

Systematic Benchmarking of a Fully Distributed Communication System

Master Thesis

by

Dlikman, Iakov Arkadjevic

born in
St.-Petersburg, Russia

submitted to

Technology of Social Networks Lab
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich-Heine-Universität Düsseldorf

Juni 2018

Supervisor:
Jun.-Prof. Dr.-Ing. Kalman Graffi
M. Sc. Newton Masinde

Abstract

This thesis focuses on the evaluation and benchmarking of a peer-to-peer (p2p) based framework. The p2p-framework builds on top of the structured p2p-overlay Pastry. It offers a platform for distributed applications. These can be constructed using the modular OSGi environment and communicate with the p2p-framework through a well-defined application programming interface (api). The currently available application is LibreSocial, a distributed, online social network.

Prior to this thesis, several works focused on benchmarking. This thesis introduces a systematic categorization of the p2p-framework into the field of distributed systems in general and p2p-systems in specific. It helps to choose an emphasis of the analysis for this thesis and motivates further aspects of analysis.

In addition, this thesis implements tooling for the deployment of the p2p-framework and its current application in a local area environment. It shows how to setup and conduct experiments focusing on the defined properties. finally, it gives tools and instructions how to extract and present results in a tabular and graphic format for use in publications.

The core part of the thesis evaluates the p2p-framework through the LibreSocial application and evaluates measurement results focusing on system availability, scaling, and robustness. For the evaluation, we use a distributed monitoring protocol - SkyEye.KOM - to gather statistics about system performance. We test the p2p-framework in a local area network environment and present two options how this environment can be further extended.

The benchmarking results show that availability of the system remains high between 92.5% and 100% during light and intermediate system load. Scalability benchmarking shows that the p2p-framework scales up in case of exponential population growth, maintaining overall system performance between 71% and 84%. Robustness testing reveals that 16% of the carrying nodes can fail without affecting overall system performance while 33% of node failure causes the system availability to drop significantly.

Acknowledgments

The past three years have been an adventure full of ups and downs. This part of the journey now comes to an end with this master thesis. I would like to take the chance and express my gratitude to the people that have been with me during this time. Thanks to prof. Kern and prof. Brown for their effort in the field of mathematics. Thanks to Jens Bendisposto and Philipp Koerner for introducing me to the magic of functional programming and thanks to my colleagues Felix, Oktay, Dorian, and Marcin for helping me "‘understand"’ the concept of monads. I am grateful for prof. Rothe and the monkey, my colleagues Basel and Hubert for helping me find my Turing machine that I left in the parking lot. Thanks to prof. Lercher and Jannik for their effort during the bioinformatics courses. Thanks to prof. Wanke, Duygu, Vitali, Saeed, Julian, Maike, Thau, and "‘the guy in the suit"’ for the challenging and exciting courses on theoretical computer science. Thanks to prof. Verde, Dominique, Tobias, and Tom for the course on computational statistics where I felt like being at M.I.T. for two weeks. Big Thanks to prof. Graffi, Artur, Achmed, Hamed, Andre, Tobias, Andreas, and Raphael for the course on p2p-systems and mobile communication. Thanks to prof. Schoettner for teaching me the foundations of operating systems and the foundations of perseverance and patience. I would like to say thanks to prof. Leuschel and Dominik, to Nicholas and to Lightbot, I hope I got all invariants right. A big thanks to Jochen, Malte, Dietmar, Michael, Markus, Irina and Aline for the time-outs at the time-out cafe. Thanks to the Fellowship of the Spirit for teaching me the most important lesson of all. Last but not least, thanks to my parents Arkadiy and Alexandra for the opportunities they opened up for me and to my half-sis Katja for saving my life. Thanks to all who helped me the last two weeks, to Felix, Moritz, Judith, Patrick, and Uli for their very practical support and the right words at the right times. A big thanks to my two mentors during the last 12 month, Mr. Raed Al-Aaridhi for introducing me to the scientific way of working and being patient and supportive to me and Mr. Newton Masinde for the valuable support, feedback, and direction on the last meters to the finish line.

Contents

List of Figures xi

List of Tables xiii

1. Introduction 1
- 1.1. Motivation . . . 1
- 1.2. Contribution . . . 2
- 1.3. Organization of the thesis . . . 3

I. Systematic Approach 5

2. Categorization of System to be examined 7
- 2.1. Distributed System . . . 7
  - 2.1.1. Definition of a Distributed System . . . 7
  - 2.1.2. Evolution of Distributed and P2P-Systems . . . 8
  - 2.1.3. Goals . . . 9
  - 2.1.4. Key Concepts . . . 10
  - 2.1.5. Motivation for Usage of Distributed and P2P-Systems . . . 12
- 2.2. Peer-to-Peer Systems . . . 13
  - 2.2.1. Properties of P2P-Systems . . . 14
- 2.3. Peer-to-Peer-Framework and LibreSocial . . . 16
  - 2.3.1. Internet Layer . . . 16
  - 2.3.2. P2P-Overlay Layer . . . 16
  - 2.3.3. P2P-Framework Layer . . . 17
  - 2.3.4. Application Layer . . . 18
- 2.4. Summary . . . 19

3. Definition of Metrics and Means for Evaluation 21
- 3.1. Metrics for Evaluation of Peer-to-Peer Systems . . . 21
- 3.2. Monitoring the P2P Framework . . . 23
  - 3.2.1. Gossip-based Monitoring . . . 24

3.2.2. Tree-based Monitoring . . . 25
3.2.3. SkyEye.KOM . . . 26

II. Benchmarking and Evaluation **27**

4. Deployment, Preparation and Execution **29**
4.1. Preparing the PC-Lab . . . 29
4.1.1. Compiling the source code . . . 29
4.1.2. Deployment . . . 31
4.2. Running Experiments . . . 34
4.2.1. LibreSocial Parameters . . . 34
4.2.2. Tester Plugin Configuration . . . 35
4.2.3. Physical Setup . . . 35
4.2.4. Scenarios . . . 36
4.3. Gathering Results . . . 37
4.3.1. Data Preparation . . . 37
4.3.2. Tables and Diagrams . . . 38

5. Evaluation of Benchmarking Measurements **43**
5.1. Regression Test . . . 43
5.1.1. Regression Test Scenario . . . 44
5.1.2. Metrics for Messages, Storage and Time . . . 44
5.1.3. Metrics for Plugins . . . 50
5.1.4. Mapping of Plugins, Metrics and Test Cases . . . 55
5.1.5. Plugin Analysis and Categorization . . . 57
5.1.6. Evaluation Summary . . . 58
5.2. Availability . . . 59
5.2.1. Availability Metric . . . 59
5.2.2. Measurement Results . . . 61
5.3. Scalability . . . 64
5.3.1. Scalability Metric . . . 64
5.3.2. Population Growth . . . 65
5.3.3. Scenario for Scalability . . . 65
5.3.4. Measurement Results . . . 67
5.4. Robustness . . . 69
5.4.1. Robustness Metric . . . 69
5.4.2. Peer Dynamics . . . 70
5.4.3. Scenario of Robustness . . . 70
5.4.4. Measurement Results . . . 71

III. Summary and Outlook **75**

6. Measurement Summary **77**

6.1. Summary and Conclusion 77
6.1.1. Availability 78
6.1.2. Scalability 79
6.1.3. Robustness 79

7. Future Benchmarking **81**

7.1. High Performance Computing 81
7.2. Small and Large Scale Internet Setups 82

IV. Appendix **85**

Appendix A - Detailed Metric Overview for Availability Benchmarking **87**

7.3. Message Dispatcher 89
7.3.1. Sent Messages 89
7.3.2. Received Messages 91
7.4. Storage Dispatcher 92
7.4.1. Stored Data 92
7.4.2. Retrieved Data 92
7.4.3. Stored and Retrieved Data Rate 94
7.4.4. Stored DDS Data 95
7.4.5. Retrieved DDS Data 95
7.5. Null Pointer and Other Exceptions 97
7.6. Plugins with high Messaging Impact 98
7.6.1. Message Plugin 98
7.6.2. Forum Plugin 103
7.7. Plugins with high Storage Impact 108
7.7.1. Photo Plugin 108
7.8. Plugins with Mixed Impact 112
7.8.1. Vote Plugin 112
7.8.2. Filestorage Plugin 116

Bibliography **119**

List of Figures

2.1. Distributed Systems Overview, based on [12] 8
2.2. LibreSocial and P2P-Framework Architecture, based on [22] 20

3.1. Gossip-Based Monitoring 24
3.2. Tree-Based Monitoring 25

4.1. Launch LibreSocial within Eclipse 30
4.2. LibreSocial from User's Perspective 31
4.3. Export Wizard for LibreSocial Compilation 32

5.1. Number of Nodes 44
5.2. Join Network Time vs. Registration Time 45
5.3. Average Lookup Time 45
5.4. Message-Dispatcher - Sent vs. Received Messages 46
5.5. Message-Dispatcher - Size of Sent vs. Received Messages 47
5.6. Network - Sent vs. Received Messages 47
5.7. Storage-Dispatcher - Stored vs. Retrieved Data 48
5.8. Storage-Dispatcher - Used Size 49
5.9. Framework - Sent vs. Received Data 49
5.10. Missed Calculactions 50
5.11. Wall Entries and Views 51
5.12. Outbox - number of Views 51
5.13. Number of executed Searches 52
5.14. Number of Calendar Views 52
5.15. Voting - number of Deletions 53
5.16. Photos - number of Uploads 54
5.17. Files - number of Uploads 54
5.18. Saturation of uploaded Files and Photos 63
5.19. Saturation of Sent Messages and Created Posts 63
5.20. Population Growth 65
5.21. Messaging Scalability 68

5.22. Storage Scalability 69
5.23. Light (LE) and Medium (ME) Churn 72
5.24. Messaging and Storage during Light (LE) and Medium (ME) Churn 72

6.1. Data-Rates for Availability 79
6.2. Data-Rates for Scalability 80
6.3. Data-Rates for Robustness 80

7.1. MessageDispatcher - Sent Messages (Count, Rate) 89
7.2. MessageDispatcher - Received Messages (Count, Rate) 91
7.3. StorageDispatcher - Stored Data 92
7.4. StorageDispatcher - Retrieved Data 92
7.5. StorageDispatcher - Stored and Retrieved Data Rate 94
7.6. StorageDispatcher - Stored DDS Data 95
7.7. StorageDispatcher - Retrieved DDS Data 95
7.8. StorageDispatcher - Stored and Retrieved DDS Data Rate 96
7.9. Null Pointer Exceptions vs. Other Exceptions 97
7.10. MESSAGING plugin - Sent Messages 98
7.11. MESSAGING plugin - Inbox and Outbox Views 99
7.12. MESSAGING plugin - Data Stored Amount and Rate 100
7.13. MESSAGING plugin - Data Retrieved Amount and Rate 101
7.14. FORUM plugin - Number of Created Forum Threads and Posts 103
7.15. FORUM plugin - Number of Forum Views 104
7.16. FFORUM plugin - Data Stored Amount and Rate 105
7.17. FFORUM plugin - Data Retrieved Amount and Rate 106
7.18. PHOTO plugin - Created Album and Views 108
7.19. PHOTO plugin - Data Stored Amount and Rate 109
7.20. PHOTO plugin - Data Retrieved Amount and Rate 110
7.21. VOTE plugin - Created and Deleted Votes 112
7.22. VOTE plugin - Retrieved Voting Results and Voting List Views 113
7.23. VOTE plugin - Data Stored Amount and Rate 114
7.24. VOTE plugin - Data Retrieved Amount and Rate 115
7.25. FILESTORAGE plugin - Stored Files 116
7.26. FILESTORAGE plugin - Data Stored Amount and Rate 117
7.27. FILESTORAGE plugin - Data Retrieved Amount and Rate 118

List of Tables

3.1. Metric Summary and Calculation 22

4.1. LibreSocial Parameters 34
4.2. Test Plugin Parameters 34

5.1. Plugins, Metrics and Test Cases 55
5.2. Storage-Dispatcher Table 57
5.3. Message-Dispatcher Table 57
5.4. Storage Availability Results 61
5.5. Messaging Availability Results 62
5.6. Linear Population Growth 67
5.7. Exponential Population Growth 67
5.8. Light and Medium Churn 71

6.1. Performance of Messaging and Storage on P2P-Framework Layer 77

Chapter 1.

Introduction

1.1. Motivation

Peer-to-Peer Systems (P2P) offer an efficient, alternative way to use the internet's resources and infrastructure. While client-server architectures are predominant in modern web applications such as social networks, marketplaces, knowledge sharing etc. this type of architecture also introduces a number of problems that cannot be solved within the client-server mindset. Typical problems occur and sometimes become visible to the general public when the central instance in such applications gets attacked, is temporarily not available or is used in a way that is not desirable.

Following are a few examples of everyday technological systems based on the centralized architecture. The providers of Facebook, due to centralized storage of user information and profiles can and do easily distribute and sell this data to data analysis agencies [1]. The public transportation system is organized in a client-server way. The digital schedule panels at the train and bus stops all retrieve information from a central database. This way, temporary unavailability of the database leads to unavailability of all panels in town. Furthermore, to communicate current traffic information, the busses and trains first contact the dispatcher who broadcasts the information to all busses and trains, most of which are not affected by the information. An additional example is messaging services for mobile devices that transmit messages to the closest radio receiver station, which then transmits the data to a server. This server sends the message back to a radio receiver station that sends the data to the recipient. This is especially resource consuming when both, sender and receiver's geographical location is very close compared to the distance to the server instance.

P2P-Systems address these and more use cases. Each participant in such a system takes on the role of client and server at the same time. For messaging services, each sender can for a moment become a server and directly provide data to the recipient or act as an intermediate node and route incoming packages to a node closer to the recipient. In public transportation, buses and trains can use existing

radio equipment or be equipped with WiFi technology, so that traffic information is spread locally to passing buses and trains, that again spread this information to passing buses and trains. Web-based applications also work on P2P-basis. Skype [2] is a successful voice-over-IP service, that based on P2P-technology in its origins and became very popular. Microsoft bought its licenses and the technology, though fully functional, changed back to the centralized architecture.

Furthermore, the p2p-framework presented and benchmarked in this thesis offers a platform for the development of a wide range of internet applications. Its core was laid out and implemented by Graffi [3] in 2009. It further developed through project groups during 2011, 2012 and 2013 [4] [5] [6]. Since then, a number of master theses focused on single aspects of the system, such as security, data structures and graphical presentation. Of course, a number of benchmarking theses exists [7] [8], of which this is the most recent as of June 2018.

The p2p-framework is a p2p-based application platform. Its proof of concept is the implementation of an online social network LibreSocial [9] [10]. At the same time, it is not limited to this type of application. This thesis shows benchmarking results of the p2p-framework in a local area environment. One could think of use cases in schools, offices or university campus, where knowledge sharing systems, collaborative tools, messaging and high-performance computing can be solved on a p2p basis without a dedicated additional server. This can save costs or let the budget flow in projects and research, that can not be realized, at the same time using the existing resources in a more efficient way. The back side of the coin is, that such a system requires an open mindset for solving computational problems in a distributed manner. Centralized algorithm not only differ from distributed algorithms. There are also computational problems for which either a centralized or a distributed solution exists, but not both at the same time. In addition, the implementation and deployment of p2p-based systems require a group of specialists with an advanced understanding of distributed systems.

1.2. Contribution

The contribution of the following thesis is threefold. First of all, it presents up-to-date results on the performance of the p2p-framework, focusing on different aspects of the system. Second, it provides hands-on tools and instructions for system deployment, conduction of experiments and data analysis. This way it is easy to continue the benchmarking or focus on different aspects, easily generate tables and graphs for reports or publication. Third, the p2p-framework is categorized. This way, it is possible to refer to a list of common characteristics and goals of a p2p-system when looking for further aspects of benchmarking. E.g. scalability, availability and robustness are the main focus for benchmarking in the context of this thesis. More general aspects, such as load distribution, security are part of the field of distributed systems as well as p2p-systems. They may lead to further new benchmarking setups.

1.3. Organization of the thesis

The organization of the thesis follows the structure of the contribution. Part I focuses on the categorization of the system. In chapter 2 we present an overview of key concepts and goals for distributed systems. These can be used for further planning and analysis of benchmarking. Chapter 3 sets the focus of analysis for this thesis and gives an overview of metric calculation and means of gathering the metrics - a distributed monitoring protocol.

Part II consists of hands-on instructions how to deploy, setup and evaluate experiments in chapter 4. In addition, chapter 5 gives a detailed, up-to-date evaluation of the p2p-framework focused on the aspects we defined in part I. This is the core of the thesis.

Part III summarizes observations and gives further details on overall p2p-framework performance in chapter 6. Chapter 6 7 motivates further benchmarking in a High-Performance Computing environment or internet setup.

The Appendix IV contains condensed figures and tables of one of the benchmarking results. It shows one result set with extended details offering a template for subsequent evaluations.

Part I.

Systematic Approach

Chapter 2.

Categorization of System to be examined

2.1. Distributed System

This chapter introduces distributed systems. First of all, it gives a definition of the term distributed systems for the scope of this thesis. Next, it gives a very short overview of the historical development of distributed systems in general and the emergence of peer-to-peer-systems (p2p) in specific. It lists technological advances that led to distributed systems. The chapter continues with a discussion of distributed systems goals as well as the key concepts of distributed systems to reach these goals. Distributed systems build the foundation for the discussion of p2p-systems in chapter 2.2. We use these properties distributed systems have in common to show how the p2p-framework acquires these properties in later chapters. The last part of this chapter gives motivation for the choice of distributed and p2p-architectures instead of the classical client-server architecture.

2.1.1. Definition of a Distributed System

A distributed system can be defined as *"any system in which a number of independent interconnected computers can cooperate in order to achieve a common goal"* [11]. Nonetheless, it contains key elements for the further discussion. First of all, there is a number of interconnected computers. The interconnection may be a local area network (LAN), a wide area network (WAN) or a combination of several networks, such as the internet. The definition further mentions the need for cooperation. Cooperation can mean a communication protocol that enables the computers to coordinate tasks. This communication reaches from low-level communication over the physical layers of the network to the application layer. Last, the definition mentions a common goal. As we will see in subsequent sections, this common goal may be the computation of a result that needs combined computational power, or, as in case of middle-ware to provide an application platform, hiding the heterogeneity of the network to its users.

