompressed Data



(f) Total Traffic Overhead

Figure 107: Monitoring View on the Overhead of LifeSocial.KOM

networks. Groove [Edw02] is a p2p-based collaborative tool providing some limited support for group interaction, like message boards, chatting and shared folders. However, it only scales to a dozens of users in a group and does not support millions like in common multimedia online communities. The friend-of-a-friend approach [GR08] interlinks web profiles of people based on their friendship status. Although being decentralized, it assumes that every user sets up his personal web server to host his profiles. Distributed online social networks, as described in [YLL+09], assume that users operate small web servers with their data and does not focus on data availability in case of joining and leaving peers. Freenet [Cla] is a p2p-based website platform, providing anonymous and resilient website and data

storage. Although anonymity is achieved, the performance of Freenet is very questionable, mainly due to the effects of the security mechanisms.

The research field of p2p-based online social networks is young and comprises only a few approaches. Safebook [CMS09] creates a new p2p overlay for a secure online social network with specific rules for the topology. Although the idea is interesting, the performance of the approach is still to be evaluated. Peerson [BSVD] uses a structured p2p overlay es well, but eventually had issues with the chosen p2p overlay OpenDHT [RGK+05]. Cryptree [GMSW06] provides a file sharing network with social components like profiles and friends. It resulted in the commercial application Wuala and uses a server for the key management and as data backup, and thus comprises a single point of failure.

Structured p2p overlays fit more to the idea of online social networks as user generated objects and information have unique identifiers (e.g. "username_friendlist") and need to be retrieved quickly in the application. Our solution for a p2p-based online social network, LifeSocial.KOM, uses a structured p2p overlay for fault-tolerant and efficient data storage and builds the desired social functionality in a plugin-based manner on top.

7.6 CONCLUSIONS

Online social networks are very popular nowadays in the Internet. They allow users to create profiles and photo albums, link their friends, comment each other and provide several other communication and collaboration tools. Websites like Facebook, LinkedIn and Bebo show remarkable growth rates and contain altogether several hundred Million users worldwide. Using the p2p paradigm, like in Skype or file sharing applications, the load for operating the infrastructure can be shifted from the service providers to the service consumers, the users. In order to provide a platform for reliable, monitorable and manageable quality of service supporting a wide set of services and functionality we proposed LifeSocial.KOM, a p2p platform for plugin-based p2p applications, implementing a secure online social network.

LifeSocial.KOM uses the structured p2p overlay FreePastry for interconnecting the participating nodes and providing with PAST a reliable, replicated data storage. We completed the platform with a Message Dispatcher, Storage Dispatcher and Information Cache. It further has a plugin-based architecture on top of the p2p overlay, implementing the functionality of online social networks and some additional collaboration functionality, like a shared Whiteboard. We also extended the core architecture of LifeSocial.KOM with a security layer providing authenticated, secure communication and a user-based data access control, addressing the privacy concerns in current centralized online social networks. Details to our security approach are given in Section 7.3.3. In order to monitor the quality of service in the distributed architecture, we designed and implemented the monitoring solution SkyEye.KOM for structured p2p overlays. We evaluated LifeSocial.KOM in a testbed showing the feasibility of the approach and the distribution of the load among the participating nodes.

We presented a general p2p-based platform for building p2p applications and in specific for secure online social networks, which provides the desired functionality of today's popular online social networks while eliminating the operational costs for the service provider. We believe that the p2p paradigm would drastically help online social network providers to cut their costs and be profitable as well as that online social networks are the next big application area for p2p-based solutions.

We demonstrated with LifeSocial.KOM the feasibility in a real application to monitor all layers of the p2p platform. We also showed and motivated the automated adaptation of the used mechanisms in order to comply with the quality goals. This complex scenario of online social networks with several functionality and apps is popular in the Internet with millions of users and operational costs. With LifeSocial.KOM, we have demonstrated a p2p-based platform that allows for rapid application development with the benefit of load and cost distribution of the deployed apps. Thus, we demonstrated with this application scenario that complex applications consisting of several functionality and optimization goals can easily be built, while resulting in no operational costs for the provider and controlled and monitored quality of service. With this, the p2p paradigm becomes more mature for commercial applications and a feasible alternative to existing IT infrastructures.

8 CONCLUSIONS AND OUTLOOK

We must always remember with gratitude and admiration the first sailors who steered their vessels through storms and mists, and increased our knowledge of the lands of ice.
- Roald Amundsen
Superhuman effort isn't worth a damn unless it achieves results.
- Ernest Shackleton

This chapter summarizes the problem statements, solutions and evaluation results that have been presented in the dissertation. We conclude the findings of this dissertation and discuss their implications. The dissertation ends with an outlook on future research challenges in the field of monitoring and management of p2p systems.

8.1 CONCLUSIONS

The p2p paradigm has gained large impact over the last decade. Starting with file sharing, voice over IP and content distribution, its application range focused in general on data-centric applications that provided their functionality for free. Current p2p applications aim at dedicated functionality with a p2p software solution that is optimized for this specific single application. However, dynamism in the scope of the scenario, the user behavior and the peer heterogeneity lead to changes in the behavior of the p2p system that is currently not observable or even controllable on a large scale. One specific functionality and mechanism may be evaluated well and its behavior may be predictable. However, in combination with other mechanisms, the behavior of the p2p system is unknown in advance. In order to facilitate the development of high-quality p2p applications, modular components and mechanisms with dedicated functionality may be combined in future, to create a multi-purpose p2p platform. In this case, the need for solutions regarding the monitoring and management of the quality of service of the p2p system is evident. A second challenge manifests with regard to multi-component p2p platforms. All functional components operate on the same set of resources of the p2p network. In order to overcome the limitation of the peer lifetime in the process of resource provisioning, which may result in resource provision failures, the need for a consistent view on the capacities in a p2p system is apparent. High quality p2p applications and services require reliable resource provision from the p2p system, without interruptions or degrading capacities.

This dissertations provides several contributions to the field of monitoring and management of p2p systems. In Chapter 1, we introduce the history of p2p systems and point out their limitations in terms of providing controlled quality of service. We delineate the challenges inherent in monitoring the quality of a live p2p system and its limited available resources, and motivate the management of these challenges in order to enable p2p systems to act as reliable IT infrastructures for a wide set of applications. Chapter 2 discusses the background of p2p systems and presents various functional components in p2p systems, including overlays, content distribution mechanisms, replication mechanisms, publish/subscribe, application layer multicasting, distributed computation and accounting. The quality of service provided by these individual components is challenging to predict or configure, and even more challenging when they are combined. However, with a dedicated component for monitoring and management of p2p systems, the complexity can be concentrated and addressed consistently.

In Chapter 3, we motivate and design, in Chapter 4, we evaluate, SkyEye.KOM, an approach for monitoring both the global system state and the capacities of the peers in the p2p system. For that, we create a tree topology on top of a structured, KBR-compliant p2p overlay and implement protocols for gathering both system-specific and peer-specific information. Each peer identifies its Coordinator position in the tree based on its Peer ID. In the case of monitoring system-specific information, the tree structure is established and maintained by periodic metric update messages sent to Parent-Coordinators

in the tree. Each Coordinator aggregates on its level the view from lower levels and forwards it on towards the root. Once the root receives the aggregated global view on the p2p system, it disseminates the information back towards the leaves of the tree. The evaluation shows that the tree structure is very robust against churn and requires very low traffic overhead for maintenance. The results are fresh and configurable in their precision. With around 110 byte/s per peer, SkyEye.KOM is able to provide a fresh view on the global system state to all peers, which is fresher than 200 seconds. Regarding the monitoring of peer-specific information, SkyEye.KOM provides the functionality of capacity-based peer searching. To this end, it extends the monitoring tree with Support Peers that are capable of bearing the load resulting from monitoring the peer capacities. Evaluation shows that the dynamic allocation of load to Support Peers is effective and no peer is overloaded. Monitoring information is fresh with an average age of 200s, and queries are resolved in average within 5-6 hops in the monitoring tree. With SkyEye.KOM, we are able to observe the current behavior of a p2p system and provide access to the capacities of the peers in the p2p system to a full extent.

In Chapter 5, we address the issue of unreliable resource provisioning in p2p systems due to the imminent danger of peer failures. We present P^3R^3O.KOM, a p2p protocol for reliable resource provision which allows the reservation of resources for time scales that go beyond the average lifetime of a peer. A dedicated reservation management set is created per reservation which uses SkyEye.KOM to identify suitable capacities in the p2p system for providing the reservation and maintains a set of resource providers in a dynamic quantity that makes sure that even under churn, the resource reservation is not endangered. We compared two approaches for determining the quantity of redundant peers: the probabilistic buffer assignment (PBA) approach and the redundant peer assignment (RPA) approach. The RPA approach outperforms the PBA approach in terms of performance and provides with at least 4 redundant peers a reservation success ratio of 100% in the case that enough suitable peers are in the network.

Chapter 6 discusses, first, viable approaches to control the quality of service provided in p2p systems. We show that the configuration of a specific mechanism has a great influence on its resulting behavior. In the case of overlay bandwidth management, we show that through prioritized handling of messages in the p2p overlay, we are able to address routing delay and failure very efficiently. Extending the scope of information that is available for a decision allows for more versatile optimization goals. We show through the example of p2p-based multimedia streaming that adaptable optimization goals may be addressed with an extended monitoring view. To conclude, we introduced the concepts of autonomic computing, especially the monitor, analyze, plan and execute (MAPE) cycle that enables a system to autonomously react on variations in the environment in order to maintain a given quality of service goal. We implemented a self-optimization cycle of autonomic computing for p2p systems, namely SkyNet.KOM, using SkyEye.KOM as monitoring component. We defined both static and adaptive rules for the reconfiguration of the system in the cases where precision quality metric goals are missed. With the example of Chord, the hop count metric as well as the finger table size as parameter we demonstrated and evaluated the feasibility of our management approach for p2p systems. The evaluation shows that the MAPE cycle is effective and preset quality intervals for the hop count in Chord are reached and held. The adaptive rules, which take the magnitude of quality deviation into account, lead to a quicker convergence of the measured quality towards the desired quality interval. The management approach for p2p systems, SkyNet.KOM, is both very well performing, in other words, a few iterations are sufficient for reaching a valid configuration and quality level, and very lightweight as its protocol messages are integrated in SkyEye.KOM, which itself is very lightweight.

In Chapter 7, we present an application scenario for the proposed monitoring and management mechanism. We introduce LifeSocial.KOM, a p2p platform with various functional building blocks, which hosts a set of plugins, implementing the application of online social networks. Due to the large set of involved components, SkyEye.KOM is implemented prototypically to monitor the behavior of the p2p system, specifically the peer resources, overlay behavior, and SkyEye.KOM behavior, as well as specific information about the online social network. LifeSocial.KOM is the first and most mature, totally distributed, p2p-based online social network that demonstrates the feasibility and benefits of component-based p2p application development as well as a dedicated monitoring and management component in this p2p platform.

In terms of the implications of our work, we identified that having a dedicated component in a p2p platform for monitoring and management facilitates the constant supervision of the behavior of the p2p system. Therefore, this provides a basis for commercial utilization. The monitoring information obtained is crucial for a system provider deploying a p2p application in the Internet. In addition to the passive observing of the system state, we gave a tool at the hand of a system provider to control the quality of the p2p system in the wild. A set of p2p components "of the shelf" can be flexibly combined and reused without the need to manually configure them for a given scenario, expected user behavior or peer heterogeneity.