2.1.2. Evolution of Distributed and P2P-Systems

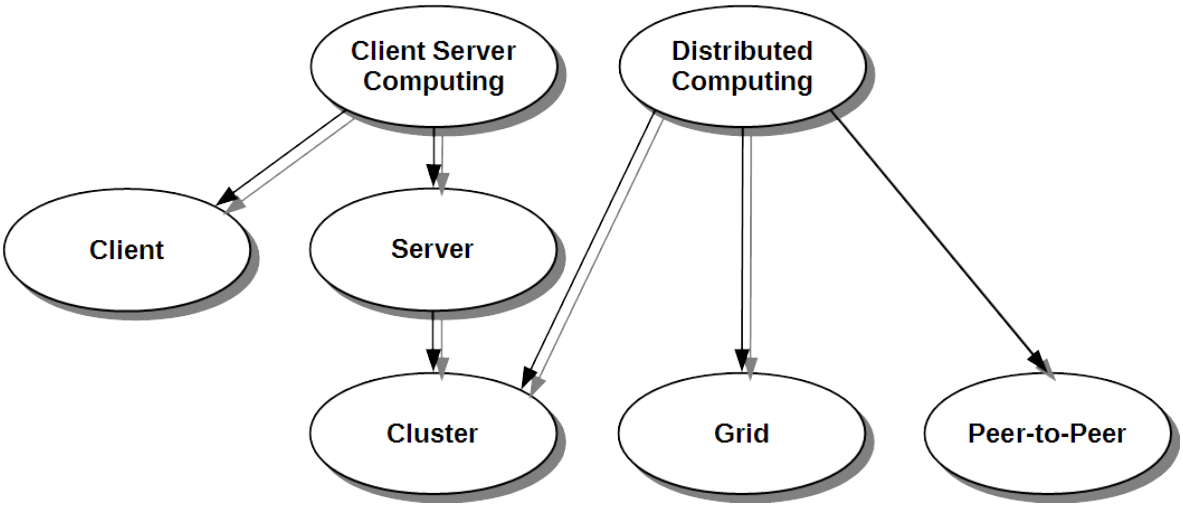


Figure 2.1.: Distributed Systems Overview, based on [12]

Figure 2.1 shows how p2p-systems emerged from distributed systems, what other types of distributed system exist alongside p2p-systems. In addition, we see the most common centralized solution as opposed to distributed systems, the client-server systems. Baun et al. [12] describe the historical evolution of distributed systems and p2p-systems as follows:

In 1985 first client-server systems emerge. servers offer services such as printing or file storage, whereas clients access these services e.g. with a terminal. An operating system that provides services on the server side and enables users to access and use these services on the client side is called *network operating system*.

In 1990 cluster systems come up. All components here are homogenous, i.e. processor type or networking. Provided services are the same across all physical machines. They add additional physical servers to existing ones to make them more reliable in cases of failure. Furthermore, a new operating system type that hides the actual number of physical machines and their interconnection and provides a single system view to the user comes up - the *distributed operating system*.

In 1995 first peer-to-peer (p2p) systems develop. In contrast to client-server architectures every machine acts in both roles, i.e. provides and accesses services to and from other participants. this architecture becomes known through Napster [13], an MP3 file exchange service. The p2p architecture further matures through applications such as the original Skype [2] for voice-over-ip telephony or BitTorrent [14] for distributed file sharing. More about p2p-systems follows in chapter 2.2

In 2000 grid-systems come up. Different to cluster systems they connect heterogeneous machines and present to the user a selection of different services. The means of choice for this use case is

middleware, an abstraction layer that hides heterogeneity and presents a clear interface to the user. The concept of middleware is crucial to p2p-overlays such as Pastry [15], which is the basis for the p2p-framework discussed in this thesis.

At the same time, another area becomes popular where p2p-technology plays an important role. Wireless and mobile computing allow portable devices such as notebooks, PDAs or mobile phones to communicate without infrastructure. As they do not rely on a central instance to coordinate communication, this type of systems also follows the p2p architecture. Recent research in this area using Android devices is done by Ippisch et al. [16].

In addition to the historical evolution, we sum up technological development that led to both - distributed and p2p-systems. These are occurrence of high-performance microprocessor, development of local area networks, development of wireless and ad-hoc networks, and emergence of the internet as the interconnection of local area networks. The next paragraph introduces the common properties of distributed systems that also define properties of a p2p-system.

2.1.3. Goals

Tanenbaum and Maarten define the following goals for a distributed system [17].

Connect users and resources

A distributed system should easily connect users with resources. It should hide the fact, that resources are distributed across the network – the distributed system should be transparent.

Transparency

Transparency in this context means that the systems hide information from the user. E.g. transparency of replication hides that files may be replicated to maintain consistency. Transparency of concurrency means the systems hide the fact that multiple users may access the same data simultaneously. Transparency of location and relocation hides the location of the data from the user. The distributed systems provide this type of transparency by means of naming. Transparency of failure hides unavailability of services from the user. Sometimes the authors recommend giving the user more information instead of hiding information.

Openness

Another goal of a distributed system is to be open. Openness can refer to extendability and interoperability in the sense that interfaces between components of the system are well defined. This enables exchange and extension of components of the system. Further, openness can refer to portability in the sense that interface of the distributed system to the underlying operating system or underlay is well defined. This enables the system to be portable to different underlay systems that provide the application programming interface (API).

Scalability

The fourth goal for a distributed system is to scale. Scaling refers to the size, geographic location, and administrative workload. Challenges regarding scalability arise from centralized systems. When more users or computers request data or services from a centralized unit, the unit becomes a bottleneck. Further, centralized algorithms, e.g. routing algorithms, require a complete overview of the underlying network which leads to a huge network load. Therefore the bottleneck and centralized network overview lead to the introduction of distributed systems that solve the bottleneck and distributed algorithms that require only a local knowledge of the network. Another solution is to split large components into smaller and distribute these parts across the network. Geographic scaling means the following. Systems developed for narrow geographic location, e.g. LAN rely on synchronous communication. As soon as the system scales to WAN with unsecured communication components block until response returns. To ensure scalability in widely spread distributed systems the solution is to use asynchronous communication and multi-threaded systems. Scaling in terms of administrative workload means to replicate popular services and data and make them available for a larger amount of requests.

2.1.4. Key Concepts

Schill and Springer [18] point out six goals such a system needs to accomplish.

Cooperation is possible across spatial boundaries

This means the location of the computers that contribute to the system may vary from an intranet in an office building to personal computers across the globe connected through their respective internet service providers (ISPs). As we will see in the following chapter on p2p-systems this involves

reachability of computers behind network address translation (NAT).

Sharing Resources

One of the main aspects of any distributed system is that any computer of the distributed systems has in a predefined manner access to resources of other computers of the system. Compared to a classical client-server distribution where the entire resources of the system are located at the central server (or a limited number of interconnected servers) in a distributed systems each new computer adds to the overall performance of the system by sharing its resources.

Naming

In a distributed system we require the resources such as memory or computing power to be available. Therefore we need a way to address these resources. This is done by *naming*. An example may be the IP address of a physical machine, a file name or URL to locate a resource on the web. In Pastry, a peer-to-peer overlay the framework uses, nodes and data have unique identifiers. This way both have a location that messages can route to. A challenge regarding Naming is how to locate mobile entities. Through NAT and firewall reachability of nodes may change or be difficult to establish.

Synchronization

In a distributed system we want to establish a consistent state of the overall system. We can accomplish this with logical or physical clocks. To avoid drift Pastry uses a logical clock in addition to the hardware clock of each node. Through the means of tree-based monitoring SkyEye.KOM in a fully distributed manner gives each user insight into the global configuration of the network. Specific metrics such as bandwidth consumption or storage size of the entire system is available to each node.

Consistency and Replication

How can we ensure that data stored on a node that is currently not available is still available to other nodes of the system? Through replication, we can ensure availability of data and reduce transmission size by placing replicas to geographically distributed nodes. PAST is a protocol on top of Pastry. It enables consistent storage of data that is immune to churn. By introducing replication services we

have to care about data consistency so that the replicas represent the latest version of a file. One protocol that ensures consistency in Pastry is the repair mechanism. It makes sure the routing tables of the participants stay up-to-date when nodes leave the network.

Fault Tolerance

How should the distributed system react to the different types of errors that may occur? How should it react in case of node failure, in case of a procedure exception? Middleware implementations such as Pastry hide such failures from the application layer. This is also called transparency and is used for abstraction in layered architectures.

Security

We want to make sure data is accessible only by a user with appropriate access rights. In addition, communication needs encryption when messages trespass nodes that should not be able to intercept and read the messages.

2.1.5. Motivation for Usage of Distributed and P2P-Systems

Client-server systems in general, most of today's web applications such as marketplaces, mailing services, geographical navigation or social networks, have a number of challenges that distributed and p2p-systems address. The following list is based on Baun et al. [12].

Unbalanced Load Distribution

In client-server applications, the server carries all data, performs major computation and handles requests. Clients reduce to request and display functionality. A p2p-system benefits from every new participant that offers additional computing power and storage capacity.

Performance Bottleneck

While the server handles all requests which get more difficult with rising number of clients in a p2p-system requests distribute to other peers.

Single Point of Failure

A known weakness of centralized systems is that in case the central instance fails the entire system is out of service. In p2p-systems the norm is that new participants join while others leave. Single breakdowns of peers do not affect the system as a whole.

Single Point of Attack

As a recent discussion in the media about privacy and personal data in online social networks show it is easy to gather sensitive data in a client-server system. [1] In contrast, in a p2p-system data is stored across several peers and system-wide information cannot be retrieved from a single source. In addition, this issue is extended to censorship as well. It is much more difficult to censor a p2p-system, e.g. by attacking single participants while in a client-server scenario it suffices to censor the server.

Channel Bottleneck

In a client-server system, there is no possibility of direct communication. All messages first go to the server and then to the recipient. In a p2p scenario routing information may require several hops to reach the recipient. Nevertheless, all intermediate hops follow the same protocol and procedure as the sender and receiver node. Even though the single point of failure in client-server systems is addressed by the replication of servers it still requires a monitoring instance that determines when a server failed, which then becomes the single point of failure. P2p-systems entirely solve this problem by introducing distributed approaches for messaging, storage and computation.

2.2. Peer-to-Peer Systems

This section introduces p2p-systems. It lists specific properties of p2p-systems in general. In this section we see, that in addition to general properties each distributed system has in common, such as being able to scale up, being secure and fault tolerant, p2p-systems introduce a number of properties that are specific to this field. a list of these properties follows in the next sections.

2.2.1. Properties of P2P-Systems

Leof [19] lists six properties most p2p-systems have in common.

End-to-End Connectivity

Similar to the introduction of analog telephony where a physical line is reserved for the communication partners, in a p2p-system end-to-end connectivity is a key aspect. In terms of IP, this can be done by a TCP connection. In addition, the actual routing in the underlay can differ dramatically as long as it is possible for the peers to communicate directly.

Transient Connectivity is Norm

Compared to network-operating-systems or distributed computing where nodes stay connected to the network and node failure is an exception in a p2p-system transient connection is the norm. Machines leave and join the network so that the main challenge for an application to keep content and messaging consistent while transient connections take place. Kwok in his book on "Peer-to-Peer Computing" [20] distinguishes two additional aspects regarding connectivity.

a) Dynamic Population
Not only do peers join and leave the system, their overall number varies as well. Consider a system with 1000 participants where around 60 % of users are online. When such a system gains popularity its overall number of users may rise to 100.000 or 1 million. The online and offline behavior can vary as well.

b) Dynamic Topology
Here we consider the interconnection of peers on underlay level, i.e. network level. A system can cross ISP borders but stay within one country during a period of time and then gain international popularity with global interconnections. This leads to changes in transmission times and number of hops. From an architectural point of view, a p2p-system hides the topology of the network layer and introduces its own more abstract topology, such as tree-based or ring-based topologies. This way routing protocols can be independent of the network layer.

Focus on Individual and Content Identity

For two participants to communicate it is not important to know IP addresses of their corresponding machines or IP addresses of machines that provide routing of messages. Far more important is to find the appropriate identity of a user in such a network so that one user can actually log on to a different physical machine and still be able to retrieve his profile information. In addition, retrieval of data files through their names or meta-data is much more important than their actual location across the network or location of their replications.

Distributed Ownership of Resources

Recent German news [1] discusses the economic use of personal data in commercial online social networks. This happens if data ownership lays in the hands of the service provider. From a technical point of view data in commercial online social networks stays in data centers. In contrast, p2p-systems handle data ownership differently. Only fractions of user's data are accessible by other users. In addition, data that is accessible is still useless for other users due to encryption mechanisms. In short - in a p2p-system every user owns only a fraction of the overall data of the system. As a result, the whole peer community has ultimate ownership of the p2p-system's data.

Distributed Control of Resources

Due to the heterogeneity of nodes in a p2p-system in terms of storage and computing capacity as well as network connection the p2p-systems need some way to adjust its performance to the nodes' capabilities. In comparison to a network operating system where the administrator may decide which task should perform on which processor or machine, a p2p-system decides where computing, storage and messaging takes place in a distributed manner. Since every peer has only a local view of the entire system this is the only way to decide upon the distribution of resources. In case of Skype [2] super-peers, nodes with high bandwidth and network capacity, take on a specific role to redistribute routing load.

Persistent Storage of Content

Connected to transient connectivity, persistent storage of content is crucial to ensure data is available when parts of the system go offline and other parts go online. One way to achieve this is storage of file replications. Here the right measure is important to find a balance between storage overhead and

file availability and download time. We've seen in the previous chapter that persistent storage and replication is also a key aspect of distributed systems in general.

2.3. Peer-to-Peer-Framework and LibreSocial

This section describes the layered structure of the p2p-framework and LibreSocial. It introduces briefly into the interfaces of FreePastry [21], the p2p-framework and a selection of OSGi plugins that comprise the LibreSocial application. Figure 2.2 shows the layered architecture of the p2p-framework and the LibreSocial application. The following sections describe the architecture bottom-up, starting with the internet (network) layer.

2.3.1. Internet Layer

For the context of this thesis, the internet layer is realized through a small-scale local area setup. Other realizations of this layer can be a large-scale local area network setup or a realistic global scale internet setup. We discuss both alternatives in chapter 7. The internet layer is the basis for all subsequent layers and provides packet routing, sending and receiving services to the p2p-overlay layer, which is the next layer we discuss.

2.3.2. P2P-Overlay Layer

As we have seen in the goals of distributed systems, one of the aspects is transparency. The p2p-overlay does exactly that. Regardless of the underlying physical interconnections between nodes such as a local area network or a global-scale internet connectivity, the p2p-overlay hides the complexity of physical connection and introduces an abstract service layer. This layer uses messaging and routing services of the internet (network) layer and provides a service for routing messages from one node to another in an end-to-end manner. To identify sender and receiver it often introduces a so-called identifier space, that allocates a unique id to participants and data objects. Here, several goals we discussed in the p2p-systems section come in. We have end-to-end connectivity and focus on peer and content identity.

Two major protocols of p2p-overlays are Pastry [15] and CHORD [23] which both have open source implementations fro which we name two - openChord [24] and freePastry [21]. While CHORD organizes the peers and resources in a ring topology, Pastry uses prefixes of the identifiers to locate peers and resources, so that numerically closest object and peer ids fill up the routing table. as we can

see, the general aspect of naming, we discussed in the section on distributed systems, is tackled by the p2p-overlay as well.

The p2p-framework bases on FreePastry. FreePastry provides services to route messages end-to-end, look-up objects or peers provided they have been allocated a unique identifier, and a service to store an object or contact information for a specified id. We can see, how we reached the p2p-overlay interface in Figure 2.2. We refer to the functionality of storing and retrieving information given a unique identifier while the data is stored over physically distributed machines using the term distributed hash table (DHT). The next section introduces the p2p-framework layer, which is the main focus of this thesis.

2.3.3. P2P-Framework Layer

The p2p-framework addresses several key concepts we discussed in the distributed systems and p2p-systems sections. First of all, it addresses the consistency and replication aspect of distributed systems. It does so, by using an application that builds on top of the p2p-overlay and is called PAST [25]. PAST ensures that data that is stored in the overlay is available even if the carrying nodes leave the network. A key aspect of PAST is replication, and the replication factor, which describes the number of copies (or replicas) of a data object is one of the available p2p-framework parameters as we will see in the deployment of the system in chapter 4.

Next, the p2p-framework addresses openness, another major goal of distributed systems in general. It further abstracts the service interface of the p2p-overlay and introduces a storage-dispatcher and message-dispatcher. These two core parts of the framework open up the implementation possibilities to a wide range of applications, provided these applications use the messaging services of the message-dispatcher or storage services of the storage-dispatcher. One of our main goals in this thesis is to measure the performance of both dispatchers.

Security, as well as distributed ownership of resources, key aspects of distributed and p2p-systems are also addressed at this layer. While the storage-dispatcher decides where to store data, i.e. local or remote storage, it also ensures that data is only accessible to those with appropriate rights by introducing public-private-key encryption.

One last component of the p2p-framework layer is the distributed monitoring system SkyEye.KOM [26]. More details follow in chapter 3 that describes means we use to evaluate the p2p-framework. At this point we reach the p2p-framework interface in Figure 2.2. A number of key aspects and goals is addressed at this point and we proceed to the application layer in the next section.

2.3.4. Application Layer

The application layer splits into two parts. The first part consists of independent software packages, that follow a predefined pattern. These packages are called OSGi bundles [27]. OSGi stands for Open Services Gateway Initiative. As the name suggests, it describes a standard for software packages. A key idea is that these packages, or bundles, provide a set of services that they offer to a so-called service registry. Other packages can look up a service in the service registry and use it, provided the offering bundle is available. this loose architecture offers new bundles the option to be activated at any point in time, or bundles that are not used anymore to disconnect from the system without affecting overall system performance.

At this point, a different aspect of openness is tackled. By choosing this architectural design for the application layer, not only is the p2p-framework open to different application types, the building blocks of the application are open as well and can easily be exchanged. This way, the application itself remains open for changes and improvements.

The current application is called LibreSocial [9] and focuses on providing online social network functionality. The software packages, or bundles, each cover a specific aspect of an online social network, such as the messaging plugin, that offers chat functionality, the groups plugin, that enables sharing data files to user groups, or the photos plugin to upload photos.

Each one of the plugins, that are software bundles at the same time, provides a set of services to the service registry. We can use a graphical user interface (GUI) to trigger these services. We examine this method in chapter 4. We also can use a special bundle, the so-called tester plugin, that offers us a method to trigger plugin actions on remote peers.

In combination with a scenario file, that contains a set of actions to be triggered, this plugin offers us a powerful tool to setup and conduct a variety of experiments and tests, focusing on different aspects of distributed and p2p-systems.

At this point, we reached the commands interface in the architectural overview. Our main focus will be on the interaction with the services of the bundles through the tester plugin. When we reach a sufficient amount of data on the level of application in chapter 5 we take an additional step of analysis and descend one layer down to the p2p-framework layer. with the monitoring protocol SkyEye.KOM and a set focus on key aspects of the system to be examined, see chapter 3, we are ready to evaluate the system in part II of this thesis.

2.4. Summary

In this chapter, we categorized the system to be examined. we have seen, how p2p-systems evolved from distributed systems, what properties of distributed systems are of importance when analyzing such a system, and what characteristics p2p-systems have that distinguish them from other distributed systems. This way, we choose a focus for our further analysis. Furthermore, we took a close look at the architecture of the p2p-framework and the application that currently runs on top of it - LibreSocial, an online social network. We have discussed how the p2p-framework addresses the goals and shares common properties with distributed systems and p2p-systems. We have also laid out our benchmarking approach, using the application layer interface to create system load and the analyze overall system performance. The next chapter introduces metrics - scales focusing on specific system parameters, and means to measure them - a distributed monitoring protocol.