Through the integrated self-configuration following the MAPE cycle of the autonomic computing, a suitable configuration for the components in the p2p system can be automatically approached in any given scenario. The system provider is responsible for defining valid quality intervals for the system behavior and integrating suitable adaptation rules based on expert knowledge. Thus, the quality of service of the p2p system is managed automatically. Regarding the quality of the resource provision, we introduced with P^3R^3O.KOM a mechanism that enables the reliable reservation of resources in unreliable p2p systems. With this, we create a platform for hosting applications and services that require a dedicated amount of resources. Inspired by the idea of service-oriented architectures, p2p platforms may host services and use services offered in the p2p system and build service oriented applications that benefit from the distributed modular application composition. Through the systematic approach, the service quality of the p2p system and the resources in the p2p network became observable and manageable, an aspect that renders the IT infrastructures based on the p2p paradigm reliable and prepared for commercial utilization.

8.2 OUTLOOK

The research in the field of monitoring and management of p2p systems can be extended in various ways. While the monitoring mechanism, SkyEye.KOM, is used in the management approach to provide sufficient information on the p2p system, SkyEye.KOM itself could be optimized in its behavior through applying SkyNet.KOM. SkyEye.KOM offers modifiable parameters for the update intervals, which may be adapted in order to meet preset freshness intervals. In order to use SkyEye.KOM in a real scenario more efficiently, it could be integrated in tree-based p2p overlays such as Globase.KOM.

Besides the improvements of the efficiency of SkyEye.KOM, security issues should be addressed in future to counterfeit possible attacks of peers trying to forge or modify monitoring results. The plausibility of the monitoring scope would need to be crosschecked with monitoring results from lower levels. The additional local view on the system state or peer capacity of a single peer added to the monitoring view of a Domain might only have an effect on the resulting view which is antiproportional to the Domain size.

SkyEye.KOM provides a global view on the behavior of the p2p system under variable parameter setups. Through the systematic testing of the effects, the variation of specific parameters both the p2p system can be calibrated and its quality range identified. This quality scope marks the possible states which the p2p system may aim at and is reachable through setting a specific system configuration. Through systematic comparison of the quality scopes of individual p2p components, a benchmark may be derived comparing the quality boundaries of various solutions for a specific functionality. Benchmarking of p2p-based mechanisms is, in general, a noteworthy approach to systematically and comparably evaluate a mechanism. Through defining a scenario, peer behavior and relevant metrics, a setup is created in which any approach implementing the same functionality can be evaluated and evaluation results can be compared.

Through calibrating, the possible quality scope of a mechanism can be identified. Using benchmarking, a comparison can be made among individual solutions, which allows for the identification of the effects of specific design decision of a mechanism on the resulting quality of service it provides. Through this, a systematic engineering process for p2p systems and applications that gives information at the hand of developers is possible, and related design decisions can follow, in order to meet specific quality requirements of an envisioned p2p system. Using monitoring and management, the resulting p2p system is observed and controlled in its behavior. However, a quality optimized system design allows for the desired quality levels to be reached in a more efficient manner.

The management of the resources in the p2p system is the next logical step; the implementation of p2p-based service oriented architectures could be envisioned. In a service-oriented architecture, services are provided through their distribution on reliable nodes. Now, with P^3R^3O.KOM, p2p systems may also provide reliable "virtual nodes" with sufficient resource capacities. On top of this, services may be deployed and used. However, further research should be conducted in this field to elaborate the benefits and challenges arising from the p2p characteristics in the service-oriented architecture.

For the management of the quality of service of p2p systems, we proposed both a static and a dynamic set of rules to adapt to the configuration in case of an observed quality violation. However, these rules are manually implemented using known interdependencies between the parameters and metrics. As a next step, these rules should be derived automatically through identifying the interdependencies between parameters and metrics in long observation periods. Using machine learning approaches, the management solution should systematically vary the parameter settings of the p2p system and learn the effects on the quality metrics. As a result of the learning process, for each metric, a dependency matrix can be created identifying the core parameters, which then can be adapted in corresponding rules.

Extending the scope of monitoring and management of p2p systems from a provider-oriented view to a self-managing view, we envision a p2p system that first learns its own abilities and the interdependencies between its configuration and the resulting behavior of the p2p system (i.e. like an infant). A self-aware p2p system could emerge that identifies the quality needs of the users and automatically adapts to these. In addition, using the ability to manage the resources in the p2p network, the emerging self-aware p2p system may create a self-driven market place for resources and capacities, implementing a p2p cloud in a self-managed manner as we sketched in [GKL+08]. As a final research agenda, we envision a p2p system that uses its ability to monitor its behavior, identify the effects of its re-configuration and its potential to fully utilize the resources available in the p2p network. This new entity, based on the worldwide contributions of the peer capacities, would first aim to keep itself alive and second, would provide useful services to mankind, such as calculating formulae to cure cancer or create programs that are creative and smarter than humans. Although the worldwide interconnection and utilization of all computer devices through a self-conscious distributed network offers a great chance to address critical research questions of mankind through the mere resource capacity, we must not forget that research and technology can bring both blessing and death to mankind.

BIBLIOGRAPHY

[AAG+05] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth: *The Essence of P2P: A Reference Architecture for Overlay Networks*. In *IEEE P2P '05: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 11–20, 2005. (Cited on page 14.)

[AAGW04] K. Albrecht, R. Arnold, M. Gähwiler, and R. Wattenhofer: *Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave*. In *IEEE P2P '04: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 227–234, 2004. (Cited on page 66.)

[ABB+08] M. Abid, A. Berl, Z. Boudjemil, A. Cheniour, S. Davy, S. Denazis, A. Galis, J.-P. Gelas, C. Fahy, A. Fischer, J. Rubio Loyola, L. Lefèvre, D. Macedo, L. Mamatas, H. de Meer, Z. Movahedi, J. Strassner, and H. Zimmermann: *Autonomic Internet Initial Framework - Deliverable D6.1*. Technical report, INRIA, 2008. (Cited on page 165.)

[ACK+02] 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. (Cited on page 16.)

[ACMD+03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt: *P-Grid: A Self-organizing Structured P2P System*. SIGMOD Record, 32(3):29–33, 2003. (Cited on page 14.)

[AGG+07] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez: *Exploring VoD in P2P Swarming Systems*. In *INFOCOM '07: Proceedings of the International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 2571–2575, 2007. (Cited on page 143.)

[AKWL05] D. Agrawal, L. Kang-Won, and J. Lobo: *Policy-based Management of Networked Computing Systems*. IEEE Communications Magazine, 43:69 – 75, 2005. (Cited on page 165.)

[Ale] Alexa: *Alexa - The Web Information Company*. http://www.alexa.com. (Cited on page 173.)

[And04] D. P. Anderson: *BOINC: A System for Public-Resource Computing and Storage*. In *IEEE GRID '04: Proceedings of International Workshop on Grid Computing*, pages 4–10, 2004. (Cited on page 16.)

[Apa96] Apache Software Foundation: *Apache HTTP Server*. http://httpd.apache.org/, 1996. (Cited on page 4.)

[ARZC05] M. Amoretti, M. Reggiani, F. Zanichelli, and G. Conte: *SP2A: Enabling Service-Oriented Grids using a Peer-to-Peer Approach*. In *IEEE WETICE '05: Proceedings of International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 301–304, 2005. (Cited on page 127.)

[AX02] A. Andrzejak and Z. Xu: *Scalable, Efficient Range Queries for Grid Information Services*. In *IEEE P2P '02, Proceedings of the International Conference on Peer-to-Peer Computing*, pages 33–40, 2002. (Cited on page 15.)

[BACH08] E. Buyukkaya, M. Abdallah, R. Cavagna, and S.-Y. Hu: *GROUP: Dual-Overlay State Management for P2P NVE*. In *IEEE ICPADS '08: Proceedings of the International Conference on Parallel and Distributed Systems*, pages 817–822, 2008. (Cited on page 164.)

[BBK02] S. Banerjee, B. Bhattacharjee, and C. Kommareddy: *Scalable Application Layer Multicast*. In *ACM SIGCOMM '02: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 205–217, 2002. (Cited on page 16.)

[BCC+06] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang: *Improving Traffic Locality in BitTorrent via Biased Neighbor Selection*. In *IEEE ICDCS '06: Proceedings of the International Conference on Distributed Computing Systems*, page 66, 2006. (Cited on page 168.)

[BCCW88] A. Broder, M. A. Charette, B. Croll, and T. Whitted: *What can we learn by Benchmarking Graphics Systems? (Panel Session)*. In *ACM SIGGRAPH'88: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, page 348, 1988. (Cited on page 19.)

[BCD+08] N. Brake, J. Cordy, E. Dan, M. Litoiu, and V. Popescu: *Automating discovery of software tuning parameters*. In *ACM SEAMS '08: Proceedings of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 65–72, 2008. (Cited on page 165.)

[BDA07] P. Bocciarelli, A. D'Ambrogio, and M. Angelaccio: *QShare: QoS-Enabled Description and Discovery of Services in SOA-Based P2P Applications*. In *IEEE WETICE '07: Proceedings of International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 159–166, 2007. (Cited on page 127.)

[Ber07] R. Berbner: *Dienstgüteunterstützung für Service-orientierte Workflows*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2007. (Cited on pages 3, 127, and 163.)

[Bey06] D. Beyer: *Relational Programming with CrocoPat*. In *ACM ICSE '06: Proceedings of International Conference on Software Engineering*, pages 807–810, 2006. (Cited on page 165.)

[BFHM04] A. R. Butt, X. Fang, Y. C. Hu, and S. P. Midkiff: *Java, Peer-to-Peer, and Accountability: Building Blocks for Distributed Cycle Sharing*. In *Proceedings of USENIX Virtual Machine Research and Technology Symposium*, pages 163–176, 2004. (Cited on page 16.)

[BGMGM03] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani: *Estimating Aggregates on a Peer-to-Peer Network*. Technical report 2003-24, Stanford InfoLab, 2003. (Cited on page 66.)

[Bit] BitTorrent: *BitTorrent DNA*. http://www.bittorrent.com/dna/. (Cited on page 143.)

[BKM08a] D. Bradler, J. Kangasharju, and M. Mühlhäuser: *Evaluation of Peer-to-Peer Overlays for First Response*. In *IEEE PerCom '08: Proceedings of the International Conference on Pervasive Computing and Communications*, pages 463–467, 2008. (Cited on page 167.)

[BKM08b] D. Bradler, J. Kangasharju, and M. Mühlhäuser: *Systematic First Response Use Case Evaluation*. In *Workshop on Mobile and Distributed Approaches in Emergency Scenarios*, 2008. (Cited on page 167.)

[BL91] T. Berners-Lee: *World Wide Web*. http://info.cern.ch/, 1991. (Cited on page 12.)

[BNW03] R. Bless, K. Nichols, and K. Wehrle: *RFC 3662: A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services*, 2003. (Cited on page 134.)

[BRP+05] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang: *The Impact of Heterogeneous Bandwidth Constraints on DHT-Based Multicast Protocols*. In *IPTPS '05: Proceedings of the International Workshop on Peer-To-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 115–126, 2005. (Cited on page 139.)

[BSVD] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta: *PeerSoN: P2P Social Networking - Early Experiences and Insights*. In *ACM SNS ' 09: Proceedings of the ACM Workshop on Social Network Systems Social Network Systems*. (Cited on pages 174 and 202.)

[BT05] A. Binzenhöfer and K. Tutschku: *DNA - A P2P-based Framework for Distributed Network Management*. In *Proceedings of KiVS Kurzbeiträge und Workshop*, pages 135–138, 2005. (Cited on page 164.)

[BTadG+05] A. Binzenhöfer, K. Tutschku, B. auf dem Graben, M. Fiedler, and P. Arlos: *A P2P-Based Framework for Distributed Network Management*. In *EuroNGI Workshop '06: Proceedings of the International Workshop of the EURO-NGI Network of Excellence on Wireless Systems and Network Architectures in Next Generation Internet*, volume 3883 of *Lecture Notes in Computer Science*, pages 198–210, 2005. (Cited on page 164.)

[BVV03] R. Bhagwan, G. Varghese, and G. Voelker: *CONE: Augmenting DHTs to Support Distributed Resource Discovery*. Technical Report CS2003-0755, University of California, San Diego, 2003. (Cited on page 67.)

[Cal] Caleido AG: *Wuala - Secure Online Storage*. http://www.wua.la. (Cited on page 200.)

[CB08] D. R. Choffnes and F. E. Bustamante: *Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems*. In *ACM SIGCOMM '08: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 363–374, 2008. (Cited on page 168.)

[CD88] G. F. Coulouris and J. Dollimore: *Distributed Systems: Concepts and Design*. Addison-Wesley, 1988. (Cited on page 11.)

[CDK+03] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh: *SplitStream: High-Bandwidth Content Distribution in Cooperative Environments*. In M. F. Kaashoek and I. Stoica (editors): *IPTPS '03: Proceedings of the International Workshop on Peer-To-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 292–303, 2003. (Cited on page 175.)

[CFL+06] R. Carroll, C. Fahy, E. Lehtihet, S. van der Meer, N. Georgalas, and D. Cleary: *Applying the P2P Paradigm to Management of Large-scale Distributed Networks using a Model Driven Approach*. In *IEEE NOMS '06: Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, 2006. (Cited on page 163.)

[CFSD90] J. Case, M. Fedor, M. Schoffstall, and J. Davin: *RFC 1157: Simple Network Management Protocol (SNMP)*. http://www.ietf.org/rfc/rfc1157.txt, 1990. (Cited on pages 65 and 163.)

[CG01] B. Carlsson and R. Gustavsson: *The Rise and Fall of Napster - An Evolutionary Approach*. In *Active Media Technology*, pages 347–354, 2001. (Cited on pages 3 and 13.)

[CGM09] R. Canonico, C. Guerrero, and A. Mauthe: *Content Distribution Infrastructures for Community Networks*. Journal on Computer Networks, 53(4):431–433, 2009. (Cited on page 16.)

[Cha03] Y. Chawathe: *Scattercast: An Adaptable Broadcast Distribution Framework*. Springer Multimedia Systems, 9(1):104–118, 2003. (Cited on page 16.)

[CKWC03] J. Cao, O. M. K. Kwong, X. Wang, and W. Cai: *A Peer-to-Peer Approach to Task Scheduling in Computation Grid*. In *Springer GCC '04: Proceedings of the International Workshop on Grid and Cooperative Computing*, volume 3032 of *Lecture Notes in Computer Science*, pages 316–323, 2003. (Cited on page 127.)

[Cla] I. Clarke: *Freenet - The Free Network*. http://freenetproject.org. (Cited on page 201.)

[Clio2] Clip2 - The Gnutella Developer Forum: *Gnutella*. http://rfc-gnutella.sourceforge.net/developer/stable/index.html, 2002. (Cited on page 13.)

[CMS09] L. A. Cutillo, R. Molva, and T. Strufe: *Safebook: Feasibility of Transitive Cooperation for Privacy on a Decentralized Social Network*. In *IEEE WOWMOM '09: Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, 2009. (Cited on pages 174 and 202.)

[CN95] S. Chutani and H. J. Nussbaumer: *Network Management and System-level Diagnosis*. In *IEEE ICCCN '95: Proceedings of theInternational Conference on Computer Communications and Networks*, pages 280–287, 1995. (Cited on page 163.)

[Coh] B. Cohen: *BitTorrent*. http://www.bittorrent.com. (Cited on page 15.)

[Coh03] B. Cohen: *Incentives Build Robustness in BitTorrent*. In *P2PECON '03: Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003. (Cited on page 146.)

[CRB+03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker: *Making Gnutella-like P2P systems scalable*. In *SIGCOMM '03: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 407–418, 2003. (Cited on page 139.)

[Cro82] D. H. Crocker: *RFC 822: ARPA Internet Text Messages*. http://tools.ietf.org/html/rfc822, 1982. (Cited on page 12.)

[CRZ00] Y.-H. Chu, S. G. Rao, and H. Zhang: *A Case for End System Multicast*. In *ACM SIGMETRICS '00: Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, 2000. (Cited on page 16.)

[CT02] M. Calzarossa and S. Tucci (editors): *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, volume 2459 of *Lecture Notes in Computer Science*, Springer, 2002. (Cited on page 19.)

[CWS01] C. Cavanaugh, L. R. Welch, and B. Shirazi: *Towards a characterization of quality of service management approaches in distributed, real-time systems*. In *IEEE IPDPS '01: Proceedings of the International Parallel and Distributed Processing Symposium*, pages 970 – 977, 2001. (Cited on page 163.)

[DAR69] DARPA: *Arpanet (Advanced Research Projects Agency Network)*, 1969. (Cited on page 12.)

[Dar05] V. Darlagiannis: *Overlay Network Mechanisms for Peer-to-Peer Systems*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2005. (Cited on page 14.)

[DG97] M. Dorigo and L. M. Gambardella: *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. IEEE Transactions on Evolutionary Computation, 1(1):53–66, 1997. (Cited on page 166.)

[DG05] X. Dai and J. C. Grundy: *Off-Line Micro-payment System for Content Sharing in P2P Networks*. In *ICDCIT '05: Proceedings of the International Conference on Distributed Computing and Internet Technology*, volume 3816 of *Lecture Notes in Computer Science*, pages 297–307, 2005. (Cited on page 16.)

[DHT] T. Do, K. Hua, and M. Tantaoui: *P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment*. In *IEEE ICC '04: Proceedings of the International Conference on Communications*, pages 1467–1472. (Cited on page 143.)

[DKK+01] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica: *Wide-Area Cooperative Storage with CFS*. In *ACM SOSP '01: Proceedings of Symposium on Operating Systems Principles*, pages 202–215, 2001. (Cited on page 15.)

[dMT02] H. de Meer and K. Tutschku: *A Performance Management Architecture for Peer-to-Peer Services based on Application-level Active Networks*. In *IEEE NOMS '02: Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, pages 927–929, 2002. (Cited on page 163.)

[DNF05] G. Doyen, E. Nataf, and O. Festor: *A Hierarchical Architecture for a Distributed Management of P2P Networks and Services*. In *Springer DSOM'05: Proceedings of the International Workshop on Distributed Systems: Operations and Management*, volume 3775 of *Lecture Notes in Computer Science*, pages 257–268, 2005. (Cited on page 67.)

[DPTS06] G. Denaro, M. Pezzè, D. Tosi, and D. Schilling: *Towards Self-adaptive Service-oriented Architectures*. In *ACM SIGSOFT TAV-WEB'06: Proceedings of the Workshop on Testing, Analysis, and Verification of Web Services and Applications*, pages 10–16, 2006. (Cited on page 164.)

[DR01] P. Druschel and A. I. T. Rowstron: *PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility*. In *IEEE HotOS '01: Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 75–80, 2001. (Cited on pages 4, 15, 174, and 176.)

[DR02] J. Daemen and V. Rijmen: *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002. (Cited on page 186.)

[DR05] W. Dan and Z. Rongjuan: *A Layered Resource Management Model in P2P System*. In *IEEE PDCAT '05: Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 569–572, 2005. (Cited on pages 66 and 127.)

[DvdGH+06] L. Durham, J. van de Groenendaal, J. He, J. Hobbs, M. Milenkovic, and Y. Mazin: *Platform Support of Autonomic Computing: an Evolution of Manageability Architecture*. Intel Technology Journal, 10:253–263, 2006. (Cited on page 65.)

[DZD+03] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica: *Towards a Common API for Structured Peer-to-Peer Overlays*. In *IPTPS '03: Proceedings of the International Workshop on Peer-To-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 33–44, 2003. (Cited on pages 24, 25, 27, 28, 69, 115, 120, 143, 144, 154, and 175.)

[Eck04] C. Eckert: *IT-Sicherheit. Konzepte - Verfahren - Protokolle, 3rd edition*. Oldenburg Verlag, 2004. (Cited on page 176.)

[Eck09] J. Eckert: *Cross-organizational Service-based Workflows - Solution Strategies for Quality of Service Optimization*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2009. (Cited on page 163.)

[Ecl] Eclipse Foundation: *Eclipse - An Open Development Platform*. http://www.eclipse.org/. (Cited on page 194.)

[Edw02] J. Edwards: *Peer-to-Peer Programming on Groove*. Addison-Wesley Professional, 2002. (Cited on page 201.)

[EK07] K. Eger and U. Killat: *Fair resource allocation in peer-to-peer networks (extended version)*. Journal on Computer Communications, 30(16):3046–3054, 2007. (Cited on page 167.)

[ESCK08] A. El-Saddik, J. Cha, and K. Kahol: *Haptics Technologies: Theory and Applications from a Multimedia Perspective*. In *ACM Multimedia '08: Proceedings of the International Conference on Multimedia*, pages 1163–1164, 2008. (Cited on page 192.)

[EV03] C. Estan and G. Varghese: *New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice*. ACM Transactions ond Computer Systems, 21(3):270–313, 2003. (Cited on pages 133 and 135.)

[Exn07] Exner: *Techniques for Approximation and Optimization of IBM DB2 UDB Performance Functions (in German)*. Master's thesis, University of Jena, Germany, 2007. (Cited on page 165.)

[FCC+03] Y. Fu, J. S. Chase, B. N. Chun, S. Schwab, and A. Vahdat: *SHARP: An Architecture for Secure Resource Peering*. In *ACM SOSP'03: Proceedings of the Symposium on Operating Systems Principles*, pages 133–148, 2003. (Cited on page 16.)

[FDB+07] F. L. Faucheur, B. Davie, P. Bose, C. Christou, and M. Davenport: *RFC 4860: Generic Aggregate Resource ReSerVation Protocol (RSVP) Reservations*. http://www.ietf.org/rfc/rfc4860.txt, 2007. (Cited on page 134.)

[FGS01] S. Floyd, R. Gummadi, , and S. Shenker: *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*. Technical report, 2001. (Cited on page 139.)

[FJ93] S. Floyd and V. Jacobson: *Random Early Detection Gateways for Congestion Avoidance*. IEEE/ACM Transactions on Networking, 1(4):397–413, 1993. (Cited on page 139.)

[FSB09] A. Fiorese, P. Simões, and F. Boavida: *A P2P-Based Approach to Cross-Domain Network and Service Management*. In *Springer AIMS '09: Proceedings of the International Conference on Autonomous Infrastructure, Management and Security*, pages 179–182, 2009. (Cited on page 164.)

[Fuj] Fujitsu: *Triole*. www.fujitsu.com/global/services/solutions/triole/. (Cited on page 165.)

[Gö5] M. Görtz: *Effiziente Echtzeitkommunikationsdienste durch Einbeziehung von Kontexten*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2005. (Cited on page 192.)

[GB09] M. Gharzouli and M. Boufaïda: *A Generic P2P Collaborative Strategy for Discovering and Composing Semantic Web Services*. In *IEEE ICIW '09: Proceedings of the International Conference on Internet and Web Applications and Services*, pages 449–454, 2009. (Cited on page 127.)

[GDS+03] 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 *ACM SOSP '03: Proceedings of the ACM Symposium on Operating Systems Principles*, volume 37 of *Operating Systems Review*, pages 314–329, 2003. (Cited on pages 3, 14, and 140.)

[GGRVA05] C. Guerrero, J. García-Reinoso, F. Valera, and A. Azcorra: *QoS Management in Fixed Broadband Residential Gateways*. In *Management of Multimedia Networks and Services*, volume 3754 of *Lecture Notes in Computer Science*, pages 338–349, 2005. (Cited on page 163.)

[GKL+08] K. Graffi, A. Kovacevic, N. Liebau, , and R. Steinmetz: *From Cells to Organisms: Long-Term Guarantees on Service Provisioning in Peer-to-Peer Networks*. In ACM (editor): *ACM SIGAPP NOTERE '08: Proceedings of the International Conference on New Technologies of Distributed Systems*, pages 1–6, 2008. (Cited on page 206.)