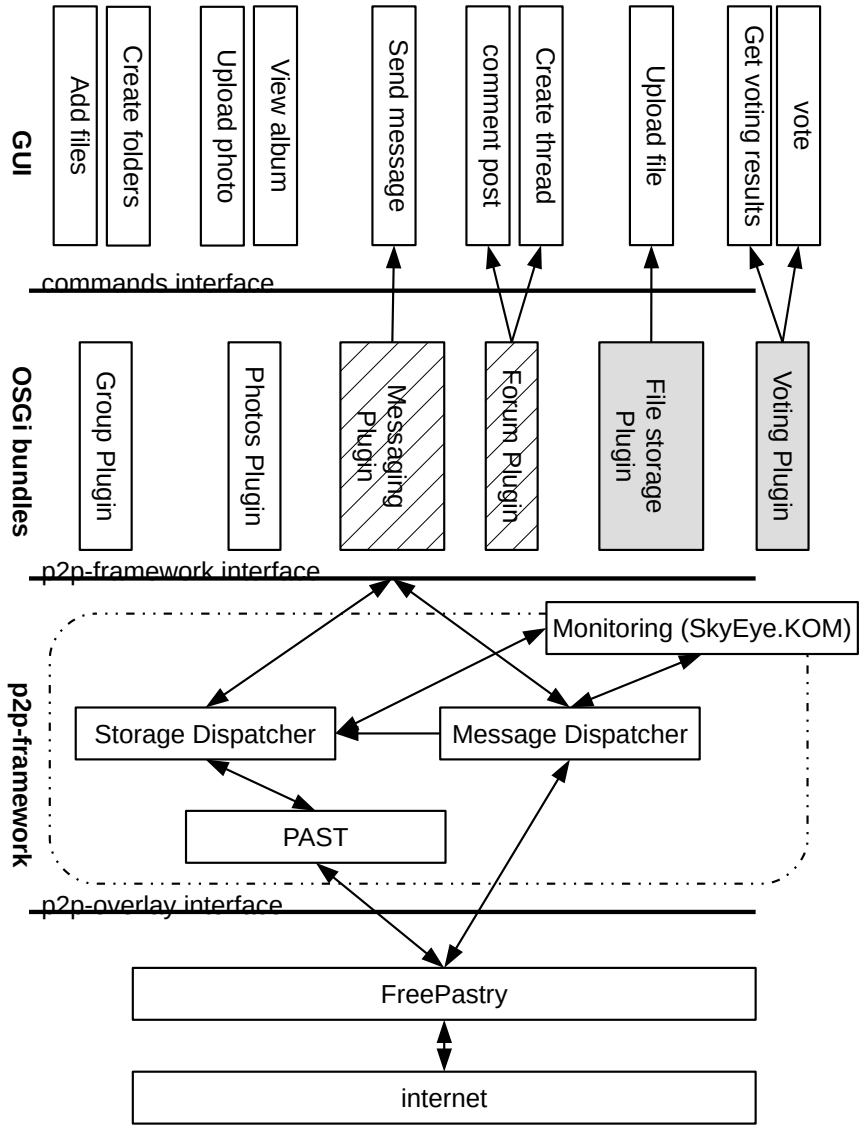


Figure 2.2.: LibreSocial and P2P-Framework Architecture, based on [22]

Chapter 3.

Definition of Metrics and Means for Evaluation

The following sections describe what aspects of the p2p-framework we choose to evaluate and how we evaluate them. The next section introduces a set of metrics - scales for specific system properties - such as scalability, availability, and robustness. We follow these aspects throughout the rest of the thesis. we introduce a calculation formula for each of these metrics. The second section describes distributed monitoring protocols. with these, we learn how to gather statistical information in a distributed system without relying on a central instance.

3.1. Metrics for Evaluation of Peer-to-Peer Systems

Throughout the previous sections, we've discussed properties that are specific for p2p-systems and in some cases their connection to the general field of distributed systems. In this section, we describe metrics to evaluate the performance of p2p-systems in general. This way we choose metrics for later evaluation of the p2p-framework and the online social network LibreSocial. Kwok [20] names two categories of p2p-system evaluation.

a) System-Oriented Metrics
These metrics show system-wide measurements. As we will see in later chapters, by the means of distributed monitoring we can measure system-wide storage, messaging and traffic.

b) Peer-Oriented Metrics
These metrics tell us what experience a peer who uses the system may expect. These values result from system-oriented measurements divided by the total number of peers in the system. These show approximate measurements as capacities of peers vary. This is also a result of distributed monitoring. Other monitoring procedures, e.g. track system performance locally and periodically store results on a central server, can provide a detailed view but require additional server capacity and lead to same

Metric Name	Summary	Calculation
Availability	probability of successful file transfer	$p_A = \frac{\#successfullTransfers}{\#requestedTransfers}$
Robustness	probability of reaching a certain availability or download rate given certain peer dynamics	$\mathscr{P}(p_A \leq 90\% \vee avg(down_rate) \leq 50KBps)_{\mid peer_dynamics=model_A}$
Scalability	probability of stable availability or download rate given population growth	$\mathscr{P}(p_A = 70\% \vee avg(down_rate) = 30KBps)_{\mid population_growth=10\%}$
Server Cost	cost of system maintenance regarding ratio of required stationary machines divided by sum of all devices	$avg(ISP_costs) \times \#static_peers_{\mid \frac{\#static_peers}{\#all_peers} = \frac{2}{3}}$
Download Rate	peer oriented metric	$avg(down_rate)$

Table 3.1.: Metric Summary and Calculation

results when calculating average measurements. More details on Monitoring follow in section 3.2. The next two sections define system-oriented and peer-oriented metrics.

System-Oriented Metrics

Availability

Availability is measured as a probability of a successful file transfer or file download. This transfer can involve successful look-up of data, according to its ID, routing of signaling messages across other participating nodes and finally the download of one or several peers who store the data, replications of data or parts of distributed data structures. Obviously, this measurement involves and gives a quality measurement of the system. It is affected by online and offline behavior from the peers.

Robustness

Robustness describes the probability of providing a given level of performance, e.g. average download time or average availability subject to a particular model of peer dynamics. Let us assume the average download time for peers in an experiment is 4 seconds. Let us further assume peers split into 5 groups, such that each group is 20 % of the total number of peers. Robustness shows the probability of the average download time to reach 4 seconds when each hour a new group goes offline and another one comes online. Then we can change the group size and or online duration and measure the availability. For the remaining thesis, we alter this definition of robustness. We want to find out where the limit of system performance is when measuring messages, storage size, and traffic. We want to find out how many messages can successfully be transferred, what data size can be stored and retrieved.

Scalability
Scalability refers to the capability of the system to provide a stable level of performance in terms of download time or average availability while a given rate of population growth occurs. For example, the effect on download time when system population grows linearly or exponentially is a question scalability answers.

Server Costs
Although the p2p paradigm does not require servers at all, for a real implementation e.g. VoIP telephony servers setup signaling between participants who then exchange video and sound end-to-end. Recent developments e.g. WebRTC [28] [29] also require so-called STUN servers whose task it is to establish a connection between two nodes behind a firewall or NAT. Therefore the server cost can be a first, qualitative statement to compare different p2p-systems.

Peer-Oriented Metrics

Download Time
An important question from user experience point of view is the responsiveness of a system. We want to measure how long it takes for a file to transfer, a blog to load or a profile to store. Replication level, peer connectivity, local uploading and downloading rates all affect the user experience that is best expressed in the download time. In addition to this metric as proposed by Kwok [20] we add storage size and messaging metrics for the evaluation of the p2p-framework.

Table 3.1 summarizes how we choose to calculate the measure for a given property of the system. An example is given, to evaluate the availability of the system, we count the number of successful file transfers. We can harness this information using the monitoring system that is part of the p2p-framework. In addition, we need the number of initiated transfers which we can extract from the scenario of the experiment. To calculate robustness, we can use either the availability value we computed before or a measurement of an average download rate that the monitoring system measures. In addition, we have a side condition of a specified peer dynamic model. In short, we examine if availability or the average download rate measure up to a value we expect when peers come online and go offline following a certain pattern.

3.2. Monitoring the P2P Framework

This section introduces a tree-based monitoring protocol for p2p-systems - SkyEye.KOM [30] [22]. There are several approaches to gather statistical data in a distributed system. One intuitive approach

is to let all nodes send their measurements to a central node and let it calculate the aggregated value. It may then disseminate the aggregated value to all nodes.

The advantage of such an approach is the easiness of its implementation and deployment. The main disadvantage, as we already seen in the discussion on distributed systems in general 2 is the single point of failure of the central node and the inability to scale up. With a rising number of nodes in the system, the central node needs more networking capacity to gather and provide measurements from and to all participants.

A different approach is to calculate measurements in a distributed way. We distinguish here between gossip-based and tree-based approaches. There are, of course, further methods to monitor distributed systems. We first give a very brief overview of gossip-based monitoring protocols before we explain our choice, tree-based monitoring, and its implementation SkyEye.KOM in more detail.

3.2.1. Gossip-based Monitoring

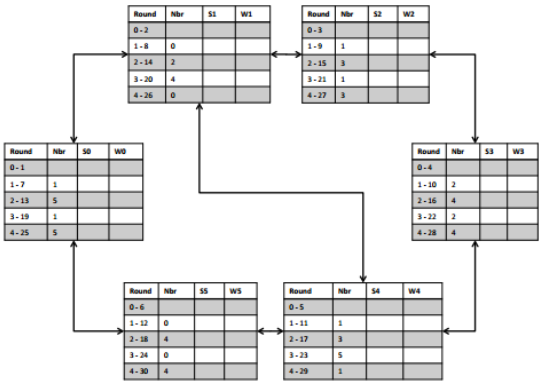


Figure 3.1.: Gossip-Based Monitoring

Figure 3.1 shows an example of gossip-monitoring in a network with six participants. These participants follow a certain protocol to exchange their local value with a peer they randomly choose in each round. In each round of communication, this node exchanges its value with values of the communication partners. This way, each node involved in this exchange gains one or several new measurements. These measurements together with the own measurement of the node aggregate into a new local measurement value. After several communication rounds, this local value converges towards the desired system-wide value. In specific, this approach works to calculate overall node count of the system, as well as to calculate minimum, maximum and average values of e.g. total bandwidth consumption or total storage capacity of the system.

The advantages of gossip-based monitoring protocols are that the protocol is applicable to different types of p2p-systems. In addition, the computed values converge towards the actual system value. Therefore it is practical for computing estimations for the desired information in a distributed system. A major disadvantage of the gossip-based monitoring protocol is the redundancy of communication in case both communication partners already have the same value. In addition, this protocol has a high consumption of network traffic.

3.2.2. Tree-based Monitoring

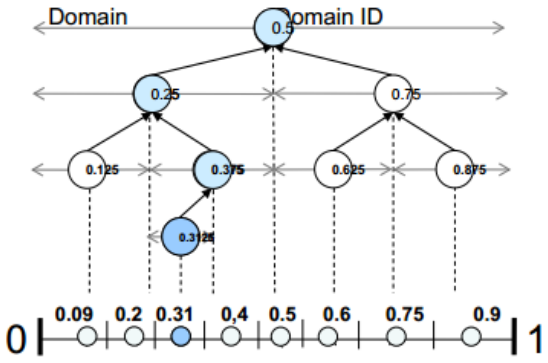


Figure 3.2.: Tree-Based Monitoring

As the name suggests, in tree-based monitoring protocols the nodes arrange themselves into a tree topology to exchange measurements. The measurements follow the tree topology from leaf up to root for measurement aggregation and from root down to the leaves for result dissemination.

Figure 3.2 shows the general idea of tree-based monitoring. At the bottom of the figure, we see a scale from 0 to 1. It symbolizes the unique identifiers of peers and objects in the network. Based on this identifier, peers compute their position in the tree topology. In addition, identifiers for parent nodes can be computed as well, so that the communication between nodes of the monitoring tree is well-defined.

Therefore, it is necessary for each node to establish its position in the monitoring tree. While gossip-based monitoring doesn't require a unique identifier for measurement calculation, a tree-based protocol relies on the unique identifier for each node. Consequently, a requirement for tree-based monitoring in a p2p-system is that it has a structured p2p-overlay, such as CHORD or Pastry. The main advantage of tree-based monitoring is its high precision and low overhead of network traffic. On the other hand, its applicability is limited to p2p-systems that provide an identifier space.

3.2.3. SkyEye.KOM

One implementation of a tree-based monitoring protocol is SkyEye.KOM [30] [26]. It is one of the building blocks of the p2p-application-framework that is discussed in this thesis. For an architectural overview and more details on the interconnection with SkyEye.KOM and the other p2p-application-framework components please see chapter 2.3.

SkyEye.KOM requires a structured overlay, in this case, Pastry with its open source implementation FreePastry. This way, each node has a unique identifier in the overlay and is able to calculate its responsibility and position in the monitoring tree. In addition, each node is able to calculate the position of the monitoring tree parent node to send the measurements to. Furthermore, when child nodes contact the current node to send their measurements, the current node stores their positions for result dissemination. Using this method it has its own position in the tree, as well as root and child node positions. This way it can aggregate measurements and send them to the parent node and disseminate results to child nodes.

SkyEye.KOM provides us with means to gather statistics on storage and messaging capabilities in the p2p-application-framework. We are now able to calculate availability by gathering data on storage and messaging, robustness by measuring the download rate and scalability through measurements of the peer count.

In this chapter, we defined aspects of focus for the further benchmarking analysis - scalability, robustness, and availability. We chose a method to calculate these. In order to gather statistics for the calculation, we introduced distributed monitoring protocols. We distinguished between gossip-based and tree-based monitoring protocols and chose SkyEye.KOM as an implementation of tree-based monitoring. Part II of this thesis explains preparation and deployment steps in order to run the benchmark experiments. Chapter 5 evaluates the outcomes produced by the tree-based monitoring protocol SkyEye.KOM discussed in this section.

Part II.

Benchmarking and Evaluation

Chapter 4.

Deployment, Preparation and Execution

In this section, we describe step by step which actions need to be taken to run a simulation in the pc-lab. First of all, we describe the procedure to compile an executable binary file of LibreSocial. Next, we describe the deployment of the binary file to the computers in the pc-lab and configuration of the deployed system. The next section 4.2 gives detailed instructions on running the experiments and section 4.3 shows how to extract result measurements and prepare tables and diagrams for publication.

4.1. Preparing the PC-Lab

4.1.1. Compiling the source code

The following instructions assume that Eclipse Kepler for RCP developers is installed, the java paths configured accordingly and the source code of LibreSocial is downloaded or pulled from the gitlab repository `gitlab.cs.uni-duesseldorf.de:cn-tsn/project/libresocial.git`. Furthermore, for questions regarding the required Java version and Eclipse setup, a how-to document exists showing a step-by-step instruction. This document is on the DVD in the attachments to this master thesis and it is part of the gitlab project as well.

The following package contains a `README.txt` and the configuration files we use to create a binary file for LibreSocial: `org.lifesocial.ui`. Here two folders are important: `/postbuildscripts` and `/ProductConfigurations`. The product-configurations folder contains two configurations. The first is `libresocialdeployment.product`. We can use this configuration to manually test the plugin functionality. As we can see in diagram 4.1 we can start LibreSocial within the Eclipse development environment by clicking on `Launch as Eclipse application`.

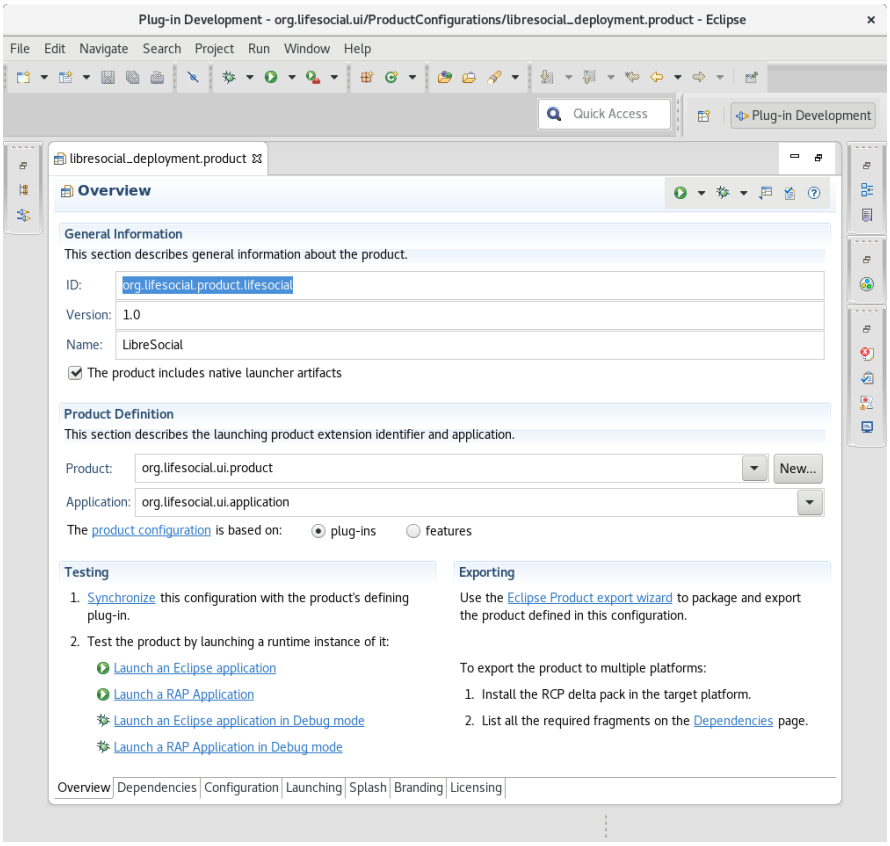


Figure 4.1.: Launch LibreSocial within Eclipse

A browser window opens and we can experience LibreSocial from a user point of view, create or join a network, invite friends, chat, etc. Figure 4.2 shows a very brief impression of the graphical user interface. It is possible to start LibreSocial on the same machine twice or more times to get a first impression of its functionality. The second configuration `libresocial.product` starts LibreSocial in a test mode. Here, by defining the IP address of the master node in the appropriate configuration file, we use the tester plugin component of LibreSocial. We run this configuration on one or several machines and delegate the test cases to the slave nodes with the delegator-view. More on the execution of test cases is in the next section. We use this configuration file in all measurements that follow in chapter 5 of this thesis.

After the decision to run LibreSocial from a user point of view or to perform system-wide tests we proceed by clicking on: `Eclipse Product Export Wizard`. Although a number of parameters are recommended in the previously mentioned `README.txt`, experience shows that the default setting works fine. The only important setting is the name of the root folder that needs to be `Libresocial`. Please note the correct spelling that is case-sensitive. Once we confirm the binary file is generated. In some case, there is an error message at the very end of the compilation process that does not have any effect on the execution of the binary.