[GKP+08] K. Graffi, S. Kaune, K. Pussep, A. Kovacevic, and R. Steinmetz: *Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems*. In *ACM NOSSDAV '08: Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 99–104, 2008. (Cited on page 167.)

[GKWS07] K. Graffi, A. Kovacevic, K. Wulffert, and R. Steinmetz: *ECHoP2P: Emergency call handling over peer-to-peer overlays*. In *IEEE ICPADS '07: Proceedings of the International Conference on Parallel and Distributed Systems*, pages 1–10, 2007. (Cited on page 167.)

[GLS07] K. Graffi, N. Liebau, and R. Steinmetz: *Taxonomy of Message Scheduling Strategies in Context of Peer-to-Peer Scenarios*. Technical Report KOM-TR-2007-02, KOM - Multimedia Communications Lab, Technische Universität Darmstadt, Germany, 2007. (Cited on page 138.)

[GMHS07] K. Graffi, P. S. Mogre, M. Hollick, and R. Steinmetz: *Detection of Colluding Misbehaving Nodes in Mobile Ad Hoc and Wireless Mesh Networks*. In *IEEE GLOBECOM '07: Proceedings of the Global Communications Conference*, pages 5097–5101, 2007. (Cited on page 166.)

[GMSW06] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer: *Cryptree: A Folder Tree Structure for Cryptographic File Systems*. In *IEEE SRDS '06: Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pages 189–198, 2006. (Cited on page 202.)

[Gol89] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989. (Cited on page 166.)

[GPK+07] K. Graffi, K. Pussep, S. Kaune, A. Kovacevic, N. Liebau, and R. Steinmetz: *Overlay Bandwidth Management: Scheduling and Active Queue Management of Overlay Flows*. In *IEEE LCN '07: Proceedings of the Annual Conference on Local Computer Networks*, pages 334–342, 2007. (Cited on pages 145 and 167.)

[GPLS07] K. Graffi, K. Pussep, N. Liebau, and R. Steinmetz: *Taxonomy of Active Queue Management Strategies in Context of Peer-to-Peer Scenarios*. Technical Report KOM-TR-2007-01, KOM - Multimedia Communications Lab, Technische Universität Darmstadt, Germany, 2007. (Cited on page 138.)

[GPW05] D. Gregg, J. F. Power, and J. Waldron: *A Method-level Comparison of the Java Grande and SPEC JVM98 Benchmark Suites*. Concurrency - Practice and Experience, 17(7-8):757–773, 2005. (Cited on page 19.)

[GR08] J. Golbeck and M. Rothstein: *Linking Social Networks on the Web with FOAF: A Semantic Web Case Study*. In *AAAI '08: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1138–1143, 2008. (Cited on page 201.)

[Gra] K. Graffi: *LifeSocial.KOM*. http://www.lifesocial.org/. (Cited on pages 186 and 195.)

[Gri00] C. Griwodz: *Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, Multimedia Communications Lab (KOM), 2000. (Cited on page 143.)

[GSBB08] P. E. Guerrero, K. Sachs, S. Butterweck, and A. P. Buchmann: *Performance Evaluation of Embedded ECA Rule Engines: A Case Study*. In *EPEW '08: Proceedings of the European Performance Engineering Workshop on Computer Performance Engineering*, volume 5261 of *Lecture Notes in Computer Science*, pages 48–63, 2008. (Cited on page 19.)

[GSR+09] K. Graffi, D. Stingl, J. Rueckert, A. Kovacevic, and R. Steinmetz: *Monitoring and management of structured peer-to-peer systems*. In *IEEE P2P '09: Proceeding of the International Conference on Peer-to-Peer Computing*, pages 311–320, 2009. (Cited on pages 72 and 120.)

[GSS06] R. Gupta, V. Sekhri, and A. K. Somani: *CompuP2P: An Architecture for Internet Computing Using Peer-to-Peer Networks*. IEEE Transactions on Parallel Distributed Systems, 17(11):1306–1320, 2006. (Cited on page 16.)

[GvRB01] I. Gupta, R. van Renesse, and K. P. Birman: *Scalable Fault-Tolerant Aggregation in Large Process Groups*. In *IEEE DSN '01: Proceedings of the International Conference on Dependable Systems and Networks*, pages 433–442, 2001. (Cited on page 66.)

[HAL+02] J. Hwang, P. Aravamudham, E. D. Liddy, J. Stanton, and I. MacInnes: *Charging Control and Transaction Accounting Mechanisms Using IRTL (Information Resource Transaction Layer) Middleware for P2P Services*. In *QofIS '02: In Proceedings of the International Workshop on Quality of Future Internet Services*, volume 2511 of *Lecture Notes in Computer Science*, pages 239–249, 2002. (Cited on page 16.)

[HASG07] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas: *A Survey of Application-Layer Multicast Protocols*. IEEE Communications Surveys and Tutorials, 9(1-4):58–74, 2007. (Cited on page 16.)

[HBTK] P. Hurley, J. Boudec, P. Thiran, and M. Kara: *ABE: Providing a Low-Delay Service within Best-Effort*. IEEE Network, 15(3):60–69. (Cited on page 134.)

[HC09] C. P. Hall and A. Carzaniga: *Uniform Sampling for Directed P2P Networks*. In *Springer Euro-Par '09: Proceedings of International Euro-Par Conference*, pages 511–522, 2009. (Cited on page 66.)

[Hec05] O. Heckmann: *A System-oriented Approach to Efficiency and Quality of Service for Internet Service Providers*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2005. (Cited on page 164.)

[HM08] M. C. Huebscher and J. A. McCann: *A Survey of Autonomic Computing - Degrees, Models, and Applications*. ACM Compututing Surveys, 40(3), 2008. (Cited on page 164.)

[Hol92] J. H. Holland: *Adaptation in Natural and Artificial Systems*. MIT Press, 1992. (Cited on page 166.)

[Hol05] M. Hollick: *Dependable Routing for Cellular and Ad hoc Networks*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2005. (Cited on page 28.)

[Hor01] P. Horn: *Autonomic Computing: IBM's Perspective on the State of Information Technology*. http://www.research.ibm.com/autonomic/, 2001. (Cited on pages 150 and 164.)

[Hri06] C. E. Hrischuk: *A Tutorial on SIP Application Server Performance and Benchmarking*. In *CMG '06: Proceedings of the International Computer Measurement Group Conference*, pages 729–740, 2006. (Cited on page 19.)

[HRVM07] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib: *PeerSum: Summary Management in P2P Systems*. In *Journées Bases de Données Avancées*, 2007. (Cited on page 164.)

[HRVM08] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib: *Summary management in p2p systems*. In *ACM EDBT '08: Proceedings of the International Conference on Extending Database Technology*, ACM International Conference Proceeding Series, pages 16–25, 2008. (Cited on page 164.)

[HS05] D. Hausheer and B. Stiller: *PeerMint: Decentralized and Secure Accounting for Peer-to-Peer Applications*. In *NETWORKING '05: Proceedings of International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols*, volume 3462 of *Lecture Notes in Computer Science*, pages 40–52, 2005. (Cited on page 16.)

[HSL+06] O. Heckmann, R. Steinmetz, N. Liebau, A. Buchmann, C. Eckert, J. Kangasharju, M. Mühlhäuser, and A. Schürr: *Qualitätsmerkmale von Peer-to-Peer-Systemen*. Technical Report KOM-TR-2006-03, Multimedia Communications Lab, Technische Universität Darmstadt, 2006. (Cited on page 18.)

[HSS10] D. Hausheer, G. D. Stamoulis, and B. Stiller: *Special Issue of the International Journal on Network Management (IJNM) on 'Economic Traffic Management'*. International Journal of Network Management, 20(2):107, 2010. (Cited on page 163.)

[HTA05a] T. Hoßfeld, K. Tutschku, and F.-U. Andersen: *Mapping of File-Sharing onto Mobile Environments: Enhancement by UMTS*. In *IEEE PerCom Workshops '05: Proceedings of the Conference on Pervasive Computing and Communications Workshops*, pages 43–49, 2005. (Cited on page 139.)

[HTA05b] T. Hoßfeld, K. Tutschku, and F.-U. Andersen: *Mapping of File-Sharing onto Mobile Environments: Feasibility and Performance of eDonkey with GPRS*. In *IEEE WCNC '05: Proceedings of the IEEE Wireless and Communications and Networking Conference*, 2005. (Cited on page 139.)

[IBM] IBM Tivoli: *Tivoli*. www.ibm.com/tivoli. (Cited on page 165.)

[IES08] R. Iglesias and A. El-Saddik: *A Glimpse of Multimedia Ambient Intelligence*. In *ACM Multimedia '08: Proceedings of the International Conference on Multimedia*, pages 1159–1160, 2008. (Cited on page 192.)

[IFN02] A. Iamnitchi, I. T. Foster, and D. Nurmi: *A Peer-to-Peer Approach to Resource Location in Grid Environments*. In *IEEE HPDC '02: Proceedings of the International Symposium on High Performance Distributed Computing*, page 419, 2002. (Cited on page 67.)

[IKT04] S. Idreos, M. Koubarakis, and C. Tryfonopoulos: *P2P-DIET: An Extensible P2P Service that Unifies Ad-Hoc and Continuous Querying in Super-Peer Networks*. In *ACM SIGMOD '04: Proceedings of the International Conference on Management of Data*, pages 933–934, 2004. (Cited on page 66.)

[Int98] International Telecommunication Union ISO IEC 10746-2: *Itu-t recommendation x.902. open distributed processing - reference model*, 1995-1998. (Cited on page 17.)

[IOS03] IOS Press: *OSGi Service Platform, Release 3*, 2003. (Cited on page 175.)

[JMB09] M. Jelasity, A. Montresor, and Ö. Babaoglu: *T-Man: Gossip-based Fast Overlay Topology Construction*. Computer Networks, 53(13):2321–2339, 2009. (Cited on page 66.)

[JNJH09] B. John, P. Nair, E. John, and F. Hudson: *Performance Measurement of Single, Dual and Quad Core Machines Using SPEC CPU2006*. In *CSREA CDES'09: Proceedings of the 2009 International Conference on Computer Design*, pages 143–149, 2009. (Cited on page 19.)

[Karoo] M. Karsten: *QoS Signalling and Charging in a Multi-service Internet using RSVP*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2000. (Cited on page 134.)

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao: *OceanStore: An Architecture for Global-Scale Persistent Storage*. In *ACM ASPLOS '00: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, 2000. (Cited on pages 15 and 200.)

[KC81] R. E. Kahn and V. G. Cerf: *RFC 793: Transmission Control Protocol*. http://tools.ietf.org/html/rfc793, 1981. (Cited on pages 54, 56, and 73.)

[KC97] S. Khuri and T. Chiu: *Heuristic Algorithms for the Terminal Assignment Problem*. In *ACM SAC '97: Proceedings of the ACM Symposium on Applied Computing*, pages 247–251, 1997. (Cited on page 166.)

[KCH02] M.-S. Kim, M.-J. Choi, and J. W.-K. Hong: *A Load Cluster Management System using SNMP and Web*. International Journal on Network Management, 12(6):367–378, 2002. (Cited on page 65.)

[KDG03] D. Kempe, A. Dobra, and J. Gehrke: *Gossip-Based Computation of Aggregate Information*. In *IEEE FOCS '03: Proceedings of the Symposium on Foundations of Computer Science*, pages 482–491, 2003. (Cited on page 66.)

[Kel90] T. W. Keller: *SPEC Benchmarks and Competitive Results*. SIGMETRICS Performance Evaluation Review, 18(3):19–20, 1990. (Cited on page 19.)

[KGK+08] A. Kovacevic, K. Graffi, S. Kaune, C. Leng, and R. Steinmetz: *Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments*. In *IEEE ICPADS '08: Proceedings of the International Conference on Parallel and Distributed Systems*, pages 799–804, 2008. (Cited on page 19.)