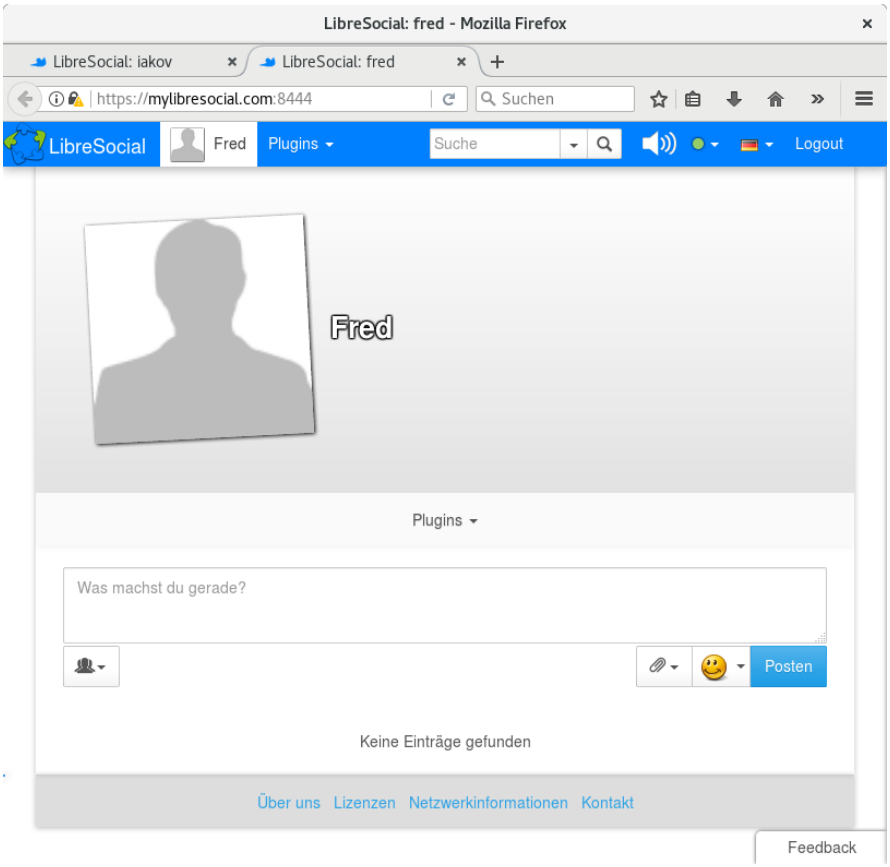


Figure 4.2.: LibreSocial from User's Perspective

One thing to notice at this point is that in previous work it is mentioned to make binary files for Linux, MacOS, and Windows. For this thesis, we compiled LibreSocial for a 64-Bit Linux system only. Furthermore, the Unity Desktop causes trouble with the tray icon of LibreSocial. On KDE, LXDE, and Gnome the tray icon works fine. The next section describes how to deploy the binary file to the test environment.

4.1.2. Deployment

Our choice for deployment and remote execution is bash. It is the name of a terminal that is available on most modern Linux systems, as well as the name of a powerful scripting language. Besides its own set of commands, it depends on Linux commands to automate repetitive tasks or simplify command execution.

We introduce an excerpt of the commands that are most important in the scripts we prepared during this thesis. All scripts can be found on the DVD in the attachments, as well as the gitlab repository.

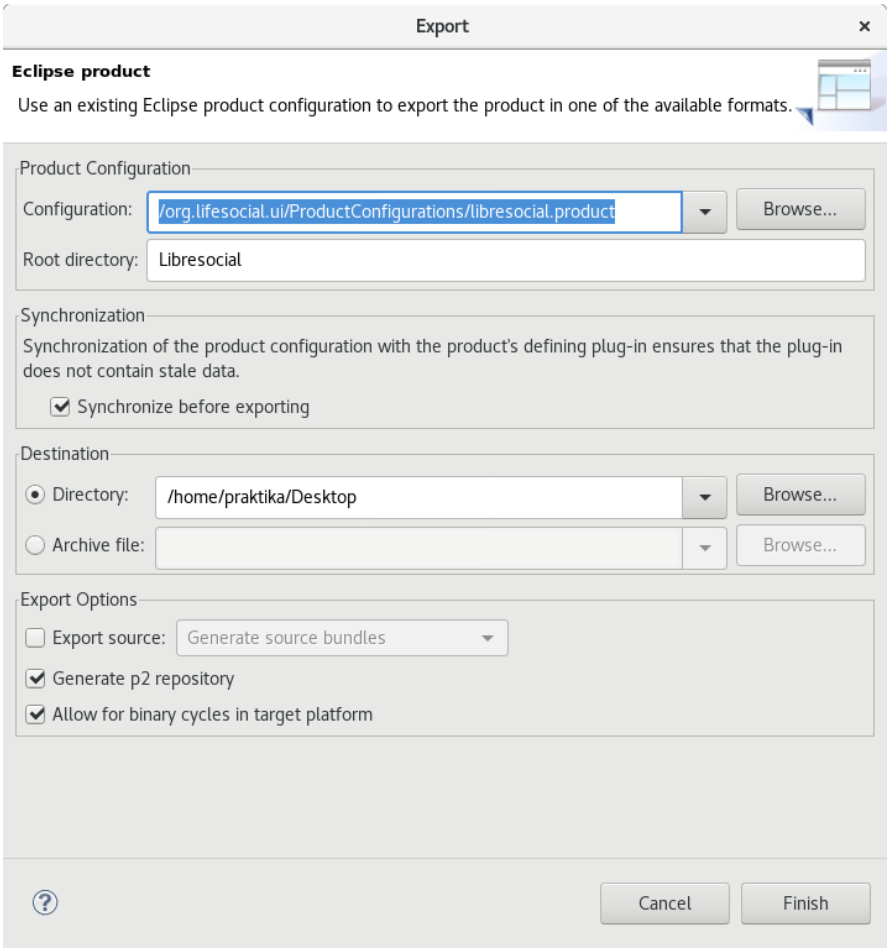


Figure 4.3.: Export Wizard for LibreSocial Compilation

Listing 4.1: deployment.sh

```
# networking commands
ifup enp4s0
ifdown enp0s25
ip addr show

# secure connection
ssh-copy-id 192.168.222.2
ssh praktika@192.168.222.105

# remote command execution
scp /home/praktika/Libresocial/resources/ \
  praktika@192.168.222.71:/home/praktika/Libresocial/
ssh <remote-ip> "sudo␣rm␣/<folder>/*.txt*"
```

```
sudo nohup ./LibreSocial 0<&- 1>./libsoc$i.log \
  2>./libsoc$i.log &

# data structures
array=(
        hiwi@192.168.222.105
        praktika@192.168.222.21
)

for address in "${array[@]}" ; do
        echo "$address"
        sleep 5s
done
```

The example script, Listing 4.1 consists of four parts. The first part concerns the machines in the pc-lab. When we start the machine for the very first time after a shutdown or a lecture course, they install the operating system from the network. Per default, after installation, the network interfaces of the machines are off. Therefore we provide the commands with the matching network interfaces.

The next command `ip addr show` helps to find out the current IP address of the machine. We need this address to loop over a number of addresses to execute commands on several machines automatically. In addition, future benchmarking in the High-Performance Computer dynamically configures IP addresses of the cluster. Therefore, to e.g. decide whether one node is a master or slave node this command can be of help.

The next set of commands focuses on a secure connection between the nodes. For the automatic execution of commands from one node to another node, it is necessary to exchange public keys. We can make one machine known to all others by copying its public key to a file that contains all known hosts of the remote machine with `ssh-copy-id`. An example is the master node from which we conduct the experiments. It is useful that the master node can connect to every other node without being asked for a password during execution of the experiment.

The third set of commands shows different ways to remotely copy and remove files. The last command helps to run commands silently, i.e. without a requirement for user input and with the output redirected to a text file. The `scp` command allows us to distribute the compile LibreSocial binary to every machine of the pc-lab. The `ssh $address "sudo rm *.txt"` helps to clean up log-files and also statistic data as well as p2p-framework data after the experiment. We run this command before a new experiment to have an empty setup that is not affected by the outcome of previous experiments.

The last set of commands employs the array data structure and for loop of the bash scripting language. These two allow executing a set of commands on each machine consecutively. The `sleep` command helps to give the LibreSocial application time for stabilization before the next action starts. A use case is the join of new nodes to the network, that takes around 3 seconds per node. A simultaneous execution shows that not more than eight nodes can join at the same time. Therefore we add a padding of 2 seconds so that the join procedures takes 5 seconds in total.

With these basic tools, we proceed to the next section that gives details on experimental setup. For further information on bash and Linux commands we refer to the following two resources [31] [32].

4.2. Running Experiments

This section is a detailed description of the experimental setups. First, it shows LibreSocial parameter setup that is hard-coded into the LibreSocial binary file. Second, it shows the configuration file setting of the tester plugin. A separate section describes the physical setup of the machines as well as the deployment procedure and intervention during the experiment execution. This section also describes how monitoring data is parsed to diagrams and tables. Last we discuss the scenarios in a general manner. Scenarios are lists of command that the peers execute at a given time.

4.2.1. LibreSocial Parameters

LibreSocial Parameter	Value
heterogeneity awareness	on
replication factor	6
object size calculation	on
social metrics	on
automatically accept friend request	on
weak (mobile) nodes	off

Table 4.1.: LibreSocial Parameters

Test Plugin Parameter	Value
master node IP adress	e.g. 192.168.222.2
enable test plugin	true
location of timed test plan	e.g. /LibreSocial/resources/testplan.txt

Table 4.2.: Test Plugin Parameters

Table 4.1 shows LibreSocial parameters used throughout the experiments. The configuration file is part of the `org.lifesocial.core.utilities` package. These parameters need to be set be-

fore LibreSocial compilation as they are not available afterward. The `LibresocialConfig.txt` has comments for each of the available parameters. Most of the parameters in this thesis are left as they are. *AutostartMonitoring* is on for all subsequent experiments. *AcceptFriendRequest* is on to provide interconnections between peers. This is a requirement for most of the plugins. *SocialMetrics* is on to provide information on the plugin behavior. To gather information on network, messaging and storage only this parameter can stay off to save storage capacity and run the application more smoothly. *ReplicationFactor* stays at six.

4.2.2. Tester Plugin Configuration

Table 4.2 shows the configuration of the tester plugin. This file is editable at any time. The most important information here is the location of the test plan (scenario for the experiment) and the IP address of the master node. Other settings sty unchanged. Its location is the *resources* folder of the *com.p2pframework.tester* package. The resource folder needs to be in the LibreSocial root folder for the testing to run appropriately.

4.2.3. Physical Setup

For the subsequent experiments, a PC lab with eight machines is used. They connect via a gigabit ethernet switch which connects to a second switch that leads to the public PC lab. For the experiments, we use one of the eight machines as the master node so that 7 machines remain to run the application. In the second experiment, we choose to use one of the public machines as the master node so that eight machines can all run the application. The machines in the private PC lab have 4GB of RAM and can run up to 8 instances of LibreSocial before RAM is full and swapping starts. In the public PC lab, the machines have 8GB of RAM and can run up to 16 instances of LibreSocial in parallel.

Through the *ssh* command and or remote desktop control it is convenient to start and stop the experiments from the public PC lab. It is also possible to connect a private notebook to the switch in the public PC lab and use it as a master node. The only requirement is to add the appropriate IP address to the test plugin's configuration file.

To run an experiment we copy the binary files to each machine. The master node starts one instance of LibreSocial. If the correct IP address is in the configuration file and the resource folder is in place the OSGi console lists all registered and logged in user. Two bash scripts help to start and stop a fixed number of instances on each of the machines. We can observe the join procedure in the master node console that fills up with names of registered and online users.

The next step after joining is to let the peers send friend requests, as a prerequisite for the following test cases. This is done via OSGi console *sendFriendRequestInOrder* command or test case delegator view of the master node. Last but not least the master nodes initiates execution of a timed test plan (experiment scenario) by sending the corresponding message.

Once the timed test plan is done we stop the master node. A second bash script remotely shuts all instances of LibreSocial down. Statistical data is in the *SkyEye.KOM* folder in the root directory of LibreSocial. With a .jar file, it converts to a data table and Gnuplot script that write the graphics. For detailed instructions refer to Appendix IV

4.2.4. Scenarios

The most important parameter for each of the different benchmarking tests is the scenario. It has the most influence on the outcome of the results, once system parameters and application parameters are set. It consists of a text file, that is stored in the `LibreSocial/resources` folder at each of the physical machines. In case several instances of LibreSocial run on the same machine, they share the scenario file. The scenario file contains a list of test case commands. These trigger a set of actions in the LibreSocial application. For example, the upload-photo command first ensures that the preconditions are met, then uploads a photo from the `resources` folder to the application. These actions can directly be typed into the OSGi console of the individual peers, as well. List **??** shows an excerpt of available commands in form of a scenario file.

Listing 4.2: scenario-example

```
1x TestCaseChangeProfilePicture 1m
1x Wait 1m
1x TestCaseChangeProfileInformation 1m
1x Wait 1m
1x TestCaseViewOwnProfile 1m
1x Wait 1m
1x TestCaseViewFriendsProfile 1m
1x Wait 1m
5x TestCaseSendWallPost 2m
1x Wait 1m
5x TestCaseCommentWallPost 2m
1x Wait 1m
5x TestCaseViewOwnWall 2m
1x Wait 1m
5x TestCaseViewFriendsWall 2m
```

```
#
#  ...
```

To list all available commands we can type `help` to the OSGi console. In addition to the command name, the first number indicates the repetition count, so that the same command is executed the given amount of times. The number after the command name is a time frame in minutes, in which the command shall be executed. The test cases are independent and do not require a fixed order. Once the peers finish executing the commands in a scenario file, they wait for additional commands. To shut down an instance we can use the `exit` command of the OSGi console. The next sections show, how to gather results once the experiment is done.

The next section shows how to extract statistical information from monitoring data.

4.3. Gathering Results

This section describes in detail how to extract information from monitoring data and prepare tables and diagrams for publication. First of all, we show how to transform the .sql file, which is the output of the monitoring protocol SkyEye.KOM, to a .gnuplot file and to graphs showing the metric results. Next, we focus on data extraction and summary with R [33], a programming language that focuses on statistical data analysis and helps us to prepare tables for export to a LateX format.

4.3.1. Data Preparation

SkyEye.Kom writes all data into the

`/resources/SkyEye.KOM/`

folder in the root directory of Libresocial during experiment execution. For experiments that take longer than 30 minutes SkyEye.KOM writes several intermediate snapshots to this directory. This allows experiment evaluation at intermediate steps and is especially useful for long-term monitoring. The monitoring protocol also enables each node in the network to gain a system-wide “sky” view on the network, so that statistical data of each node is the same. Therefore we focus on the master node and choose the latest file that SkyEye.KOM writes to the statistics folder. Its name has the following structure:

`statistics_output_localNode_admin_<date>_<time>.sqlite3`

Here <date>_<time> denotes the timestamp of experiment execution. Furthermore, this file name contains brackets and white spaces that need to be removed to further process this file. For a single evaluation we do this manually and also prepare a bash script that contains commands to rename several files simultaneously:

```
generate_gnuplot.sh
```

This script and all following bash scripts and .jar files are on the DVD in the attachments to this thesis.

Next, we translate the .sqlite3 file to a .gnuplot file. In previous work, several java-classes have been developed to translate the .sqlite3 files into diagrams. They are located in the following package:

```
com.p2pframework.monitoring/sql2gnuplot/
```

During this thesis, we adjusted these source files and added two parameters to it, the location of the .sqlite3 file and a name for the new .gnuplot output file. Furthermore, we extracted this file as a runnable .jar file so it is now possible to convert the .sqlite3 file to a .gnuplot file from the console using the following command:

```
java -jar sql2gnuplot.jar <name of sql file> <name of output file>
```

The next section gives instructions on how to create diagrams and tables using the .gnuplot file.

4.3.2. Tables and Diagrams

In addition to the .gnuplot file we also get a .plt file as output of the previous step. This file contains a set of instructions to gnuplot, a utility that produces scientific plots. Simply invoking the following command starts the tool and we observe how new diagrams are written to the current directory:

```
gnuplot <filename>.plt
```

To change font size or output format from .pdf to .png we can edit the .plt file and replace the corresponding code sequence. In addition, we can alter the source code and compile a new version of the runnable .jar file.

To prepare tables, we use the statistical programming language R. The following requirements need to be done before we start. We split the .gnuplot file that contains all numerical data into two files and separate the commented lines, that mark the column names of the monitoring data, into a separate

file. Of course, this step can easily be done within the R software. Next, we take a look at a few code segments from the following R file that we used to extract exact measurement values:

Listing 4.3: data-analysis.R

```
# data import
p2p.data <- read.table("monitoring2-p2p.table")
p2p.captions <- read.table("monitoring2-p2p.captions")
colnames(p2p.data) <- as.character(p2p.captions$V2)

# extracting group plugin data
plugin.groups.table <- select(
  p2p.table,
  "TIMESTAMP",
  "GLOBAL_PLUGIN_GROUPS_ADDFILES_MEAN_COUNT_elements",
  "GLOBAL_PLUGIN_GROUPS_CREATEFOLDER_MEAN_COUNT_elements",
  "..."
)
# helper function
metric_value_groups <- function(t_start, t_end, m_name) {
  as.integer(max(filter(
    plugin.groups.table,
    (plugin.groups.table$time > 60 * t_start) &
      (plugin.groups.table$time < 60 * t_end)
  )[m_name]) -
    min(filter(
      plugin.groups.table,
      (plugin.groups.table$time > 60 * t_start) &
        (plugin.groups.table$time < 60 * t_end)
    )[m_name]))
}

# data extraction
metric_value_groups(73, 77, "create.folder")
metric_value_groups(77, 87, "add.files")

# table format
groups.row1 <- matrix(c(
  metric_value_groups(69, 73, "create.group"),
  metric_value_groups(73, 77, "create.folder")
```

```
),
ncol = 2,
byrow = T)
groups.row2 <- matrix(c(
  metric_value_groups(77, 87, "add.files"),
  metric_value_groups(87, 91, "delete.item")
),
ncol = 2,
byrow = T)
# merge into one table
cons.groups.table <- bind_rows(groups.row1,
                               groups.row2)

# latex generation
xtable(cons.groups.table)
```

The code 4.3 consists of five parts. The first part loads the .gnuplot file into R. Next, we choose what metrics we want to focus on. Here, we are interested in the groups plugin and fill in names of the corresponding metrics to be extracted from the data.

Please note that the exact column name of the metric differs from the diagram name that was generated in the previous step. Therefore double-check the correct column name in the file that contains the captions or using the colnames command in R. The third section introduces a helper function. We focus on data we extracted from the groups plugin, define a start time and end time of the time frame and the specific metric we want to observe, e.g. the number of added files. Now we are able to get the change rate of a metric given the time frame from the groups plugin, e.g.

```
metric_value_group(76, 86, add.file)
```

Returns 30, saying that the metric observed 30 file uploads in the time frame from 78 to 86 minutes. This function is especially helpful in connection with the scenario, where we find time frames for each plugin and its desired repetition rate. In addition, to find the correct metric for a plugin we can refer to table 5.1 in chapter 5.

The next part of the R file brings all data in a matrix format. We can merge several single rows of data into a table, provided the rows have matching column names. Tha last part of the source file is a very convenient way to generate lateX code used in publications and scientific texts. It converts the table structure of R into valid LateX code that we use for the tables in this thesis.

In this chapter, we have seen how to prepare the test bed and execute experiments. We focused on

compiling the source code, parameters that are of importance and scenario files that have the greatest impact on the results. Furthermore, we introduced a number of scripts that automate deployment and remote start of the experiment. Throughout the last sections, we learned how to extract desired data in form of tables and graphs. The next chapter is a core contribution of this thesis and presents evaluation results for robustness, availability, and scalability of the p2p-framework.

Chapter 5.

Evaluation of Benchmarking Measurements

Throughout this chapter, we present measurement results of the p2p-framework. We start evaluation with a regression test, to continue where previous work left off and check whether the framework and application are fully functional. We proceed by focusing on three aspects of system performance in great detail - availability, scalability, and robustness. In each chapter, we recap the metric we used for evaluation, present the scenario file as well as additional conditions required. at the end of each chapter, we present detailed measurement results and conclusions.

5.1. Regression Test

This chapter gives an overview of the p2p-framework and LibreSocial plugins. We do not focus on a specific property or parameter of the system. Instead, we conduct a regression test. Regression test in this context means, that we test whether the functionality described in previous work is available and fully functioning. At the same time, to harness additional information, we introduce a different amount of logical LibreSocial instances per physical machine. This way we can reach a node count up to 56 peers in this test. Further, we choose the scenario file from previous work, that includes most of the test cases available in the system. We introduce three setups. In the first setup, we use one machine as a master node and use the remaining seven to run one instance of LibreSocial each. The second setup uses one machine as the master node and starts eight logical LibreSocial instances per physical machine, making a total of 56 peers in the network during test execution. The third setup uses a staggered distribution of, two, four, eight and sixteen logical instances per physical machine that sum up to a total of 54 peers. We refer to the last setup as staggered setup in the evaluation of the regression test. We use our observations during regression test to plan further experiments with a focus on availability, scalability, and robustness in the upcoming sections

5.1.1. Regression Test Scenario

We start the regression testing with the same scenario as Schulte [34] in the previous work. We run the test plan three times and vary the number of peers per physical machine each time. The first time one node acts as the master node and seven nodes run one instance of LibreSocial making it a sum of 7 online peers. The second time, one node acts as master node while 7 peers execute 8 instances of LibreSocial each, making it a total of 56 online peers. The third time one node is master, three nodes run 2 instances each, 4 nodes run 4 instances each and two additional machines from the public PC lab run 16 instances each, making it a total of 54 online peers. We call the last configuration *staggered setup* in the evaluation.

The scenario contains all available test cases with a variety of durations and repetitions. Because of manual intervention during the join phase, especially for the staggered setup, the scenario does not start simultaneously. In addition the recommended *sendfriendRequestInOrder* test case is left out. All three runs consolidate into a set of graphics presented the following section.

5.1.2. Metrics for Messages, Storage and Time

Node Count

Figure 5.1 shows the node count during the experiment. For all three setups node count stays stable for the entire experiment duration. The only exception is at 70 minutes where one node drops in the staggered setup. This could relate to the Vote plugin. A more realistic explanation is the crash of one instance on the machine where 16 instances of LibreSocial run simultaneously. During experiments with 8 peers per machine with RAM of the machine being full and swapping activated a crash of a single instance occurred irregularly but predictably.

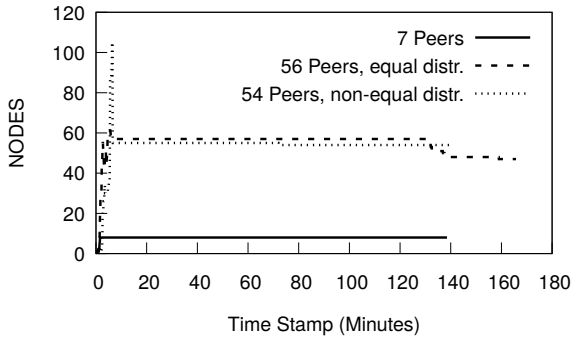
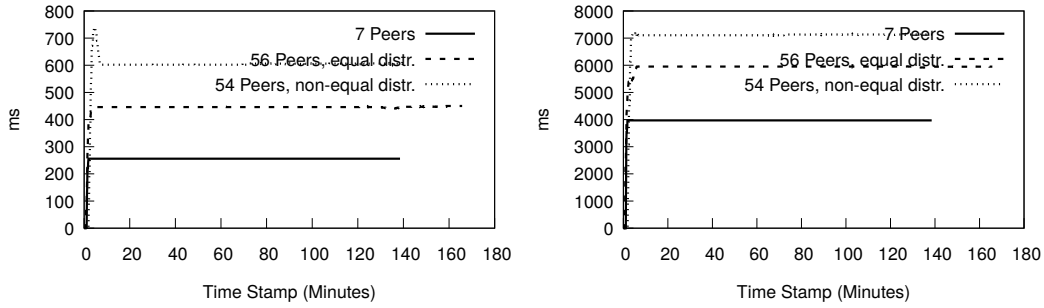


Figure 5.1.: Number of Nodes

Time

Registration and Join Network Time Figure 5.2 shows the registration and network join time. It takes less than 0.3s for the peers to join the network in a setup with 7 peers. For a network size of 56 peers with equal distribution on the physical machines, it takes only twice as long, around 0.5s and for a staggered distribution 0.6s. Compared to the previous work by Weifeng [7], although the registration time is significantly longer than the network join time, it is lower compared to previous measurements. In case of 7 peers, it takes around 4 seconds and for 50 peers between 6 and 7s.



(a) Join Network Time (b) Registration Time

Figure 5.2.: Join Network Time vs. Registration Time

Average Lookup Time Overall look-up duration, as shown in Figure 5.3 stays below 5ms for all three test cases. Between 0 and 20 minutes, the peers execute several actions of the Wall plugin. This appears to lead to look-up times up to 20ms in case of 50 staggered peers and 7ms in case of 7 peers. We note, that look-up time improves with rising number of network participants.

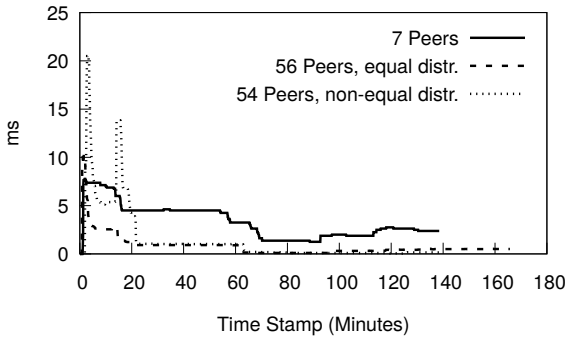
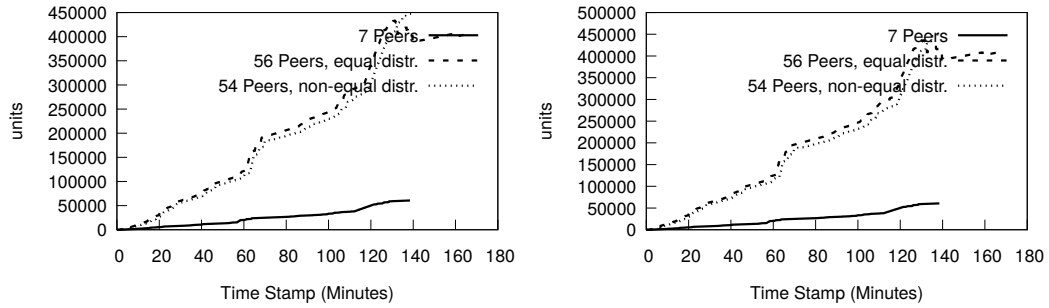


Figure 5.3.: Average Lookup Time

Messages

Messages Sent and Received by Message-Dispatcher Figure 5.4 shows the number of messages that the message-dispatcher handles. Notable increases occur around minutes 20, 60 and 120. The uploadPhoto command of the Photos plugin and the Vote plugin cause the increase of messages between 60 and 80 minutes. The uploadFileInFolder test case leads to the steepest increase around minute 120. The Groups plugin provides this functionality. 5.1 The Wall plugin causes the first rise around 20 minutes. In case the wall contains lots of comments their resolution requires extended messaging.



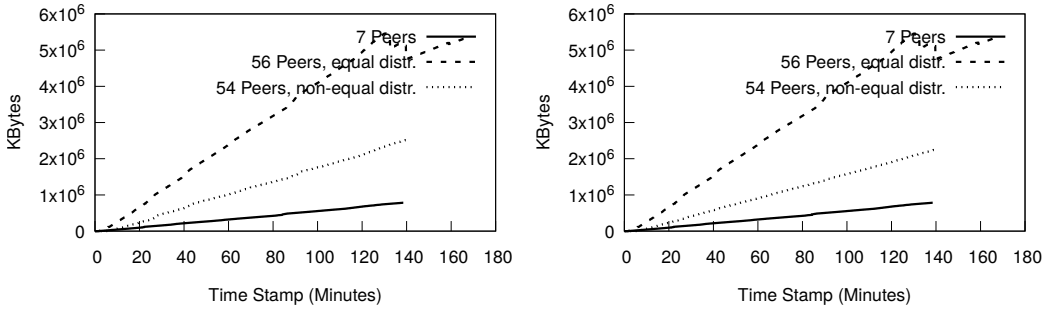
(a) Message-Dispatcher - Sent Messages

(b) Message-Dispatcher - Received Messages

Figure 5.4.: Message-Dispatcher - Sent vs. Received Messages

Storage Size of Messages by Message-Dispatcher Compared to Figure 5.4 Figure 5.5 shows the size of messages instead of their quantity. Both graphs are the same for 7 peers and 50 peers with equal distribution. This means for both that the message loss is close to zero. In a local area network setup with Gigabit Ethernet, this is what we expect. In addition, the rises seen in the previous diagrams do not appear here. This leads to the conclusion that although the Photo and Group plugin rise the number of messages, their size is not of importance considering the overall bandwidth consumed by the overall calls to the p2p-framework. From previous work [7] we also know, that most of the bandwidth of the framework at the moment goes to monitoring. Since all metrics information, currently 500 metrics, the monitoring message size sums up to 28 kByte being significantly larger than an average message.

Next, the number of sent messages in the staggered setup differs from the number of received messages. This consequent difference from the beginning of the experiment needs further examination. Also, there is a considerable difference in the amount of data in the setups with 40 peers. The equal distribution produces an overall of around 5GB traffic while the staggered setup stays close beneath 2GB. Finally, the small 7 peers setup produces around 0.5GB traffic which leads us to the conclusion that traffic rises linearly depending on the number of peers.

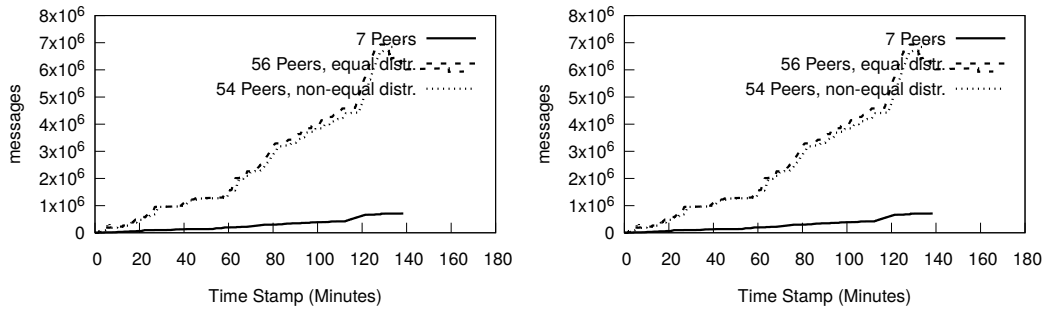


(a) Message-Dispatcher - Sent Messages Data (b) Message-Dispatcher - Received Messages Data

Figure 5.5.: Message-Dispatcher - Size of Sent vs. Received Messages

Sent and Received Messages by the Network For the messages sent and received on Network Layer, both diagrams 5.6 show almost the same graphs. This means that messages loss on Network Layer is close to zero. Additionally, the overall trend of the graphs resembles the graphs from the message-dispatcher, i.e. we observe the rise of message number around 20, 60 and especially 120 minutes.

We observe that both 50 peer setups stay close, with the staggered setup having fewer messages. The overall number of messages on Network Layer reaches 7M at the end of the experiment. With 400K messages produced by message-dispatcher, it is a fraction of 5 % of total messaging for the 50 peers setup. The number from previous work [7] states that message-dispatcher consumes 15 % of overall traffic. This leads us to the conclusion that message numbers of message-dispatcher do not scale with network size.



(a) Network - Sent Messages (b) Network - Received Messages

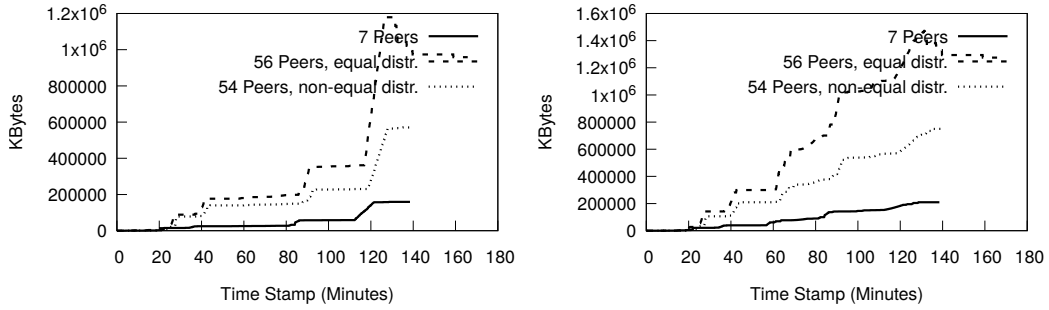
Figure 5.6.: Network - Sent vs. Received Messages

Storage

Data Retrieved and Stored by Storage-Dispatcher Figure 5.7 shows data stored and retrieved by the storage-dispatcher during the experiment. We see increases from 20 to 50 minutes, 85 to 95 minutes and 110 to 125 minutes. The Livechat and Messaging plugins cause the first rise. The Photos plugin contributes to the second increase. It is executed 50 times, therefore, the increase here is larger than the Messaging and LiveChat plugin. In addition, the Photo size contributes to this rise.

The Group plugin contributes to the last and most notable rise. We observe that overall stored data during the experiment is around 200MB for 7 peers and 600MB to 1GB for 50 peers where the equally distributed setup consumes most. The data retrieved from storage-dispatcher shown in Figure 7.4 indicates five rises at 30, 40, 60, 90 and 120 minutes.

Messaging plugin causes the first rise through retrieval of Inbox and Outbox messages. Next, the Search plugin calls retrieval functionality of the storage-dispatcher by executing a search 5 times. At 60 minutes the Vote plugin affects the retrieved data through the getVotingResult test case. The Photos plugin is responsible for the rise at 90 minutes. Finally, the Groups plugin causes the rise at 120 minutes by executing viewFolder and deleteFolder test cases respectively. The total amount of data retrieved by 50 peers ranges from 800 MB to 1.3GB. For 7 peers the total retrieved data size is around 200MB. Again the equally distributed setup generates the largest data size.



(a) Storage-Dispatcher - Stored Data

(b) Storage-Dispatcher - Retrieved Data

Figure 5.7.: Storage-Dispatcher - Stored vs. Retrieved Data

Used Size The results of this metric differ significantly from previous work [7]. In comparison to the rise and fall affected by the Photos plugin when running uploadPhoto test case and deletePhoto test case, both hardly affect the storage size used compared to the significant increase at 110 minutes. The Groups plugin especially the uploadFile function creates a huge rise in storage use from around 100MB to 5GB at the end of the experiment. This behavior needs further examination and comparison of the uploadPhoto functionality of the Photos plugin and uploadFileInFolder of the Groups plugin.

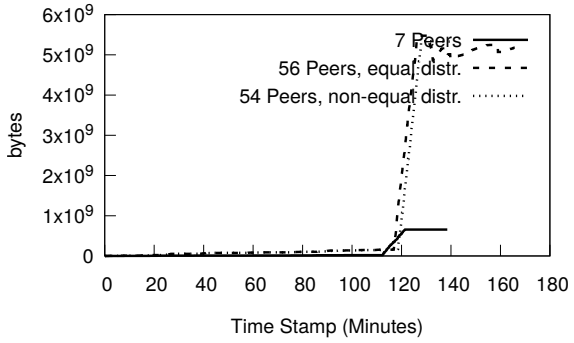
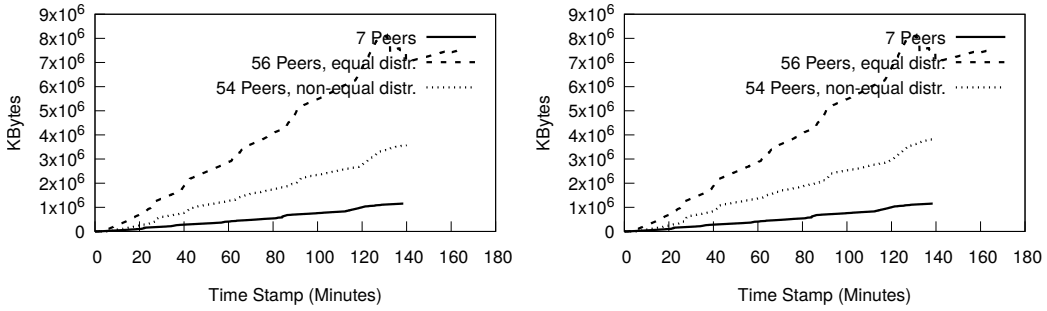


Figure 5.8.: Storage-Dispatcher - Used Size

Sent and Received Data by Framework Figure 5.9 show the overall data sent and received by the p2p-framework. The trend of these graphs is similar to the messages sent and received by the message-dispatcher. These are a majority of the total number of data sent and received by the framework. At the end of the measurements, message-dispatcher data sent of the overall data sent by the framework is around 70 % for the 50 peers leaving the remaining 30 % to the storage-dispatcher.

In addition, we observe the same rises at 30, 40, 60, 90 and 120 minutes as for the storage-dispatcher's sent and received data. These rises are not as visible or significant because they only contribute 30 % of the framework's data. The sent and received graphs of the framework are almost the same which leads us to the conclusion of lost data being close to zero. Nevertheless, the phenomenon of staggered peers receiving fewer data than being sent remains and needs further examination.



(a) Framework - Sent Data

(b) Framework - Received Data

Figure 5.9.: Framework - Sent vs. Received Data

Exceptions

Missed Calculations This metric shows the number of missed message and objects size calculations during the experiment. The monitoring calculates object and message size in case enough memory

for the application is available. Therefore this metric also is an indicator of memory consumption of the application.

Figure 5.10 shows the number of missed calculations for each of the three experiments. We can see that this number stays close to zero throughout the experiment and rises slightly to 10 around 120 minutes. During this period the bandwidth consuming uploadFileinFolder test case of the Groups plugin takes place. Therefore we expect a memory shortage at this point. This observation is a clear improvement from previous work, where the number would rise above 20 for a 10 peer experiment and 40 for an experiment with 20 peers. [7]

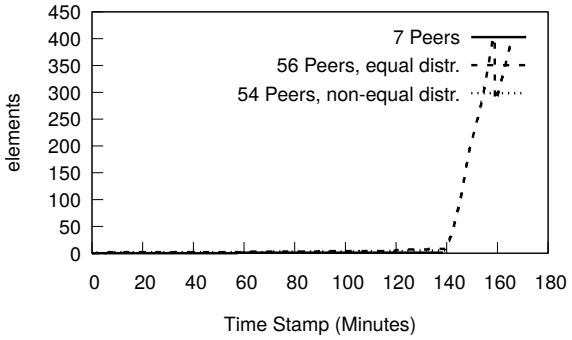
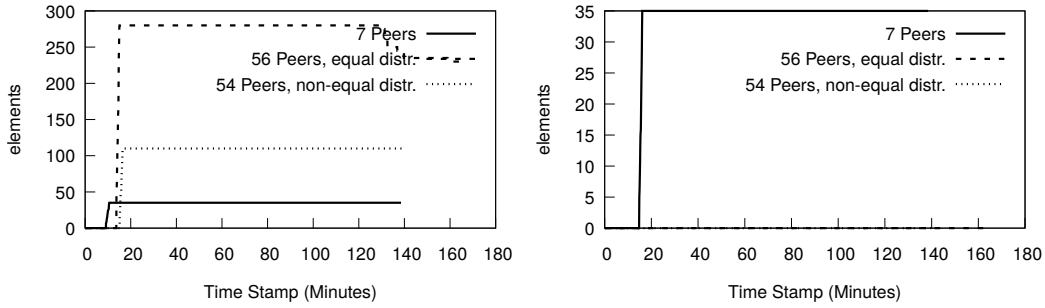


Figure 5.10.: Missed Calculactions

5.1.3. Metrics for Plugins

Wall Entries and Wall Views

The sendWallPost test cast investigates the functionality of the Wall plugin. Figure 5.11 shows the number of wall entries for the three experiments. From the timed test plan we know, that each peer executes the test case 5 times in 2 minutes. This results in a total of 35 (7 times 5) wall entries for 7 peers and >250 (50+ times 5) wall entries for the large setup. The equal distribution setup and small setup both meet this expectation. In comparison, the staggered setup shows an interesting behavior - the total number of wall posts is around 110. In case the machines that carry 16 instances of LibreSocial do not succeed in executing this test case this would lead to a total of 24 peers that post 5 wall entries each. This explanation needs verification. Figure 5.11 also shows the number of wall views. The test case runs at minute 18 of the experiment. The number of repetitions is 5 per peer. We observe successful results for the small setup with 7 peers. In both large setups, the test case doesn't lead to any changes of the overall view number.