[KHKS09] A. Khalid, M. A. Haye, M. J. Khan, and S. Shamail: *Survey of Frameworks, Architectures and Techniques in Autonomic Computing*. In *IEEE ICAS'09: Proceedings of the International Conference on Autonomic and Autonomous Systems*, pages 220–225, 2009. (Cited on page 164.)

[KHS09] A. König, M. Hollick, and R. Steinmetz: *A Stochastic Analysis of Secure Joint Decision Processes in Peer-to-Peer Systems*. In *IEEE ICC '09: International Conference on Communications*, pages 1–5, 2009. (Cited on page 114.)

[KKH+06] A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, and R. Steinmetz: *PeerfactSim.KOM - A Simulator for Large-Scale Peer-to-Peer Networks*. Technical Report Tr-2006-06, 2006. (Cited on page 160.)

[KKM+07] A. Kovacevic, S. Kaune, P. Mukherjee, N. Liebau, and R. Steinmetz: *Benchmarking Platform for Peer-to-Peer Systems*. it - Information Technology (Methods and Applications of Informatics and Information Technology), 49(5):312–319, 2007. (Cited on pages 69, 72, 120, 140, 146, and 160.)

[KLKP08] S. Kaune, T. Lauinger, A. Kovacevic, and K. Pussep: *Embracing the Peer Next Door: Proximity in Kademlia*. In *IEEE P2P '08: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 343–350, 2008. (Cited on pages 72, 120, and 168.)

[KLS07] A. Kovacevic, N. Liebau, and R. Steinmetz: *Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search*. In *IEEE P2P '07: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 87–96, 2007. (Cited on pages 72, 120, and 168.)

[KM02] T. Klingberg and R. Manfredi: *Gnutella 0.6*. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, 2002. (Cited on page 13.)

[Kov09] A. Kovacevic: *Peer-to-Peer Location-based Search: Engineering a Novel Peer-to-Peer Overlay Network*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2009. (Cited on page 26.)

[Koz90] J. R. Koza: *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Technical report, Stanford University, 1990. (Cited on page 166.)

[KRT+09] S. Kaune, R. C. Rumín, G. Tyson, A. Mauthe, C. Guerrero, and R. Steinmetz: *Unraveling BitTorrent's File Unavailability: Measurements, Analysis and Solution Exploration*. CoRR: The Computing Research Repository, abs/0912.0625, 2009. (Cited on page 15.)

[KSBB08] S. Kounev, K. Sachs, J. Bacon, and A. P. Buchmann: *A Methodology for Performance Modeling of Distributed Event-Based Systems*. In *IEEE ISORC '08: Proceedings of the International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 13–22, 2008. (Cited on page 19.)

[KSC91] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis: *Weighted Round-Robin Cell Multiplexing in a General-purpose ATM Switch Chip*. IEEE Journal on Selected Areas in Communications, 9(8):1265–1279, 1991. (Cited on page 138.)

[KSS97] I. Kirkwood, S. Shami, and M. Sinclair: *Discovering Simple Fault-Tolerant Routing Rules using Genetic Programming*. In *Springer ICANNGA '97: Proceedings of the Internation Conference on Artificial Neural Nets and Genetic Algorithms*, 1997. (Cited on page 166.)

[LBB08] B. Labno, M. Bubak, and B. Balis: *A P2P Approach to Resource Discovery in On-Line Monitoring of Grid Workflows*. In *Springer Euro-Par '08: Proceedings of the International Euro-Par Conference on Parallel Processing*, pages 37–46, 2008. (Cited on page 127.)

[LBS05] S. Liu, T. Basar, and R. Srikant: *Exponential-RED: A Stabilizing AQM Scheme for Low- and High-Speed TCP Protocols*. IEEE/ACM Transactions on Networking, 13(5):1068–1081, 2005. (Cited on page 139.)

[LC08] S.-C. Lo and Y.-T. Chiu: *Design of Content-Based Publish/Subscribe Systems over Structured Overlay Networks*. IEICE Transactions, 91-D(5):1504–1511, 2008. (Cited on page 15.)

[LCL+09] D. Li, Z. Chen, H. Liu, A. V. Vasilakos, and Y. Pan: *IPBGA: A hybrid P2P based Grid Architecture by using Information Pool Protocol*. Springer Journal of Supercomputing, 49(2):159–189, 2009. (Cited on page 127.)

[LDHM05] N. Liebau, V. Darlagiannis, O. Heckmann, and A. Mauthe: *Accounting in Peer-to-Peer-Systems*. In *Peer-to-Peer Systems and Applications*, Lecture Notes in Computer Science, pages 547–566, 2005. (Cited on page 16.)

[LFV08] J. Liebeherr, M. Fidler, and S. Valaee: *A System Theoretic Approach to Bandwidth Estimation*. CoRR: The Computing Research Repository, abs/0801.0455, 2008. (Cited on page 56.)

[LHK+06] N. Liebau, O. Heckmann, A. Kovacevic, O. Tafreschi, M. Fidler, A. Mauthe, and R. Steinmetz: *A Secure Quorum Based Membership Mechanism for P2P Systems*. In *AMCIS '06: Proceedings of Americas Conference on Information Systems*, 2006. (Cited on page 114.)

[Lie08] N. C. Liebau: *Trusted Accounting in Peer-to-Peer Environments - A Novel Token-based Accounting Scheme for Autonomous Distributed Systems*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2008. (Cited on page 16.)

[LK06] K. Loesing and S. Kaffille: *OpenChord*. http://sourceforge.net/projects/open-chord/, 2006. (Cited on pages 75 and 175.)

[LKR06] J. Liang, R. Kumar, and K. W. Ross: *The FastTrack Overlay: A Measurement Study*. Computer Networks: The International Journal of Computer and Telecommunications Networking, 50(6):842–858, 2006. (Cited on page 13.)

[LL95] J. Li and B. J. Leon: *A Formal Approach to Model SNMP Network Management Systems*. In *IEEE ICCCN '95: Proceedings of the International Conference on Computer Communications and Networks*, page 284, 1995. (Cited on page 163.)

[LL04] J. Li and D.-Y. Lim: *A Robust Aggregation Tree on Distributed Hash Tables*. In *Proceedings of the 2004 Student Oxygen Workshop*, 2004. (Cited on page 67.)

[LM97] D. Lin and R. Morris: *Dynamics of Random Early Detection*. In *ACM SIGCOMM '97: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 127–137, 1997. (Cited on page 139.)

[LPG+07] N. Liebau, K. Pussep, K. Graffi, S. Kaune, E. Jahn, A. Beyer, and R. Steinmetz: *The Impact Of The P2P Paradigm*. In *AMCIS '07: Proceedings of Americas Conference on Information Systems*, 2007. (Cited on page 200.)

[LPY+06] Q. Lian, Y. Peng, M. Yang, Z. Zhang, Y. Dai, and X. Li: *Robust Incentives via Multi-level Tit-for-Tat*. In *IPTPS '06: Proceedings of the International Workshop on Peer-to-Peer Systems*, 2006. (Cited on page 146.)

[LSACI08] S. Lorpunmanee, M. N. Sap, A. H. Abdullah, and C. Chompoo-Inwai: *An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment*. International Journal of Computer and Information Science and Engineering, 2008. (Cited on page 167.)

[LSM+05a] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil: *A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn*. In *Proc. of INFOCOM*, 2005. (Cited on page 156.)

[LZZ+04] V. M. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao: *Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet*. In *IPTPS '04: Proceedings of the International Workshop on Peer-To-Peer Systems*, pages 227–236, 2004. (Cited on pages 16 and 167.)

[MCBM] M. Mu, E. Cerqueira, F. Boavida, and A. Mauthe: *Quality of Experience Management Framework for Real-Time Multimedia Applications*. (Cited on page 142.)

[MdLH+06] J. A. McCann, R. de Lemos, M. C. Huebscher, O. F. Rana, and A. Wombacher: *Can Self-managed Systems be trusted? Some Views and Trends*. Knowledge Engineering Review, 21(3):239–248, 2006. (Cited on page 164.)

[MFW01] R. Mahajan, S. Floyd, and D. Wetherall: *Controlling High-Bandwidth Flows at the Congested Router*. In *IEEE ICNP '01: Proceedings of the International Conference on Network Protocols*, 2001. (Cited on page 139.)

[MGHS07] P. S. Mogre, K. Graffi, M. Hollick, and R. Steinmetz: *AntSec, WatchAnt, and AntRep: Innovative Security Mechanisms for Wireless Mesh Networks*. In *IEEE LCN '07: Proceedings of the Annual Conference on Local Computer Networks*, pages 539–547, 2007. (Cited on page 166.)

[MGHS10] P. S. Mogre, K. Graffi, M. Hollick, and R. Steinmetz: *A Security Framework for Wireless Mesh Networks*. Wireless Communications and Mobile Computing Special Issue: Architectures and Protocols for Wireless Mesh, Ad Hoc, and Sensor Networks, 2010. (Cited on page 166.)

[MH05] A. Mauthe and O. Heckmann: *Distributed Computing - GRID Computing*. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, pages 193–206, 2005. (Cited on page 16.)

[MHGS09] P. S. Mogre, M. Hollick, J. D. Gandía, and R. Steinmetz: *A Proportionally Fair Centralized Scheduler Supporting Spatial Minislot Reuse for IEEE 802.16 Mesh Networks*. In *QSHINE*, volume 22 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 556–566, 2009. (Cited on page 139.)

[MHS+09] P. S. Mogre, M. Hollick, R. Steinmetz, V. Dadia, and S. Sengupta: *Distributed Bandwidth Reservation Strategies to Support Efficient Bandwidth Utilization and QoS on a per-Link Basis in IEEE 802.16 Mesh Networks*. In *IEEE LCN '09: Conference on Local Computer Networks*, pages 301–304, 2009. (Cited on page 139.)

[Mic] Microsoft: *SimPastry*. http://research.microsoft.com/. (Cited on page 175.)

[MK05] R. Mason and W. Kelly: *G2-P2P: A Fully Decentralised Fault-Tolerant Cycle-Stealing Framework*. In *ACSW Frontiers '05: ACSW Workshops - the Australasian Workshop on Grid Computing and e-Research (AusGrid 2005) and the Third Australasian Information Security Workshop (AISW 2005)*, volume 44 of *CRPIT*, pages 33–39, 2005. (Cited on page 16.)

[MKBS08] P. Mukherjee, A. Kovacevic, M. Benz, and A. Schürr: *Towards a Peer-to-Peer Based Global Software Development Environment*. In *Proceedings of Software Engineering*, volume 121 of *LNI*, pages 204–216, 2008. (Cited on page 192.)

[MKS08] P. Mukherjee, A. Kovacevic, and A. Schürr: *Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Development*. In *Proceedings of Software Engineering (Workshops)*, volume 122 of *LNI*, pages 72–76, 2008. (Cited on page 192.)

[MLS08] P. Mukherjee, C. Leng, and A. Schürr: *Piki - A Peer-to-Peer based Wiki Engine*. In *IEEE P2P '08: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 185–186, 2008. (Cited on page 175.)

[MLTS08] P. Mukherjee, C. Leng, W. W. Terpstra, and A. Schürr: *Peer-to-Peer Based Version Control*. In *IEEE ICPADS '08: Proceedings of the International Conference on Parallel and Distributed Systems*, pages 829–834, 2008. (Cited on page 175.)

[MM02] P. Maymounkov and D. Mazières: *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*. In *IPTPS '02: Proceedings of the International Workshop on Peer-To-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65, 2002. (Cited on pages 14, 15, 134, 140, and 156.)

[MMKG06] L. Massoulié, E. L. Merrer, A.-M. Kermarrec, and A. J. Ganesh: *Peer Counting and Sampling in Overlay Networks: Random Walk Methods*. In *ACM PODC '06: Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 123–132, 2006. (Cited on page 66.)

[MMP06] S. Modafferi, E. Mussi, and B. Pernici: *SH-BPEL: A Self-healing Plug-in for WS-BPEL Engines*. In *ACM MW4SOC'06: Proceedings of the Workshop on Middleware for Service Oriented Computing*, pages 48–53, 2006. (Cited on page 164.)

[MYGRM09] I. Martinez-Yelmo, C. Guerrero, R. C. Rumín, and A. Mauthe: *A Hierarchical P2PSIP Architecture to Support Skype-like Services*. In *Euromicro PDP '09: Proceedings of the International Conference on Parallel, Distributed and Network-Based Processing*, pages 316–322, 2009. (Cited on page 15.)

[MYRGM08] I. Martinez-Yelmo, R. C. Rumín, C. Guerrero, and A. Mauthe: *Routing Performance in a Hierarchical DHT-based Overlay Network*. In *Euromicro PDP '08: Proceedings of the International Conference on Parallel, Distributed and Network-Based Processing*, pages 508–515, 2008. (Cited on page 15.)

[MYSG10] I. Martinez-Yelmo, I. Seoane, and C. Guerrero: *Fair Quality of Experience (QoE) Measurements Related with Networking Technologies*. In *WWIC '10: Proceedings of the International Conference on Wired/Wireless Internet Communications*, volume 6074 of *Lecture Notes in Computer Science*, pages 228–239, 2010. (Cited on page 163.)

[Nag87] J. Nagle: *On Packet Switches with Infinite Storage*. IEEE Transactions on Communications, 35(4), 1987. (Cited on page 138.)

[NG09] J. C. Nobre and L. Z. Granville: *Consistency of States of Management Data in P2P-Based Autonomic Network Management*. In *IEEE DSOM '09: Proceedings of the International Workshop on Distributed Systems: Operations and Management*, volume 5841 of *Lecture Notes in Computer Science*, pages 99–110, 2009. (Cited on pages 66 and 164.)

[NS07] T. Nakano and T. Suda: *Applying Biological Principles to Designs of Network Services*. Elsevier Journal on Applied Soft Computing, 7(3):870–878, 2007. (Cited on page 166.)

[NZ02] T. S. E. Ng and H. Zhang: *Global Network Positioning: A New Approach to Network Distance Prediction*. ACM SIGCOMM Computer Communications Review, 32(1):61–61, 2002. (Cited on pages 73, 120, and 160.)

[Oct] Octoshape: *Octoshape - Infinite Edge*. http://www.octoshape.com. (Cited on page 143.)

[OLW99] T. J. Ott, T. V. Lakshman, and L. H. Wong: *SRED: Stabilized RED*. In *INFOCOM '99: Proceedings of the International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1346–1355, 1999. (Cited on page 139.)

[On05] G. On: *Quality of Availability for Widely Distributed and Replicated Content Stores*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, Multimedia Communications Lab (KOM), 2005. (Cited on page 15.)

[Ora] Oracle: *Oracle Database*. http://www.oracle.com. (Cited on page 4.)

[PB03] P. R. Pietzuch and J. Bacon: *Peer-to-Peer Overlay Broker Networks in an Event-based Middleware*. In *ACM DEBS'03: Proceedings of the International Workshop on Distributed Event-Based Systems*, 2003. (Cited on page 15.)

[PdRM+06] A. Panisson, D. M. da Rosa, C. Melchiors, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco: *Designing the Architecture of P2P-Based Network Management Systems*. In *IEEE ISCC '06: Proceedings of the IEEE Symposium on Computers and Communications*, pages 69–75, 2006. (Cited on page 164.)

[PGM+06] T. Plagemann, V. Goebel, A. Mauthe, L. Mathy, T. Turletti, and G. Urvoy-Keller: *From Content Distribution Networks to Content Networks - Issues and Challenges*. Journal on Computer Communications, 29(5):551–562, 2006. (Cited on page 15.)

[Pla] PlanetLab: *An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services*. www.planetlab.com. (Cited on page 72.)

[Pos80] J. Postel: *RFC 768: User Datagram Protocol*. http://tools.ietf.org/html/rfc768, 1980. (Cited on page 73.)

[PPP00] R. Pan, B. Prabhakar, and K. Psounis: *CHOKe - a stateless active queue management scheme for approximating fair bandwidth allocation*. In *INFOCOM '00: Proceedings of the International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 942–951, 2000. (Cited on page 139.)

[PR83] J. Postel and J. Reynolds: *RFC 854: Telnet Protocol Specification*. http://tools.ietf.org/html/rfc854, 1983. (Cited on page 12.)

[PR85] J. Postel and J. Reynolds: *RFC 959: File Transfer Protocol (FTP)*. http://tools.ietf.org/html/rfc959, 1985. (Cited on page 12.)

[PSVW01] D. E. Pendarakis, S. Shi, D. C. Verma, and M. Waldvogel: *ALMI: An Application Level Multicast Infrastructure*. In *USENIX USITS '01: USENIX Symposium on Internet Technologies and Systems*, pages 49–60, 2001. (Cited on page 16.)

[PVM+01] S. Pereira, S. Voloshynovskiy, M. Madueno, S. Marchand-Maillet, and T. Pun: *Second Generation Benchmarking and Application Oriented Evaluation*. In *Information Hiding*, volume 2137 of *Lecture Notes in Computer Science*, pages 340–353, 2001. (Cited on page 19.)

[RBL99] V. Rosolen, O. Bonaventure, and G. Leduc: *A RED Discard Strategy for ATM Networks and its Performance Evaluation with TCP/IP Traffic*. SIGCOMM Computer Communications Review, 29(3):23–43, 1999. (Cited on page 139.)

[RD01] A. I. T. Rowstron and P. Druschel: *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*. In *IFIP/ACM Middleware '01: Proceedings of the International Conference on Distributed Systems Platforms*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, 2001. (Cited on pages 14, 15, 32, 75, 154, and 175.)

[Rep09] N. Repp: *Überwachung und Steuerung dienstbasierter Architekturen - Verteilungsstrategien und deren Umsetzung*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2009. (Cited on pages 127 and 163.)

[RF08] A. Rizk and M. Fidler: *On the Identifiability of Link Service Curves from End-Host Measurements*. In *NET-COOP '08: Proceedings of the Workshop on Network Control and Optimization*, volume 5425 of *Lecture Notes in Computer Science*, pages 53–61, 2008. (Cited on page 56.)

[RGK+05] S. C. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu: *OpenDHT: A Public DHT Service and its Uses*. In *ACM SIGCOMM '05: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 73–84, 2005. (Cited on page 202.)

[Ric] Rice University, Houston, USA: *FreePastry*. http://www.freepastry.org/FreePastry/. (Cited on pages 4, 75, 174, 175, 176, and 186.)

[Rim05] I. Rimac: *End-to-End Mechanisms for Rate-Adaptive Multicast Streaming over the Internet*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2005. (Cited on page 139.)

[RKCD01] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel: *SCRIBE: The Design of a Large-Scale Event Notification Infrastructure*. In *Proceedings of Networked Group Communication, Third International COST264 Workshop*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43, 2001. (Cited on pages 4, 15, and 175.)

[RO03] R. Rejaie and A. Ortega: *PALS: Peer-to-Peer Adaptive Layered Streaming*. In *ACM NOSSDAV '03: Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 153–161, 2003. (Cited on page 139.)

[Rob99] D. Robinson: *The Advancement of NFS Benchmarking: SFS 2.0*. In *USENIX LISA'99: Proceedings of the Annual Conference on Systems Administration*, pages 175–186, 1999. (Cited on page 19.)

[RR04] A. Rajput and S. Rotenstreich: *Making a Case for Resource Management in a P2P Environment*. In *CSREA IKE '04: Proceedings of the International Conference on Information and Knowledge Engineering*, pages 475–484, 2004. (Cited on page 127.)

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21:120–126, 1978. (Cited on page 186.)

[RVK04] J. C. Rowanhill, P. E. Varner, and J. C. Knight: *Efficient Hierarchic Management For Reconfiguration of Networked Information Systems*. In *IEEE DSN '04: Proceedings of the International Conference on Dependable Systems and Networks*, pages 517–526, 2004. (Cited on page 163.)

[RW07] G. Rabinovitch and D. Wiese: *Non-linear optimization of performance functions for autonomic database performance tuning*. In *IEEE ICAS '07: Proceedings of the International Conference on Autonomic and Autonomous Systems*, page 48, 2007. (Cited on page 165.)

[SAL06] J. Strassner, N. Agoulmine, and E. Lehtihet: *FOCALE: A Novel Autonomic Networking Architecture*. In *LAACS '06: Proceedings of the Latin American Autonomic Computing Symposium*, 2006. (Cited on page 164.)

[SBN06] G. Surendra, S. Banerjee, and S. K. Nandy: *Instruction Reuse in SPEC, Media and Packet Processing Benchmarks: A Comparative Study of Power, Performance and related Microarchitectural Optimizations*. Journal on Embedded Computing, 2(1):15–34, 2006. (Cited on page 19.)

[Sea04] Sean Rhea and Dennis Geels and Timothy Roscoe and John Kubiatowicz: *Handling Churn in a DHT*. In *USENIX Annual Technical Conference*, 2004. (Cited on pages 75 and 175.)

[SENB07] M. Steiner, T. En-Najjary, and E. W. Biersack: *Analyzing Peer Behavior in KAD*. Technical Report EURECOM+2358, Institute Eurecom, 2007. (Cited on pages 73, 74, 92, 115, 120, and 128.)

[SGG02] S. Saroiu, P. K. Gummadi, and S. Gribble: *A Measurement Study of Peer-to-Peer File Sharing Systems*. In *SPIE MMCN '02: Proceedings of the Annual Multimedia Computing and Networking*, 2002. (Cited on page 14.)

[Sha] Sharman Networks: *KaZaA*. http://www.kazaa.com. (Cited on page 4.)

[Ska02] K. Skarhoj: *TYPO3*. http://www.typo3.org/, 2002. (Cited on page 4.)

[SKAB09] K. Sachs, S. Kounev, S. Appel, and A. P. Buchmann: *Benchmarking of Message-oriented Middleware*. In *ACM DEBS'09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, 2009. (Cited on page 19.)

[SKK03] J. Strauss, D. Katabi, and F. Kaashoek: *A Measurement Study of available Bandwidth Estimation Tools*. In *ACM IMC '03: Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, pages 39–44, 2003. (Cited on page 137.)

[Sky04] Skype: *Skype - Peer-to-Peer Internet Telephony*. http://www.skype.com, 2004. (Cited on pages 4 and 15.)

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan: *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*. In *SIGCOMM '01: Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160, 2001. (Cited on pages 9, 12, 13, 32, 69, 72, 154, 156, 157, and 160.)

[SMLN+03] 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 of Networking, 11(1):17–32, 2003. (Cited on page 73.)

[SN95] R. Steinmetz and K. Nahrstedt: *Multimedia: Computing, Communication and Applications*. Prentice-Hall, Inc., 1995. (Cited on page 172.)

[SN04] R. Steinmetz and K. Nahrstedt: *Multimedia Systems*. Springer, 2004. (Cited on page 172.)

[SP07] P. Shah and J.-F. Paris: *Peer-to-Peer Multimedia Streaming Using BitTorrent*. In *IEEE IPCCC'07: Proceedings of the International Performance, Computing, and Communications Conference*, pages 340–347, 2007. (Cited on page 146.)

[SSM+00] L. M. Silva, G. Soares, P. Martins, V. Batista, and L. Santos: *Comparing the Performance of Mobile Agent Systems: A Study of Benchmarking*. Computer Networks: The International Journal of Computer and Telecommunications Networking, 23(8):769–778, 2000. (Cited on page 19.)

[SSN+08] D. Shen, Y. Shao, T. Nie, Y. Kou, Z. Wang, and G. Yu: *HilbertChord: A P2P Framework for Service Resources Management*. In *Springer GPC '08: Proceedings of the International Conference on Advances in Grid and Pervasive Computing*, pages 331–342, 2008. (Cited on page 66.)