(a) Wall Entries (b) Wall Views

Figure 5.11.: Wall Entries and Views

Messaging

Outbox - View The Messaging plugin shows a similar response to the test case as the Wall plugin. A viewOutBox test case runs 5 times for each peer. The 7 peers setup shows the expected number of 35 (57 times 5). So does the equal distribution setup, showing 270 vies. the staggered setup shows around 110 views. A similar explanation as for wall entries may apply and needs confirmation.

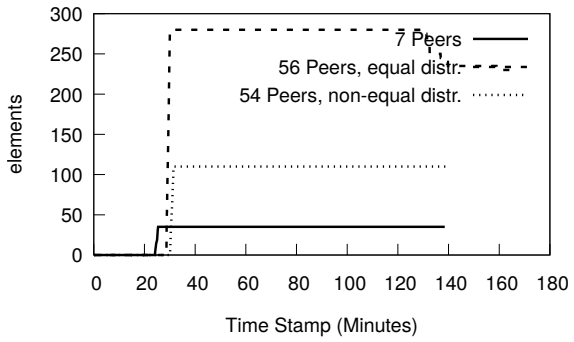


Figure 5.12.: Outbox - number of Views

Search

The search test case starts at minute 40. Similar to the Wall and Messaging plugin each peer executes the test case 5 times. The results are similar as well. The staggered distribution rises to 120 executed searches in contrast to the expected 270 as in the equal distribution.

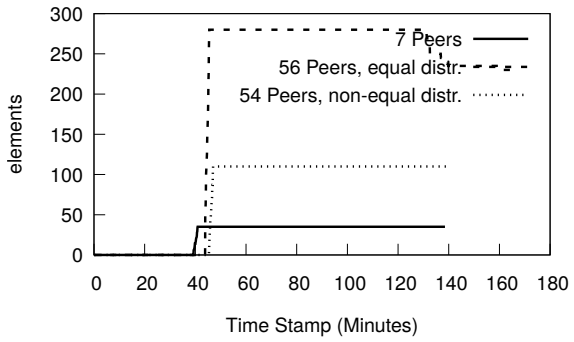


Figure 5.13.: Number of executed Searches

Calendar - View, Create, Edit and Delete

Figure 5.14 shows the number of Calender Views. From minute 43 several test cases involving the Calender plugin run. First, the viewCalendar test case, followed by createCalendar, editCalendar, and deleteCalendar test cases. The metric only shows the number of vies of the Calendar. It is not clear how many times the view functions runs during creation, deletion, and editing.

Nevertheless, we can expect at least one view per test case, making a minimum of ten views during all four tests. Therefore we expect a minimum of 70 vies for the 7 peers setup and 500 vies for the 50 peers setups. The 7 peer setup meets this assumption as well as the 50 peers equal distribution. The staggered distribution falls below the expected minimum with a total of 350 views.

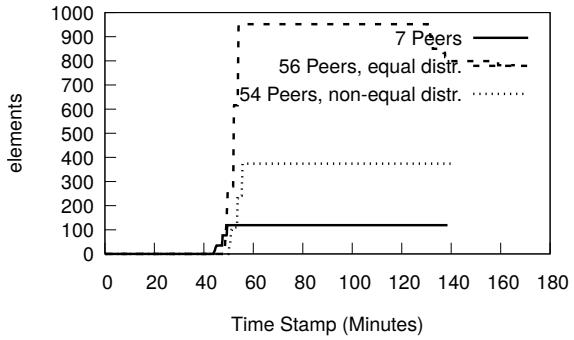


Figure 5.14.: Number of Calendar Views

Voting - Delete

For the voting plugin, the test case focuses on deletion of votes. The test cases are scheduled at minute 65. Each peer deletes 2 votes making it an expected 14 deleted votes for the 7 peers setup and 100

votes for 50 peers. As in the previous paragraphs, the 7 peers setup and equal distribution setup meet the requirements while the staggered setup does not and finishes the test with 40+ deletions.

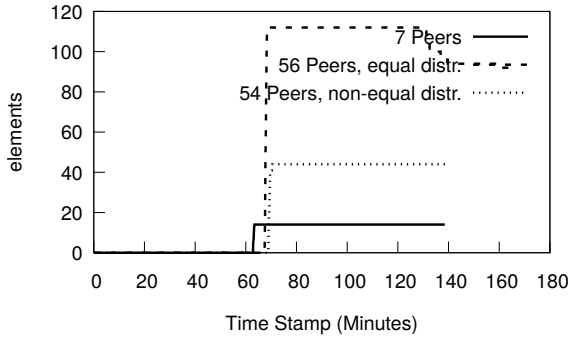


Figure 5.15.: Voting - number of Deletions

Photos - Uploads

Figure 5.16 shows the number of uploaded photos for all three experiments. The test case uploadPhoto runs at minute 70 for 10 minutes. During this time each slave node creates 5 albums and uploads 50 pictures randomly to the 5 albums. The expected number of photos for the 7 peer setup is 350. The number we observe is higher. The same observation applies to the 50+ peer setup with equal distribution. We expect 2500* uploaded photos and observe a number close to 3000. It may be that a photo album also counts in this metric. This means a total of 55 photos is created during the test case. This explains the higher numbers.

A different observation is the second rise in numbers of photos around minute 90. The timed test plan states a deletion of photos 5 times in 2 minutes as well as a deletion of 1 photo album. The expected decrease in the total number of photos doesn't take place. Instead, the number rises. Nevertheless, the absolute amount seems to correlate with the expected decrease. For the 50* peers setup we expect a 300 photos decrease and observe the same number of increase. This applies to the 7 peers setup where a decrease of 42 is what we expect and an increase of 42 what we observe. An explanation may be a wrong sign in the metric's calculation. This needs further investigation. Similar to previous observations the staggered setup doesn't show the expected behavior.

Files - Uploads

The test case uploadFileInFolder checks the Filestorage plugin. Each slave node uploads 50 files at minute 120. We expect a total number of 350 files for the small setup and 2500 files for the large

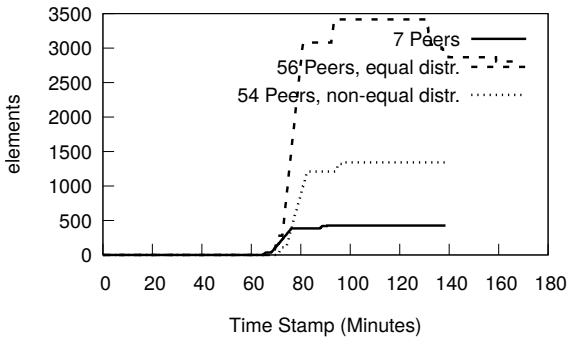


Figure 5.16.: Photos - number of Uploads

setup. The plugin meets these requirements perfectly as shown in Figure 5.17. For this, we do not take the staggered setup into account because it also shows outlying behavior for previous metrics.

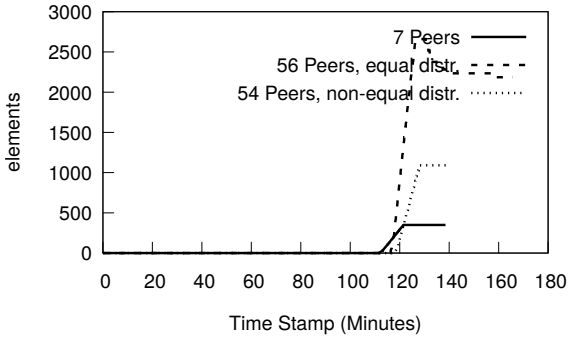


Figure 5.17.: Files - number of Uploads

5.1.4. Mapping of Plugins, Metrics and Test Cases

Plugin Name	Available Metrics	Test Cases
Calendar	views_count	createCalendar editCalendar deleteCalendar viewCalendar
Filestorage	stored_files_count	deleteFileFromFolder uploadFileInFolder
Forum	created_posts_count created_threads_count thread_views_count views_count	commentForumThread createForumThread viewForum viewForumThread
Friends	friendship_request_count	sendFriendRequestInOrder
Groups	added_files_count created_folders_count created_public_group_count deleted_items_count folder_views_count group_joins_count group_leaves_count group_views_count	createGroup leaveGroup viewGroup
Login	registrations_count	logIn logOut
Messaging	deleted_messages_count inbox_views_count sent_messages_count outbox_views_count	deleteInboxMessage sendMessage viewInbox viewOutbox
Photos	created_albums_count view_albums_count	createAlbum deleteAlbum uploadPhoto viewFriendsAlbum
Profile	basic_profiles_created_count requested_user_profiles	changeProfileInformation changeProfilePicture viewFriendsProfile viewOwnProfile
Search	searches_count	search
Voting	created_votings_count deleted_votings_count get_voting_results_count my_voting_list_views_count votes_count	createVote deleteVote vote
Wall	wall_entries_count wall_views	commentWallPost sendWallPost viewFriendsWall viewFriendsWall

Table 5.1.: Plugins, Metrics and Test Cases

As preparation for upcoming experiments, e.g. robustness, availability etc., we would like to further analyze how the existing plugins interact with the p2p-framework. We want to find out what functionality of the API they use, i.e. storage, messaging functionality or both, what data structures they need, i.e. distributed data structures or single data items. In addition, we want to evaluate the impact each of the plugins has on the p2p-framework meaning a ranking which plugin has the highest messaging impact, which of the plugins has the highest storage impact and which has a high impact on both. This information is valuable for further testing. This way it is possible in case of robustness experiments to decide which plugins to choose for a high impact on storage, messaging or both.

In order to do that we, first of all, summarize all available test cases and map them to the existing plugins. Furthermore, we list all available metrics for each plugin. These include the plugin specific metrics. We require the individual plugin metrics (social metrics) to find out whether the plugins function correctly. For every plugin, the data specific metrics of the storage-dispatcher exist. They tell the amount and rate of data every single plugin uses.

For the calculation of messaging, we require the message-dispatcher metrics. Here it is crucial to know the exact time frame of the test case execution to deduce the overall amount of messages needed for a specific test case, as well as the message rate during this test case.

Table 5.1 shows the mapping of the plugins to their respective test cases and available metrics. Most of the plugins have in common a create and a view test case. This observation applies to the metrics as well. The majority of the plugin's metrics observe the number of views and creations.

In some cases, the number of available test cases is higher as the number of metrics. The wall or calendar plugin both have four test cases and one or two metrics to observe these. The other way applies as well. The groups plugin has 8 metrics for three test cases. Therefore, for a detailed view on a single plugin functionality consider the choice of appropriate metrics in case there are many or consider the interpretation of observations in case there is only one metric for several test cases.

5.1.5. Plugin Analysis and Categorization

	plugins	sys.data.stored [KB]	sys.data.retrieved [KB]	sys.data.rate.down [KBps]	sys.data.rate.up [KBps]
1	messaging	5419	916	77061	11438
2	filestorage	763401	7314	2117369	27920
3	groups	36471	104544	105856	276480
4	photos	13389	143008	33853	730270
5	voting	631	6384	7200	36925
6	forum	703	15803	8750	66052

Table 5.2.: Storage-Dispatcher Table

	plugins	#sys.msg.total	sys.msg.rate [msg/s]	#peer.msg.total	peer.msg.rate [msg/s]
1	filestorage	86571	332	1406	4
2	forum	15462	254	488	3
3	groups	33856	332	1129	4
4	messaging	13489	233	213	3
5	photos	12968	103	144	1
6	voting	15963	398	207	6

Table 5.3.: Message-Dispatcher Table

Next, we take a look at the metrics of storage-dispatcher and message-dispatcher to organize the plugins according to their messaging and storage consumption. Please note, the analyzing does not list all available plugins (12) but an excerpt (6). The choice for the following six plugins is based on their storage and messaging requirements, these six plugins led to the top six highest peaks of the messaging rate. In specific, the Filestorage, Voting, Forum, and Groups plugins. In addition, the Photos and Messaging plugins are part of the list due to high storage impact of the Photos plugin. the Messaging plugin is important because although the Voting and Groups plugins require heavy messaging they also have a high impact on storage. They both better fit in the mixed-impact category while the Messaging plugin focuses most on the messaging functionality of the p2p-framework. Tables 5.2 and 5.3 show the storage impact and the messaging impact of the plugins.

High Messaging Impact

For the category high messaging impact we look for plugins with the highest amount of messages sent while keeping a low storage ratio. The Forum and Messaging plugins fit this category best. The

Messaging plugin has the fourth highest number of 13K messages during the experiment while having one of the lowest total storage size 5MB stored and 900kB received. The Forum plugin requires the same in the number of messages of the Messaging plugin, 12K messages, with a total storage size of 700kB and retrieving the size of 15MB. This is system-oriented data. To get an average peer load please refer to the right part of the table.

High Storage Impact

In analogy to plugins with the high messaging impact, we now look for plugins with high storage while keeping overall messaging as low as possible. Here Groups plugin fits best. With a system-wide storage of 36MB and retrieved size of 104MB, it has the second highest storage requirement in the chosen plugins. The total message amount of 6K is the lowest ranking regarding messaging. The second plugin we chose for this category is the Photos plugin. It requires a total of 13MB of storage and 143MB for retrieval while keeping messaging around 12K. Compared to the Filestorage plugin the load here mainly affects the retrieval functionality of the storageDispatcher.

Mixed Impact

Two plugins from the list remain for this last category. The Voting and Filestorage plugins. The Filestorage needs 763MB of system-wide storage and 7MB of retrieval with a total message number of 84K, being the highest consumer in both categories. The Voting plugin requires 600K of storage and 6MB for retrieval with a total message number of 32K. This is similar to the Messaging plugin and the choice of category can be different here.

5.1.6. Evaluation Summary

In the previous sections, we have seen the overall messaging and storage behavior of the p2p-framework, average look-up times, network load and the behavior of individual plugins in detail. We have mapped all existing plugins to their respective metrics and test cases. We took not of their impacts on the p2p-framework functionality in great detail and categorized them into three distinct groups - high messaging impact, high storage impact, and mixed impact. In short, the staggered experiment setup does not behave as expected. A different number of LibreSocial applications per physical machine does affect simulation results. The plugins behave as expected. There may be minor inaccuracies in the social metrics so we recommend looking into the metrics closely when analyzing the plugin's behavior.

The diagrams show slight shifts between the three experiment setups. For further testing we shall use a padding of 10 minutes for all peers to join the network, 10 minutes to send out friend-requests iteratively and finally the timed test plan execution, that will start at minute 20 of the monitoring. In addition, for further testing, we choose a different amount of peers, i.e. 64, to allow easier scaling for further experiments, e.g. 128, 256 and so on.

5.2. Availability

This section presents the evaluation results regarding availability of the p2p-framework. First of all, we recap the metric to measure availability that we introduced in chapter 3 Next, we describe the scenarios we used to measure availability. In short, we introduce three setups that vary the number of repetitions of the test cases in an increasing manner. Last, we show how the increasing number of test cases affects the availability of the framework and present measurement results for the three setups.

5.2.1. Availability Metric

Kwok [20] defines availability as the probability of a successful file transfer.

$$p_A = \frac{\#successfullTransfers}{\#requestedTransfers} \tag{5.1}$$

Equation 5.1 shows how we can calculate the probability by dividing the number of successful transfers by all requested transfers.

We extend this ratio that focuses on file transfers to apply it to the p2p-framework as follows: On the application level, i.e. using plugins that build the LibreSocial application, we observe the ratio of successful actions divided by the number of initiated actions. To stay as close as possible to the metric definition we first consider plugins with a high storage impact, as discussed in chapter 5. These plugins are the photos and groups plugins. Our attention is on actions that refer to file transfers, such as upload-photo for the photos plugin and upload-file-in-folder for the groups plugin. To get a broader overview we add the remaining actions to our measurements.

To observe the correct values we refer to table 5.1 in chapter 5 for the right choice of SkyEye.KOM metrics. In case of the photos plugin these are created-albums and view-albums metrics and for the groups plugin, we have several, from which we choose created-folders and added-files to be the most important.

Another aspect we take into consideration when analyzing the results is the duration of each action. Therefore, for each action, we also list the time frame of its execution that we extract from the scenario file of the experiment. More details on the scenario follow in the next section.

To sum up, for availability we consider ratios of executed actions to number of initiated actions, e.g. for the photos and groups plugins:

$$availability_{photos} = \frac{\#uploadedphotos}{\#initiateduploads} \tag{5.2}$$

$$availability_{groups} = \frac{\#uploadedfiles}{\#initiateduploads} \tag{5.3}$$

We consider three scenarios to measure availability. Their structure stays the same and results from considerations we gained after regression testing in chapter 5. In specific we reorganize the test cases of the original scenario, leave out several plugins and introduce a padding before starting action execution.

The resulting scenario has four parts. During the first part, that takes 20 minutes, nodes join the network and send out and accept friend requests. The second part of the scenario takes 20 minutes and focuses on messaging. In measuring availability we may consider the availability of messaging as well but first, we focus on the availability of storage. The third part of the scenario takes 60 minutes and focuses on storage. For measuring storage availability this time frame builds the basis for our analysis. The fourth and last part takes 40 minutes and focuses on mixed storage and messaging cases. Listing 5.1 gives a shortened overview of the scenario.

Listing 5.1: availability-scenario

```
# ...
# STORAGE impact
# ---------------------
# PHOTO plugin
#
4x TestCaseCreatePhotoAlbum 2m
1x Wait 2m
32x TestCaseUploadPhoto 8m
# ...
# ------------------------------------ 32 min.
#
# GROUPS plugin
#
1x TestCaseCreateGroup 2m
```

```
1x Wait 2m
8x TestCaseCreateFolder 2m
# ...
# ———————————————————————— 58 min.
#
# ...
```

To find out significant results regarding availability we run three tests. The parameters that we vary during these tests are the number of action repetitions. The first test uses values from the regression test in chapter 5 as reference rounding down to the next smallest power of 2. In the result overview, we name this scenario A and expect it to deliver availability close to 100%. The second test that we name B doubles each repetition value keeping the time frames and the third test C doubles the repetitions once more.

We want to find out if this increase affects the availability of the p2p-framework and how strong this effect is. When effects occur it is possible to get further insight into the performance of the p2p-framework by observing the storage-dispatcher metrics for the specified time frame. The detailed discussion of measurements follows in the next section.