[SSR03] F. Schintke, T. Schütt, and A. Reinefeld: *A Framework for Self-Optimizing Grids Using P2P Components*. In *IEEE DEXA '03: Proceedings of the International Workshop on Database and Expert Systems Applications*, pages 689–693, 2003. (Cited on page 127.)

[SSZ98] I. Stoica, S. Shenker, and H. Zhang: *Core-stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*. SIGCOMM Compututer Communications Review, 28(4):118–130, 1998. (Cited on pages 138 and 139.)

[Ste00] R. Steinmetz: *Multimedia-Technologie: Grundlagen, Komponenten und Systeme*. Springer Verlag, 2000. (Cited on page 172.)

[Sti09] B. Stiller: *Telecommunication Economics - Overview of the Field, Recommendations, and Perspectives*. Computer Science - R&D, 23(1):35–43, 2009. (Cited on page 139.)

[Sun] Sun Microsystems: *MySQL Server*. http://www.mysql.com. (Cited on page 4.)

[SW97] J. Schmitt and L. Wolf: *Quality of Service - An Overview*. Technical Report TR-KOM-1997-01, Technische Universität Darmstadt, Multimedia Communications Lab, 1997. (Cited on pages 5 and 17.)

[SW05b] R. Steinmetz and K. Wehrle: *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, 2005. (Cited on page 11.)

[SWM97] M.-A. D. Storey, K. Wong, and H. A. Müller: *Rigi: A Visualization Environment for Reverse Engineering*. In *ACM ICSE '97: Proceedings of International Conference on Software Engineering*, pages 606–607, 1997. (Cited on page 165.)

[SZS02] I. Stoica, H. Zhang, and S. Shenker: *Self-Verifying CSFQ*. In *INFOCOM '02: Proceedings of the International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, 2002. (Cited on page 139.)

[TAA+04] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. christophe Hugly, E. Pouyoul, and B. Yeager: *Project JXTA 2.0 Super-Peer Virtual Network*. http://www.jxta.org, 2004. (Cited on pages 4, 66, 75, and 175.)

[TBF+03] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann: *A Peer-to-Peer Approach to Content-based Publish/Subscribe*. In *ACM DEBS'03: Proceedings of the International Workshop on Distributed Event-Based Systems*, 2003. (Cited on page 15.)

[TFNK08] H. Tamaki, K. Fukui, M. Numao, and S. Kurihara: *Pheromone Approach to the Adaptive Discovery of Sensor-Network Topology*. In *IEEE IAT '08: Proceedings of the International Conference on Intelligent Agent Technology*, pages 41–47, 2008. (Cited on page 166.)

[TJ09] N. Tölgyesi and M. Jelasity: *Adaptive peer sampling with newscast*. In *LNCS Euro-Par '09: Proceedings of the International Euro-Par Conference*, pages 523–534, 2009. (Cited on page 66.)

[TSS+06] K.-D. Thoben, M. Seifert, P. Sitek, M. Emde, and R. Tarditi: *Concept for Quality Control Management Services in Distributed Design Networks - Conceptual Paper*. In *Springer APMS '06: Proceedings of the Conference on Advances in Production Management Systems*, pages 461–471, 2006. (Cited on page 163.)

[UE08] M. Umlauft and W. Elmenreich: *QoS-Aware Ant Routing with Colored Pheromones in Wireless Mesh Networks*. In *Proceedings of the International Conference on Autonomic Computing and Communication Systemss*, page 31, 2008. (Cited on page 166.)

[UJJL05] P. Uppuluri, N. Jabisetti, U. Joshi, and Y. Lee: *P2P Grid: Service Oriented Framework for Distributed Resource Management*. In *IEEE SCC '05: Proceedings of the International Conference on Services Computing*, pages 347–350, 2005. (Cited on page 127.)

[UVGS04] A. Upadhaya, S. Vashishtha, R. Grover, and A. K. Sarje: *Network Management System Using Web Server Controlled Mobile Agents*. In *Springer IWDC '04: Proceedings of the International Workshop on Distributed Computing*, volume 3326 of *Lecture Notes in Computer Science*, page 540, 2004. (Cited on page 163.)

[vB04] R. van Renesse and A. Bozdog: *Willow: DHT, Aggregation, and Publish/Subscribe in one Protocol*. In *IPTPS '04: Proceedings of the International Workshop on Peer-To-Peer Systems*, volume 3279 of *Lecture Notes in Computer Science*, pages 173–183, 2004. (Cited on page 66.)

[vBH97] G. von Bochmann and A. Hafid: *Some Principles for Quality of Service Management*. Distributed Systems Engineering, 4(1):16–27, 1997. (Cited on page 65.)

[vDD+06] S. van der Meer, A. Davy, S. Davy, R. Carroll, B. Jennings, and J. Strassner: *Autonomic Networking: Prototype Implementation of the Policy Continuum*. In *IEEE BCN' 06: Proceedings of the International Workshop on Broadband Convergence Networks*, 2006. (Cited on page 165.)

[Ver] VeriSign: *Kontiki Delivery Management System*. http://www.kontiki.com. (Cited on page 143.)

[vRBV03] R. van Renesse, K. P. Birman, and W. Vogels: *Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining*. ACM Transactions on Compututer Systems, 21(2):164–206, 2003. (Cited on page 66.)

[WDÖ08] Q. Wang, K. Daudjee, and M. T. Özsu: *Popularity-Aware Prefetch in P2P Range Caching*. In *IEEE P2P '08: Proceedings of the International Conference on Peer-to-Peer Computing*, pages 53–62, 2008. (Cited on page 167.)

[Wik02] Wikimedia: *MediaWiki*. http://www.mediawiki.org/, 2002. (Cited on page 4.)

[WSZG08] T. Weise, H. Skubch, M. Zapf, and K. Geihs: *Global Optimization Algorithms and their Application to Distributed Systems*. Kasseler informatikschriften (kis) 2008, 3, Distributed Systems Group, FB 16, University of Kassel, 2008. (Cited on page 166.)

[WT97] J. Weinstock and R. Tewari: *An Object-Oriented Approach to the Management of Distributed Application Systems*. Computer Networks and ISDN Systems, 29(16):1869–1879, 1997. (Cited on page 163.)

[XYK+08] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz: *P4P: Provider Portal for Applications*. In *ACM SIGCOMM '08: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 351–362, 2008. (Cited on page 168.)

[YD04a] P. Yalagandula and M. Dahlin: *A Scalable Distributed Information Management System*. In *ACM SIGCOMM '04: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 379–390, 2004. (Cited on page 40.)

[YD04b] P. Yalagandula and M. Dahlin: *Research Challenges for a Scalable Distributed Information Management System*. Technical Report CS-TR-04-48, The University of Texas at Austin, Department of Computer Sciences, 2004. (Cited on page 67.)

[YGM03] B. Yang and H. Garcia-Molina: *PPay: Micropayments for Peer-to-Peer Systems*. In *ACM CCS '03: Proceedings of the ACM Conference on Computer and Communications Security*, pages 300–310, 2003. (Cited on page 16.)

[YLL+09] C. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee: *Decentralization: The Future of Online Social Networking*. In *W3C FSN '09: Proceedings of the W3C Workshop on the Future of Social Networking Position Papers*, 2009. (Cited on page 201.)

[You] YouTube - Broadcast Yourself. http://www.youtube.com. (Cited on page 3.)

[YP05] C.-C. Yeh and L. S. Pui: *On the Frame-Forwarding in Peer-to-Peer Multimedia Streaming*. In *ACM P2PMMS'05: Proceedings of the ACM Workshop on Advances in Peer-to-Peer Multimedia Streaming*, pages 1–10, 2005. (Cited on page 139.)

[Zat] Zattoo Europa AG: *Zattoo - TV to Go*. http://www.zatoo.com. (Cited on page 4.)

[Zin03] M. Zink: *Scalable Internet Video-on-Demand Systems*. PhD thesis, Multimedia Communications Lab (KOM), TU Darmstadt, 2003. (Cited on page 143.)

[ZJZ02] B. Zhang, S. Jamin, and L. Zhang: *Host Multicast: A Framework for Delivering Multicast To End Users*. In *INFOCOM '02: Proceedings of the International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, 2002. (Cited on page 16.)

[ZKJ02] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph: *Tapestry: A Fault-tolerant Wide-area Application Infrastructure*. Computer Communication Review, 32(1):81, 2002. (Cited on pages 14 and 168.)

[ZLL08] Y. Zhang, D. Li, L. C. 0002, and X. Lu: *Flexible Routing in Grouped DHTs*. In *IEEE P2P '08: Proceedings of the IEEE International Conference on Peer-to-Peer Computing*, pages 109–118, 2008. (Cited on page 167.)

[ZSZ03] Z. Zhang, S. Shi, and J. Zhu: *SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT*. In *IPTPS '03: Proceedings of the International Workshop on Peer-to-Peer Systems*, pages 170–182, 2003. (Cited on page 67.)

All web pages cited in this work have been checked in May 2010. However, due to the dynamic nature of the World Wide Web, web pages can change.

CURRICULUM VITÆ

PERSONAL DETAILS

Name: Kálmán György Graffi

Date and Place of Birth:
18th June 1982 in Târgu Mureș, Romania
Nationality: German

http://www.kalman-graffi.com

EDUCATION

June 2006 - present
Ph.D. student at the Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt.
Topic: Monitoring and Management of Peer-to-Peer Systems

April 2004 - May 2006
Studies of Mathematics, Technische Universität Darmstadt.
Degree: Diplom Mathematiker (Dipl.-Math.), M.Sc. equivalent
Grade: Very Good (1.5; with 1.0 being best)

October 2002 - May 2006
Studies of Computer Science, Technische Universität Darmstadt.
Degree: Diplom Informatiker (Dipl.-Inform.), M.Sc. equivalent
Grade: Very Good (1.5; with 1.0 being best)
Duration: Fastest out of 533 graduates (2004-2008)

AWARDS

Diploma Thesis
Awarded by Darmstadt's Competence Center for applied Security Technology (CAST) in the category "Best diploma/master's thesis" with the "CAST-Förderpreis für IT-Sicherheit für herausragende Nachwuchstalente" (CAST promotion prize for IT security for outstanding young talents), endowed with 2000 Euros

Certificate
Certificate by the Competence Center for Applied Security Technology "Cast": IT-Security Certificate. Affirming excellence in IT security. Grade: Very Good

TEACHING

Lectures and Exercises at Technische Universität Darmstadt
Lecture "Communication Networks II" (WS06/07, WS07/08, WS08/09, WS09/10): Lecture, exam and exercise organization, design and implementation, designated responsibility

Seminars, Lab Exercises at Technische Universität Darmstadt
Seminar "Advanced Topics of Future Internet Research" (WS06/07, WS07/08, WS08/09): Selected topics, supervised the students. Topics include: modeling of peer-to-peer systems, QoS-awareness in peer-to-peer systems

Project seminar "Advanced Topics in Distributed Systems" (SS07, SS08, SS09): Selected topics, implementation and evalutation in state-of-the-art P2P simulators

Bachelor lab exercise "Advanced Topics in Distributed Systems" (WS 08/09, WS 09/10): Design and implementation of GUI elements for a p2p-based online social network

Master lab exercise "Advanced Topics in Distributed Systems" (WS08/09, WS09/10, SS10): Engineering of collaboration and management tools for a p2p-based online social network

Seminar "P2P and Benchmarking" (SS10): Selected topics, supervised the students. Topics include: Self-aware networks, commercial aspects of p2p, monitoring and management of p2p systems

SUPERVISED DIPLOMA, M.SC., STUDENT, AND B.SC. THESES

Mu Mu, KOM-D-285, "User-Centric End-to-End QoS for Next Generation Network Services" (Master thesis)

Kyra Wulffert, KOM-D-294, "Emergency Call Handling in Peer-to-Peer and Next Generation Networks" (Master thesis)