5.2.2. Measurement Results

photos								
action	time	metric	A	B	C	avail.A	avail.B	avail.C
create-photo-album	40-42m	created-albums	4	8	16	100%	100%	81%
upload-photo	44-52m	created-albums	32	64	128	100%	70%	34%
view-own-albums	54-56m	view-own-albums	4	8	16	100%	100%	88%
delete-photo	66-68m	created-albums	4	8	16	100%	100%	63%
						Σ 100%	Σ 92.5%	Σ 66.5%
groups								
create-group	70-72m	create-public-group	1	2	4	100%	100%	100%
create-folder	74-76m	create-folder	8	16	32	88%	63%	31%
upload-file	78-86m	add-files	32	64	128	94%	41%	28%
delete-file	88-90m	deleted-item	4	8	16	75%	NA	50%
view-group	92-94m	view-group	1	2	4	200%	0%	250%
						Σ 89.25%	Σ 68%	Σ 52.25%

Table 5.4.: Storage Availability Results

messaging								
action	time	metric	A	B	C	avail.A	avail.B	avail.C
send-message	20-22m	send-msg	8	16	32	100%	94%	50%
view-outbox	23-25m	view-outbox	4	8	16	100%	100%	56%
view-inbox	26-28m	view-inbox	4	8	16	100%	100%	54%
						Σ 100%	Σ 98%	Σ 54%
forum								
create-thread	30-32m	create-thread	4	8	16	25%	0%	0%
comment-thread	33-35m	created-posts	4	8	16	100%	100%	56%
view-thread	36-38m	view-thread	4	8	16	100%	100%	56%
						Σ 100%	Σ 100%	Σ 56%

Table 5.5.: Messaging Availability Results

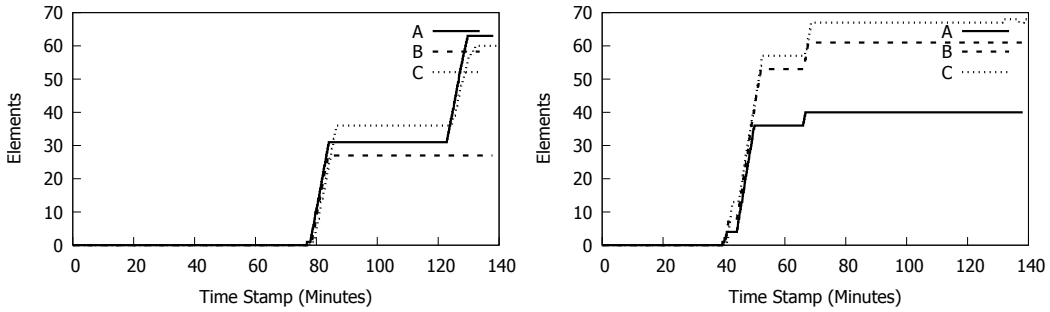
Table 5.4 summarizes the results for storage availability measurements. First of all, we see a list of plugins we considered, its actions executed, the time frame of execution and metric that we used to track the right value. For more details on each column please refer to the previous section. Next, we include three columns listing the desired number of action execution. They are part of the scenario and A, B, and C denote the three setups with increasing repetition number. The last there columns present availability values per action and a summarized availability value per plugin.

As we expected, test case A shows availability values close to 100%. In case of the photos plugin, it is in fact 100%. Significant deviations show the create-folder action of the groups plugin with 88% availability as well as the deletFileFromFolder action with 75% availability. In addition, we note that the view-folder metric provides an incorrect value of 2 views, where only 1 is set in the scenario. That is, why we do not take this value into account when calculating the overall availability of the groups plugin.

Next, we take a look at the overall trend of availability. As expected, the increasing number of repetitions does affect the availability of the plugins. During the highest number of repetitions in test run C availability of the photos plugin drops to 67% and that of the groups plugin to 52% with upload-photo and upload-file-in-folder being the least available action with 34% and 28%.

Figure 5.18 visualizes this effect for the two actions upload-photo and upload-file-in-folder. What we observe is a saturation of the photo upload around 44 during a time frame of 8 minutes and a saturation of file upload around 30 during the same time frame.

Table 5.5 shows further results. Here, we lay our focus on the availability of the messaging capability of the p2p-framework. Our plugins of choice here are the messaging plugin and the forum plugin. Similar to the availability of storage we start with a configuration that is known to provide



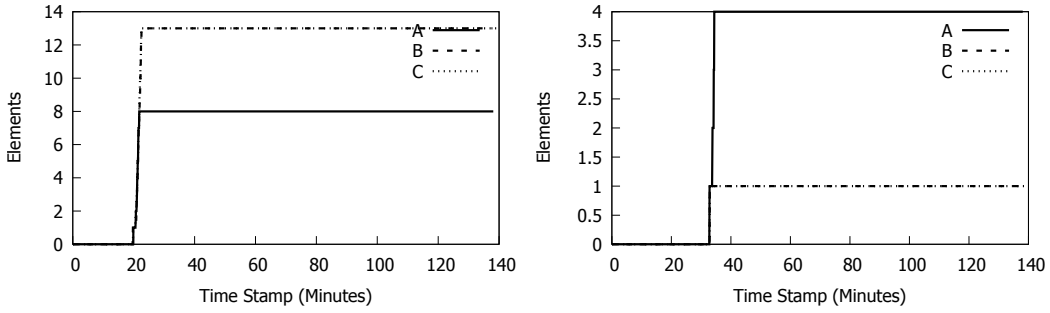
(a) Uploaded Files Saturation (b) Uploaded Photos Saturation

Figure 5.18.: Saturation of uploaded Files and Photos

high availability as shown during regression testing in 5. We proceed by increasing the number of repetitions.

We observe that messaging is also affected by the number of repetitions. With an increasing number of repetitions the availability decreases and drops to 54% for the messaging plugin in the C scenario and 56% for the forum plugin.

The actions most affected by the repetition number are the send-message acton for the messaging plugin and both comment-thread and view-thread actions for the forum plugin. We can see that the send-message action already responds to the B scenario with an availability of 94% whereas other actions still provide 100% availability.



(a) Sent Messages Saturation (b) Created Posts Saturation

Figure 5.19.: Saturation of Sent Messages and Created Posts

Similar to storage availability figure 5.19 summarizes the saturation of messaging for both plugins. Comparing overall availability we conclude that for the given set of parameters storage functionality provides lower availability than messaging. Having said that we ask to take both, the number of repetitions and the duration, into consideration for a deeper analysis. The number of repetitions for messaging is lower, at the same time, the time frames for execution are shorter.

These observations indicate a starting point for further analysis of the underlying layers. One possibility for further examination is the storage/retrieval rate of the storage-dispatcher on p2p-framework layer during these time frames.

5.3. Scalability

This section focuses on the scalability of the p2p-framework. First, we recap the metric we use throughout this section to measure scalability. We then in detail describe the scenario, i.e. which actions the peers perform. Furthermore, we introduce two models for the population growth, that is required to measure scalability according to the metric we chose. Finally, we present our measurement results, discuss them with reference to the preceding availability results and compare the results of both population growth models.

5.3.1. Scalability Metric

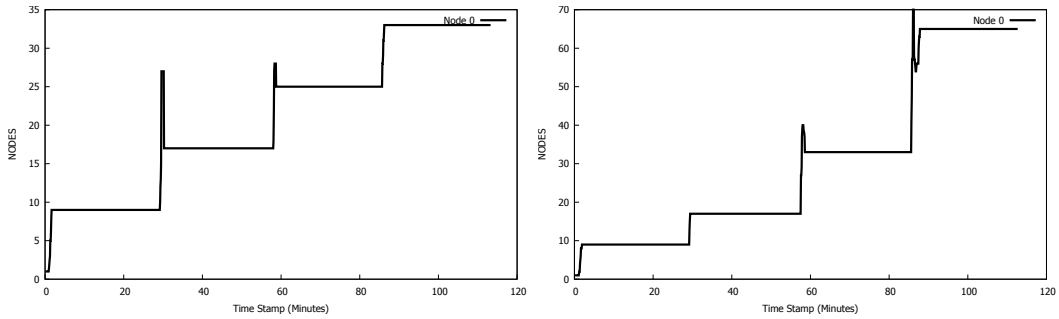
Scalability, as defined by Kwok [20] is the probability of stable availability or download rate, given a certain population growth.

$$\mathscr{P}(p_A = 70\%)_{|population_growth=10\%} \tag{5.4}$$

Equation 5.4 shows how we calculate the scalability. As reference values for an expected availability we use our results from section 5.2 that focuses on availability, i.e. we examine how storage and messaging of the p2p-framework respond to population growth and then compare it to the availability we expect in a system with a fixed population. We reduce the number and variety of test cases for an easier overview of the results. Therefore we choose the upload-photo action of the photos plugin and its according metric created-albums to evaluate how the storage functionality is affected. In addition, we choose the send-message action and its metric sent-messages of the messaging plugin to evaluate population growth effect on messaging services of the p2p-framework.

As in the availability section, we divide the number of successful actions by the number of requested actions to calculate its availability. As a second step, we compare this ratio to the availability we expect, based on the previous measurement in chapter 5.2. The next paragraph describes two population growth models we use for testing scalability.

5.3.2. Population Growth



(a) Linear (b) Exponential

Figure 5.20.: Population Growth

We implement two models of population growth for scalability testing - a linear model and an exponential model. In the linear model, each time frame 8 nodes join the network. We continue observations for four time frames, each being 30 minutes long so that we acquire a total number of 32 peers at the end of this test. In the exponential model the population doubles in each time frame. We start with 8 peers in the first time frame, continue with four consecutive time frames and finish with 64 peers at the end of the test. We keep the number of time frames the same to be able to better compare the two models. Figure 5.20 shows the node count of peers monitored by SkyEye.KOM. We see how every 30 minutes the population changes according to the specified model. The next paragraph describes the scenario for these setups.

5.3.3. Scenario for Scalability

A challenge that occurs when testing scalability is the implementation of population growth. We choose the following method. First of all, we define a set of actions that all available peers in the given time frame perform. This internal scenario is presented in figure 5.2.

Listing 5.2: internal-scenario

```
# MESSAGING impact
# ----------------------
8x TestCaseSendMessage 2m
1x Wait 1m
# ...
#
# STORAGE impact
# ----------------------
```

```
32x TestCaseUploadPhoto 8m
1x Wait 1m
# ...
```

Its total duration is limited to 20 minutes and it is meant to be repeated several times. Next, we define a global scenario which includes the population growth. Figure 5.3 shows an excerpt of the global scenario. Here we define how many peers join each time frame, i.e. 8 in the linear model and 8 times the power of 2 in the exponential model. This approach requires manual intervention every 30 minutes to add new peers and initiate the local scenario. We use the scripts for remote execution presented in chapter 4 and describe how this step can be further improved in chapter 7.

Listing 5.3: global-scenario

```
0m      8 nodes join
5m      nodes send friend requests
10m     nodes start local scenario
15m
20m
25m
30m     8 (16) nodes join
35m     nodes send friend requests
40m     nodes start local scenario
45m
50m
#       ...
```

In summary, we have two population growth models - a linear and an exponential model. Furthermore, we vary the number of repetitions of the actions. We distinguish two sets of repetitions, A and C and base them according to the number of repetitions in the availability testing. Scenario B drops out. This way we can refer to our result sets of availability and keep the number of total scalability tests to four - linear growth with repetition number A, linear growth with repetition number C, exponential growth with repetition number A, and exponential growth with repetition number C. We code these four test runs according to their first letters - LA, LC, EA and EC for easier reference. The next paragraph describes the outcome of the four test runs.

5.3.4. Measurement Results

LA			
#peers	minute	storage.avail	message.avail
8	0-30m	100%	100%
16	30-60m	94%	38%
24	60-90m	63%	63%
32	90-120m	35%	100%
		Σ 73%	Σ 75%
ref. to exp. avail.		Π 73%	Π 84%

LC			
#peers	minute	storage.avail	message.avail
8	0-30m	34%	34%
16	30-60m	35%	47%
24	60-90m	20%	41%
32	90-120m	6%	44%
		Σ 24%	Σ 42%
ref. to exp. avail.		Π 71%	Π 84%

Table 5.6.: Linear Population Growth

EA			
#peers	minute	storage.avail	message.avail
8	0-30m	91%	100%
16	30-60m	91%	100%
24	60-90m	78%	50%
32	90-120m	84%	75%
		Σ 86%	Σ 81%
ref. to exp. avail.		Π 86%	Π 81%

EC			
#peers	minute	storage.avail	message.avail
8	0-30m	35%	47%
16	30-60m	26%	47%
24	60-90m	30%	34%
32	90-120m	28%	34%
		Σ 30%	Σ 40%
ref. to exp. avail.		Π 88%	Π 80%

Table 5.7.: Exponential Population Growth

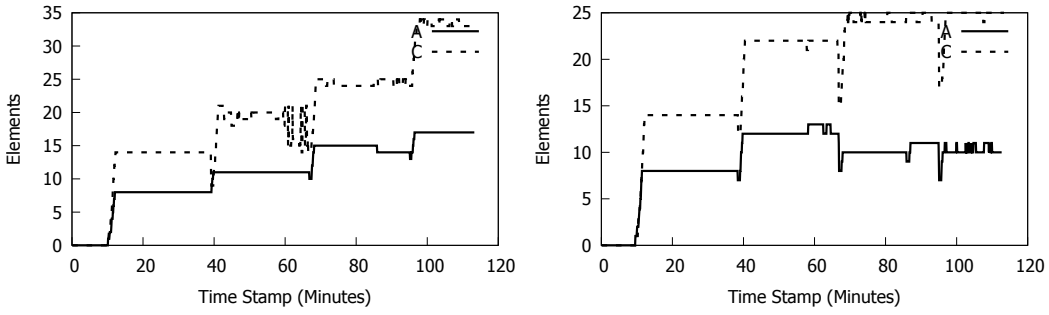
Tables 5.6 and 5.7 show our measurement results for scalability. Both tables divide into two parts - A for the low number of action repetitions and C for the high number of action repetitions. The exact

repetition number is in table 5.4 in the availability section 5.2. We observe the upload-photo action and send-message action to deduct the effect of population growth on the p2p-framework.

For the linear population growth, we observe that storage availability decreases with rising number of peers. Starting with 100% it decreases to 35% when 32 peers participate in scenario A. This tendency is also visible for scenario C where availability of storage decreases from 34% down to 6% at the end of the last time frame. Although the overall availability of 73% in scenario A and 24% in scenario C is low we also compare it to our reference value of the send-message and upload-photo action in table 5.5. From this comparison, we see that the overall value matches what we can expect as availability for scenario A and C. Storage reaches 73% of expected availability given linear population growth.

For messaging no such trend is visible. It does not seem to be affected by linear growth. The values vary between 100%, 38%, and 63% for scenario A and keep close to 40% for scenario C. Again, comparing it to reference values, we have a stable messaging availability of 84% during linear population growth. At this point, we conclude, that messaging scales up during linear population growth while storage does not. We also refer to chapter 6 which contains more details on storage-dispatcher and message-dispatcher performance during scalability analysis.

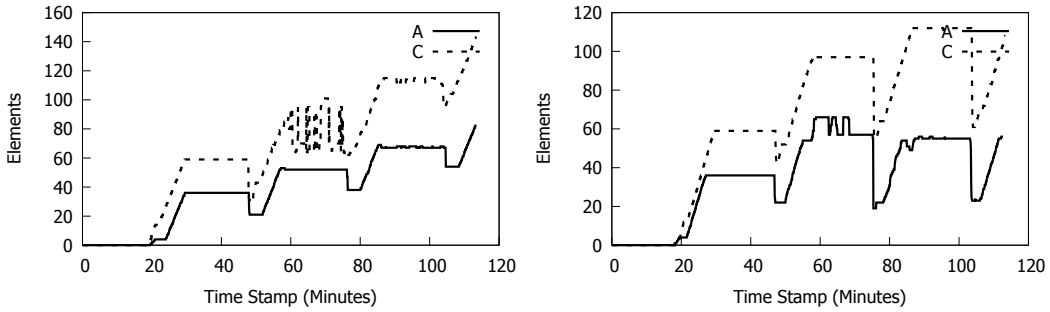
Next, we consider exponential population growth. The overall picture differs from linear growth. Both, messaging and storage do not show any trend of increasing or decreasing availability during exponential growth. In fact, the values for storage vary around 86% during scenario A and 30% during scenario C. This is very close to our reference values. The same applies to messaging. It shows variance around 80% of availability during scenario A and 40% during scenario C. Both, messaging and storage reach more than 80% of the reference values. This leads us to the conclusion that the p2p-framework scales up when exponential growth occurs.



(a) Linear Growth (b) Exponential Growth

Figure 5.21.: Messaging Scalability

Figures 5.21 and 5.22 summarize scalability results in a diagram form. We can see messaging and storage scalability behavior for the scenarios A and B.



(a) Linear Growth (b) Exponential Growth

Figure 5.22.: Storage Scalability

When we compare linear and exponential growth this seems to be a surprising result. A distributed system that scales up while exponential growth occurs should also be able to scale up during the linear growth of population.

This question may be answered more appropriate with a considerable higher amount of participating nodes. Therefore we postpone further examination and refer to High-Performance computing in section 7 as a means of testing systems with up to 128 nodes. Furthermore, the answer can be found in the algorithmic complexity of the Pastry overlay, that scales logarithmically in the number of nodes. Therefore a shift in the availability may occur in a linear growth as well, as soon as it reaches a total of 64 peers.

5.4. Robustness

This section focuses on the robustness of the p2p-framework. First of all, we reconsider the metric to evaluate robustness. Next, we introduce two models for peer dynamics, varying the percentage of churn. Then, we give details on the implementation of peer dynamics and local peer activity. Last, we present measurement results for robustness.

5.4.1. Robustness Metric

We refer to Kwok's [20] definition of robustness.

$$\mathscr{P}(p_A \leq 90\%)_{|peer_dynamics=model_A} \tag{5.5}$$

It defines robustness in terms of reaching an average availability or download rate given certain peer dynamics. Therefore, similar to scalability, we refer to measurements of availability when evaluating robustness. In addition, the calculation requires peer dynamics as a side condition. The following paragraph introduces our model for peer dynamics.

5.4.2. Peer Dynamics

We want to find out how storage and messaging functionality of the p2p-framework are affected by peer dynamics. We define peer dynamics as a percentage of peers leaving the network in a given time frame. At the same time, the same percentage of new peers joins the network, so that a rotation of peers carrying the system takes place. We expect the overall number of peers to stay constant while the rotation occurs.

The first model of peer dynamics introduces a light churn model. Churn is a term that describes peers in a network that go offline and come online again. Given 48 online peers, 8 peers leave and 8 new peers join, making the percentage of churn 16.5%. The second model keeps the number of online peers at 48. The number of peers leaving and joining rises to 16, so does the percentage, that rises to 33%. The next section describes how this peer dynamic is implemented.

5.4.3. Scenario of Robustness

Similar to scalability we use two scenarios to implement peer dynamics and measure robustness. The local scenario is equal to the scalability scenario. We simplify the restructured scenario used for availability, reducing it to one action that involves the storage and another that involves messaging 5.2. We change the global scenario that contains the peer dynamic. In addition to nodes joining the network, as was the case for scalability, nodes also leave the network

Listing 5.4: global-scenario-robustness

```
0m      48 nodes join
5m      nodes send friend requests
10m     nodes start local scenario
15m
20m
25m
30m     8 (16) nodes leave; 8 (16) nodes join
35m     nodes send friend requests
40m     nodes start local scenario
```

45m
50m
. . .