Inna Kotchourova, KOM-D-299, "Evaluation and Extension of Analytical Models for P2P Systems" (Master thesis)

Song Xiao, KOM-D-306, "Management Architectures for Efficient Resource Allocation in Peer-to-Peer Systems" (Diploma thesis)

Daniel Brückmann, KOM-D-312, "Implementation and Evaluation of Distributed and Cooperative Authentication Mechanisms for Decentralized Systems" (Diploma thesis)

Sergey Podrajanski, KOM-D-316, "A Peer-to-Peer based Framework for Large-Scale Social Networking Platforms" (Diploma thesis)

Burkhard Menges, KOM-D-317, "Design and Implementaion of a Security Framework for Peer-to-Peer based Online Community Architectures" (Diploma thesis)

Thomas Wetter, KOM-D-331, "Technical and Economical Model for the Cooperation of Internet Service Providers and Peer-to-Peer Applications" (Diploma thesis)

Dominik Stingl, KOM-D-334, "Development of a Self-Optimizing Life Cycle Framework for Structured Peer-to-Peer Systems" (Diploma thesis)
Awarded as one of the best Diploma theses in 2009 by "Freunde der TU Darmstadt"-price, endowed with 1500 Euros

Christian Groß, KOM-D-351, "Live-Monitoring of Large-Scale Peer-to-Peer Systems" (Master thesis)

Awarded as one of the best Master theses in 2009 by "Freunde der TU Darmstadt"-price, endowed with 1500 Euros

Moustafa Zohdi, KOM-D-355, "Adaptation of Management and Leadership Strategies in Peer-to-Peer Networks" (Diploma thesis)

Vitaliy Rapp, KOM-D-358, "Monitoring Solutions for Structured and Unstructured Peer-to-Peer Overlays" (Diploma thesis)

Luciana Alvite, KOM-D-367, "Development of a Dynamic Flexible Monitored Peer-to-Peer based Social Network Platform using OSGi" (Diploma thesis)

Florian Gattung, KOM-D-370, "Benchmarking and Prototypical Implementation of a Social Knowledge Network" (Diploma thesis)

Ho Soo Kim, KOM-D-388, "Analytical Modeling of a Monitoring Approach for Peer-to-Peer Systems" (Diploma thesis)

Jonas Oppenländer, coadvised at Fachgebiet für Unternehmensgründung, "Marktanalyse und Marketingkonzept für eine P2P-basierte Online Community" (Diploma thesis)

Damian A. Czarny and Alexander Gebhardt, coadvised at Real Time Systems Lab, "Engineering of an automated Evaluation Platform for distributed Applications" (Bachelor theses)

Jun Chen, KOM-S-244, "Analysis and Evaluation of Mechanisms that Monitor the Resource Usage in Peer-to-Peer Systems" (Bachelor thesis)

Jiawei Du, KOM-S-245, "Analysis and Evaluation of Matching Algorithms for Efficient Resource Allocation in Peer-to-Peer Systems" (Bachelor thesis)

Andreas Straninger, KOM-S-252, "Evaluation of Mechanisms for a Self-Organizing Self-Conscious Intelligent Network" (Bachelor thesis)

Yevgen Fanshil, KOM-S-259, "Design and Implementation of a Monitoring Architecture for Peer-to-Peer based Systems (e.g. FreePastry)" (Bachelor thesis)

Rubén Fraile Blázquez, KOM-S-264, "Literature Survey on Information Management Architectures for Peer-to-Peer Systems" (Bachelor thesis)

Ling Wang, KOM-S-268, "Taxonomy on Context Information in Peer-to-Peer Systems and their Measurement Methodology" (Bachelor thesis)

Yue Sheng, KOM-S-269, "Implementation of Chord and the Key-Based Routing Interface in PeerfactSim.KOM" (Bachelor thesis)

Luciana Alvite, KOM-S-279, "Design and Implementation of a Service Oriented, Dynamic and Modular P2P-based Multimedia Online Community Platform using OSGi" (Student thesis)

Moritz Bornwasser, KOM-S-281, "Design and Implementation of an extendible User Interface for LifeSocial.KOM" (Bachelor thesis)

Julius Rückert, KOM-S-0295, "Details and Principles in Monitoring and Systematically Improving of Large-Scale Structured Peer-to-Peer Systems" (Bachelor thesis)
Awarded with the "Datenlotsen AG"-price for best Bachelor thesis in 2009, endowed with 2500 Euros

Carsten Snider, KOM-S-0296, "Mobility Aware Peer-to-Peer Networking" (Bachelor thesis)

Christoph Neumann, KOM-S-0303, "Survey and Prototypical Implementation of Semantical Networks using Peer-to-Peer Technology" (Bachelor thesis)

Michael Behrisch, KOM-S-0318, "Literaturey Survey on Autonomic Computing in Peer-to-Peer Systems" (Bachelor thesis)

Hoang Minh Nguyen, KOM-S-0319, "Longterm Service Level Agreements in unreliable Peer-to-Peer Systems" (Bachelor thesis)

Lyudmil Vasilev, KOM-S-0328, "Improviding the Quality of Measurements through Filtering and Calibration" (Bachelor thesis)

Johannes Beutel, KOM-S-0361, "Machine Learning in Peer-to-Peer Systems" (Bachelor thesis)

Stephan Moczygemba, KOM-S-367, "Security and Privacy in Peer-to-Peer-based Online Social Networks" (Bachelor thesis)

August 10, 2010

PUBLICATIONS

PATENTS AND INVENTION REPORTS

[1] Kalman Graffi, Parag S. Mogre, Matthias Hollick, and Christian Schwingenschlögl. Detection of Colluding Misbehaving Nodes in Mobile Ad-Hoc and Wireless Mesh Networks, European Patent Office, Application No. /Patent No. 07022624.6 – 1249, 2007.

[2] Parag S. Mogre, Kalman Graffi, Matthias Hollick, and Andreas Ziller. Misbehaviour Detection in Wireless Mesh Networks without promiscuous Overhearing, European Patent Office, Application No. /Patent No. 07017485, 2007.

JOURNAL ARTICLES

[1] Kalman Graffi, Aleksandra Kovacevic, Patrick Mukherjee, Michael Benz, Christof Leng, Dirk Bradler, Julian Schröder-Bernhardi, and Nicolas Liebau. Peer-to-Peer-Forschung - überblick und Herausforderungen. *it - Information Technology (Methods and Applications of Informatics and Information Technology)*, 46(3), 2007.

[2] Parag Mogre, Kalman Graffi, Matthias Hollick, and Ralf Steinmetz. A Security Framework for Wireless Mesh Networks. *Wireless Communications and Mobile Computing, Special Issue on "Architectures and Protocols for Wireless Mesh, Ad Hoc, and Sensor Networks"*, 2010. to appear.

CONFERENCE AND WORKSHOP CONTRIBUTIONS

[1] Kalman Graffi, Christian Groß, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz. LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks. In *IEEE P2P '10: Proceedings of the International Conference on Peer-to-Peer Computing*, 2010.

[2] Kalman Graffi, Dominik Stingl, Julius Rückert, and Aleksandra Kovacevic. Monitoring and Management of Structured Peer-to-Peer Systems. In *IEEE P2P '09: Proceedings of the International Conference on Peer-to-Peer Computing*, 2009.

[3] Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. Practical Security in P2P-based Social Networks. In *IEEE LCN '09: Proceedings of the International Conference on Local Computer Networks*, 2009.

[4] Kalman Graffi, Aleksandra Kovacevic, Song Xiao, and Ralf Steinmetz. SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems. In *IEEE ICPADS '08: Proceedings of the International Conference on Parallel and Distributed Systems*, 2008.

[5] Kalman Graffi, Sergey Podrajanski, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz. A Distributed Platform for Multimedia Communities. In *IEEE ISM '08: Proceedings of the International Symposium on Multimedia*, 2008.

[6] Kalman Graffi, Sebastian Kaune, Konstantin Pussep, Aleksandra Kovacevic, and Ralf Steinmetz. Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems. In *ACM NOSSDAV '08: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2008.

[7] Kalman Graffi, Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. From Cells to Organisms: Long-Term Guarantees on Service Provisioning in Peer-to-Peer Networks. In *ACM SIGAPP NOTERE '08: Proceedings of the International Conference on New Technologies of Distributed Systems*, 2008.

[8] Kalman Graffi, Parag Mogre, Matthias Hollick, and Ralf Steinmetz. Detection of Colluding Misbehaving Nodes in Mobile Ad Hoc and Mesh Networks. In *IEEE GLOBECOM 07: Proceedings of the Global Communications Conference, Exhibition & Industry Forum*, 2007.

[9] Kalman Graffi, Konstantin Pussep, Sebastian Kaune, Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Overlay Bandwidth Management: Scheduling and Active Queue Management of Overlay Flows. In *IEEE LCN '07: Proceedings of the International Conference on Local Computer Networks*, 2007.

[10] Kalman Graffi, Aleksandra Kovacevic, Kyra Wulffert, and Ralf Steinmetz. ECHoP2P: Emergency Call Handling over Peer-to-Peer Overlays. In *IEEE P2PNVE'07: Proceedings of the International Workshop on Peer-to-Peer Network Virtual Environments*, 2007.

[11] Osama Abboud, Aleksandra Kovacevic, Kalman Graffi, Konstantin Pussep, and Ralf Steinmetz. Underlay Awareness in P2P Systems: Techniques and Challenges. In *IEEE IPDPS '09: Proceedings of the International Symposium on Parallel and Distributed Processing*, 2009.

[12] Aleksandra Kovacevic, Kalman Graffi, Sebastian Kaune, Christof Leng, and Ralf Steinmetz. Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments. In *IEEE ICPADS '08: Proceedings of the International Conference on Parallel and Distributed Systems*, 2008.

[13] Nicolas Liebau, Konstantin Pussep, Kalman Graffi, Sebastian Kaune, Eric Jahn, André Beyer, and Ralf Steinmetz. The Impact Of The P2P Paradigm. In *AMCIS '07: Proceedings of Americas Conference on Information Systems*, 2007.

[14] Parag Mogre, Kalman Graffi, Matthias Hollick, and Ralf Steinmetz. AntSec, WatchAnt and AntRep:Innovative Security Mechanisms for Wireless Mesh Networks. In *IEEE LCN '07: Proceedings of the International Conference on Local Computer Networks*, 2007.

TECHNICAL REPORTS

[1] Kalman Graffi, Aleksandra Kovacevic, and Ralf Steinmetz. Towards an Information and Efficiency Management Architecture for Peer-to-Peer Systems based on Structured Overlays. Technical Report KOM-TR-2008-2, Multimedia Communications Lab KOM, Technische Universität Darmstadt, Germany, 2008. ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2008-02.pdf.

[2] Kalman Graffi, Konstantin Pussep, Nicolas Liebau, and Ralf Steinmetz. Taxonomy of Active Queue Management Strategies in Context of Peer-to-Peer Scenarios. Technical Report Tr-2007-01, Technische Universität Darmstadt, Germany, 2007. ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2007-01.pdf.

[3] Kalman Graffi, Nicolas Liebau, and Ralf Steinmetz. Taxonomy of Message Scheduling Strategies in Context of Peer-to-Peer Scenarios. Technical Report Tr-2007-02, Multimedia Communications Lab KOM, Technische Universität Darmstadt, Germany, 2007. ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2007-02.pdf.

[4] Kalman Graffi. A Security Framework for Organic Mesh Networks. Master's thesis, Technische Universität Darmstadt, Germany, 2006.

DECLARATION

Erklärung laut §9 PromO

Ich versichere hiermit, dass ich die vorliegende Dissertation allein und nur unter Verwendung der angegebenen Literatur verfasst habe. Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Datum

Dipl.-Math., Dipl.-Inform.
Kálmán György Graffi

LIST OF FIGURES

LIST OF TABLES