Figure 5.4 shows an excerpt of the global scenario. We manually intervene here, to accomplish the peer dynamics. There are test plugin commands that make peers log out and leave the network. At the current state, there are no commands available to make peers login at a given time. In addition, the method used in this scenario uses new peers to join the network. This means the peers are not registered users that come online again, but new peers joining for the first time. Also, peers that leave the network do not inform the network. This means that they do not log out in a way that is supported by the application but rather cut off their connection to the network. This way, we focus on the robustness of storage and messaging based on a dynamic rotation of the carrying nodes. We sum up two scenarios - light churn (16%) with low action repetition number LE, and medium churn (33%) with low action repetition number ME. The next section shows results of the two test runs.

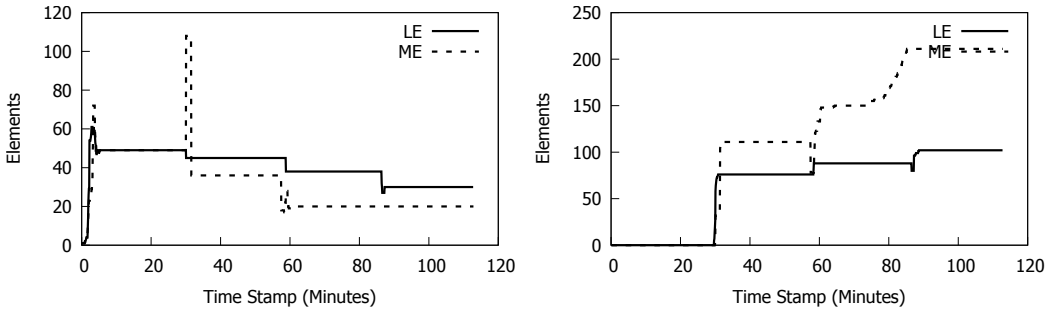
5.4.4. Measurement Results

LE				
#peers	minute	storage.avail	message.avail	#lost.msg
48/48	0-30m	91%	100%	0
45/48	30-60m	56%	100%	75
38/48	60-90m	69%	100%	10
29/48	90-120m	84%	75%	15
		Σ 75%	Σ 94%	
ref. to exp. avail.		Π 75%	Π 94%	

ME				
#peers	minute	storage.avail	message.avail	#lost.msg
48/48	0-30m	94%	100%	0
38/48	30-60m	60%	100%	112
20/48	60-90m	63%	100%	38
20/48	90-120m	0%	0%	60
		Σ 54%	Σ 75%	
ref. to exp. avail.		Π 54%	Π 75%	

Table 5.8.: Light and Medium Churn

Table 5.8 shows the outcome of robustness testing. In addition to the availability of storage and messaging, we also observe the number of online nodes and the number of lost messages. In addition to the tables, figure 5.23 shows the number of online nodes and the number of lost messages.

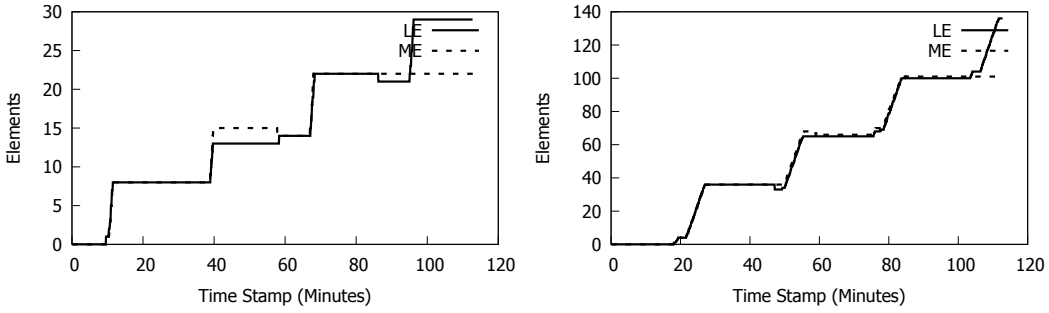


(a) Nodes Count (b) Lost Messages

Figure 5.23.: Light (LE) and Medium (ME) Churn

We expect the number of online nodes to stay constant at 48 because the same amount of peers joins the network as the amount that leaves. This is not the case. The number of online nodes drops for both peer dynamics from 48 to 29 in case of light churn and from 48 to 20 in case of medium churn. The explanation at this point remains open. We recommend prolonging the phase between nodes leaving the network and nodes joining the network to allow the overlay to refresh routing tables and the p2p-framework to refresh the list of online users. In addition, we recommend comparing the outcome with a scenario, in which the nodes follow the application protocol and log out before disconnecting off the network.

In addition to the number of online nodes dropping we observe a rising number of lost messages. For light churn, this number rises by 75 when the first rotation of peers occurs and rises by 10 to 15 lost messages for upcoming peer rotation. For medium churn in the last time frame from 90 to 120 minutes, the number of lost messages keeps rising while the number of online nodes is at 20, which is less than half of the expected online nodes. The availability of both, messaging and storage, falls down to 0%. We interpret this stage of the test run as a system failure. Figure 5.24 summarizes messaging and storage robustness.



(a) Messaging Robustness (b) Storage Robustness

Figure 5.24.: Messaging and Storage during Light (LE) and Medium (ME) Churn

Comparing availability with reference values of the previous experiment we observe 75% of availability for storage during light churn and 94% of availability for messaging. We conclude that storage and messaging are robust given peer rotation of 16%. Further, observing availability, the number of lost messages and number of online nodes during medium churn we conclude that storage and messaging do not fulfill the robustness requirement of a rotation rate of 30%.

Throughout this chapter, we presented measurement results for availability, scalability and, robustness. Beforehand, we checked the overall functionality of the system with a regression test, restructured the scenario according to our needs, and listed available test cases and metrics. The next part of the thesis takes the overall system analysis one step further and observes the p2p-framework layer performance during the preceding experiments. In addition, it motivates directions for further benchmarking experiments.

Part III.

Summary and Outlook

Chapter 6.

Measurement Summary

6.1. Summary and Conclusion

	Exp. Outcome			Storage-Dispatcher (KBps)				Message-Dispatcher (KBps)			
Exp. Type	Strg.	Msg.	#Peers	Store-Time	Min	Avg	Max	#Lost-Msg.	Min	Avg	Max
Availability											
A	100%	100%	64	0.05s	0	3	27	0	2	28	1044
B	70%	84%	64	0.06s	0	4	51	0	2	36	877
C	34%	50%	64	0.06s	0	6	86	0	2	38	1035
Σ				0.056s	0	4.3	55	0	2	34	985
Scalability											
LA-start	100%	100%	8	0.09s	1	4	71	0	3	20	86
LA-end	35%	100%	64	0.06s	0	9	55	0	2	35	1773
LC-start	34%	34%	8	0.09s	0	5	21	0	4	31	330
LC-end	6%	44%	64	0.06s	0	15	102	0	2	24	230
EA-start	91%	100%	8	0.09s	0	3	27	0	2	21	267
EA-end	84%	75%	64	0.06s	0	0	16	0	2	34	1142
EC-start	35%	47%	8	0.08s	2	5	24	0	4	27	164
EC-end	28%	34%	64	0.06s	0	10	100	0	2	25	732
Σ				0.073s	0.4	6.4	52	0	2.6	27.1	590.5
Robustness											
LE-start	91%	100%	48	0.4s	0	3	29	0	2	32	709
LE-end	84%	75%	29	0.06s	0	2	19	102	2	26	771
ME-start	94%	100%	48	0.06s	1	3	25	0	2	49	1058
ME-end	0%	0%	20	0.06s	0	1	26	211	2	23	359
Σ				0.145s	0.25	2.3	24.8	78.3	2	32.5	724.3

Table 6.1.: Performance of Messaging and Storage on P2P-Framework Layer

In this section, we summarize our observations during benchmarking. We go through each experiment and give a brief overview of the results. Then, we take another step of the analysis. We take a close look at the p2p-framework layer by focusing on the data-rates of the storage-dispatcher and message-dispatcher. At the end of this chapter, we present the summary of overall results.

So far, we analyzed the p2p-framework from the application layer perspective. We executed tests by sending commands to plugins that build the LibreSocial application. We observed the performance focusing on availability, scalability, and robustness. Our first test was the regression test in section 5. We saw that the application LibreSocial with its plugins is fully functional. We saw minor inaccuracies in the naming of metrics and also that a few metrics do not correctly calculate the result. Next, we mapped each plugin to its observing metric and listed test-cases (or actions) that they offer. Furthermore, we organized them according to their impact on the storage and messaging service on the p2p-framework layer.

Now, we take a closer look at the performance of the p2p-framework layer. Table 6.1 shows the performance of the message-dispatcher and storage-dispatcher during each of the conducted experiments. All values shown in the table are in Kilobytes per second, except the duration, number of peers and lost messages. For each experiment, we listed the outcome in percent for both, the storage and messaging on the application layer. When the corresponding percentage goes down it indicates either the decrease of application performance or the limitation of the p2p-framework. The data-rates shown for the storage-dispatcher and message-dispatcher show us if the performance of the p2p-framework correlates with the performance on the application layer. The following sections give more detail on each of the three experiments.

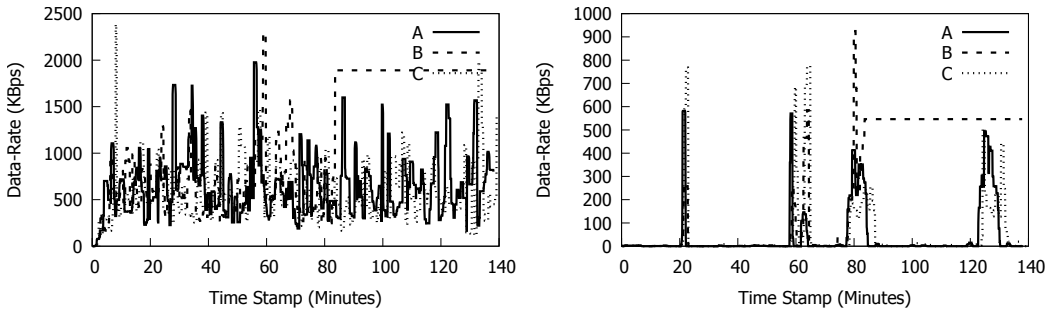
6.1.1. Availability

For availability testing, we vary the number of action repetition. For messaging and storage on the application layer, we choose two plugins each and perform their available actions. We observe, that repetition number does affect the availability of storage and messaging functionality on the application layer. Both show a saturation with rising number of repetitions and indicate a bottleneck.

When we take a look at data-rates of the underlying p2p-framework layer, we see that with rising number of storage requests the storage data-rate rises. The highest data-rate it reaches is close to 90KBps. We conclude, therefore, that during availability testing the maximum capacity of the storage service on p2p-framework layer is reached. Therefore the p2p-framework limits overall system performance.

The data-rate of messaging is not affected by the repetition of actions. It stays around 1MBps. Since

the availability of messaging on application layer does decrease, we conclude that the application layer limits overall system performance while testing availability. Figure 6.1 shows the two data-rates during availability testing.



(a) Message-Dispatcher (b) Storage-Dispatcher

Figure 6.1.: Data-Rates for Availability

6.1.2. Scalability

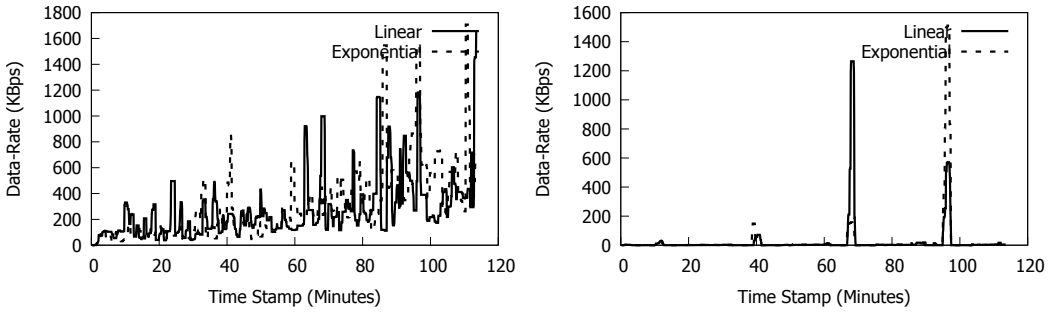
For scalability, we introduce two models of population growth, a linear and an exponential model. We observe, that linear growth affects storage on application layer for peer numbers up to 32. For the exponential growth model, we see that messaging and storage scale up.

When we look at the p2p-framework layer we see how data-rates for storage behave. They are inversely proportional to its availability on the application layer. With decreasing availability the data-rate on p2p-framework layer rises. Here, we observe a similar saturation as for availability, being limited around 100KBps. Therefore, we conclude that the p2p-framework storage service reaches its maximum around 100KBps and limits overall system performance.

For messaging, we see that high availability is proportional to high data-rates and therefore the limiting factor here is the application layer. Messaging data-rates reach up to 1.7MBps. Figure 6.2 visualizes the p2p-framework performance during scalability testing.

6.1.3. Robustness

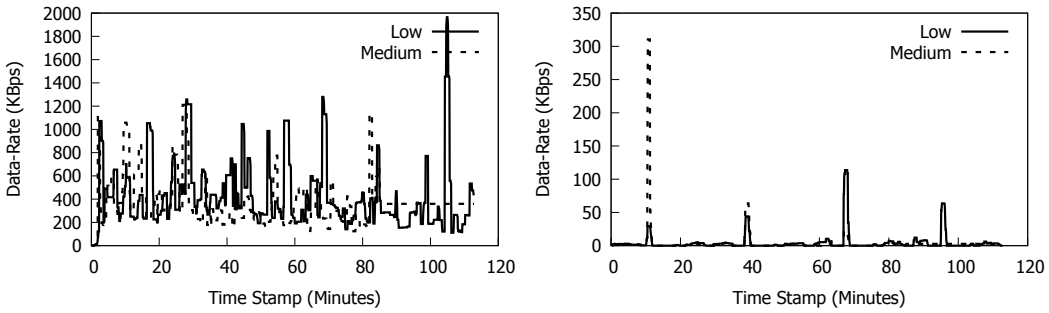
For robustness, we introduce peer dynamics. For a low churn rate, we rotate 16% of peers in the network. A medium churn rate leads to 33% of rotation. The p2p-framework is robust during the low churn rate while the overall system fails during the fourth rotation cycle of the mdium churn rate.



(a) Message-Dispatcher (b) Storage-Dispatcher

Figure 6.2.: Data-Rates for Scalability

When we take a look at the application layer, we observe that both, messaging and storage is affected by the 33% churn rate. Furthermore, robustness is the only setup showing a rise in the number of lost messages, especially during the medium churn experiment. The data-rates stay within expected boundaries with 20KBps for the storage service and 700KBps for the messaging service due to the low rate of action repetition on the application layer. Here, we cannot deduce the interaction between the application layer and p2p-framework layer, since no obvious bottleneck is present. Figure 6.3 shows the data-rates for robustness.



(a) Message-Dispatcher (b) Storage-Dispatcher

Figure 6.3.: Data-Rates for Robustness

Overall, the storage service of the p2p-framework allows data-rates up to 100KBps. The storage service scales up for exponential peer growth and it is robust against churn for up to 16%. The storage service limits overall system performance, when the number of storage requests on application level rises. The messaging service of the p2p-framework allows data-rates up to 1.7MBps. It scales up during exponential population growth and is robust for a churn rate up to 16%. Here, the application-layer leads to a performance bottleneck when overall system availability decreases.

Chapter 7.

Future Benchmarking

In this chapter, we show what options and directions benchmarking of the p2p-framework has and what challenges these options may bring.

7.1. High Performance Computing

For the scope of this thesis, we used a local area network environment to analyze properties of the p2p-framework. We can refer to this type of environment as small-scale local area setup. Its main limitations are the maximum number of physical machines, which is limited to 8, and the homogeneity of the setup. Homogeneity, in this case, means that all machines have exactly the same characteristics, such as operating system, RAM, processor speed etc. In addition, their networking connection to a Gigabit Ethernet switch is exactly the same as well. Further, the Gigabit switch itself leads to packet transmission with zero packet loss. This kind of environment, although perfect for testing, does not represent a typical environment of a p2p-system, where unreliability and heterogeneity of nodes and networking is the norm. Here we refer to the section on properties of p2p-systems 2.2. In addition, the node population can grow dramatically larger than 8 nodes or 64 logical nodes up to several million physical nodes.

High-Performance Computing tackles the limitation of the maximum peer number. The HILBERT [35] High-Performance Computer (HPC) offers up to 128 physical nodes interconnected by Gigabit Ethernet. This allows, by introducing 8 logical peers per physical node, for setups with up to 1024 total nodes. This peer number is especially interesting to make advanced studies of scalability and robustness that was limited by the maximum peer number in this thesis.

The main challenge for migrating the experimental setup to the HPC is the manual intervention required for a number of current tests. Due to the batch processing of the HPC, it is required for an

experiment to run completely autonomous after the initial start. Therefore, we propose to focus on the communication with the OSGi console of the p2p-framework. It is possible, at the moment, to execute commands within this OSGi console without using a graphical user interface. In addition, the master node has a unique set of commands to initiate the friend requests or to start local scenarios. During this thesis' work, we used the operating system's capability of interprocess communication with pipes [32] and redirected output of the OSGi console. We also succeeded to redirect the return keystroke to the OSGi console. Nevertheless, it was not possible to direct the send-friend-request or the start-timed-testplan command from a shell to the OSGi console of the master node.

A different approach is described in [36]. Here the authors introduce a set of OSGi bundles that allow for ssh connection to the existing OSGi console. We have not yet followed this proposal and it seems to be a promising option.

Last but not least, the HPC offers an interactive mode for developers. This way it is possible to reserve a single physical node for up to 3 hours and interact with the reserved node through a shell. This is a helpful option to gather and experiment with commands to set up the master node, find out its dynamically allocated IP address, make this IP address available to further slave nodes etc. More information about the developer mode can be found here [35].

In short, HPC tackles the maximum peer number limitation and offers a large scale local area network testbed for up to 1024 peers. The next section offers a solution for the homogeneity limitation introducing a realistic internet-based test bed.

7.2. Small and Large Scale Internet Setups

Two options for testing a realistic setup exist - a small scale and a large scale internet setup. Previous work [34] on testing the p2p-framework in a small scale internet setup has been successfully done and it is possible to set up a publicly accessible machine at the university campus to act as a bootstrap node. This way, further nodes can join either from the university or other locations. Although the number of nodes in such a setup may be small, such as three to five nodes, this setup introduces new characteristics and challenges that simply are not present in a local area network setup.

First of all, it introduces heterogeneity of nodes and network connections. Second, network address translation (NAT) for nodes behind a firewall can be tested. At the moment a solution for NAT is implemented in the p2p-framework and is not benchmarked in this thesis. It is also possible to test this specific functionality within the pc-lab by using one machine as a network address translation router, connecting another participant to the Gigabit switch. To reach a higher number of participants

we propose to design an exercise for the “P2P-Systems” course, where students run an instance of the p2p-framework on their private machines. This way aspects such as distributed monitoring, NAT traversal, and others can be studied in a practical way and measurements gathered.

A different approach for large-scale internet tests is the open source research platform planetlab.org [37]. It offers the possibility to test a distributed application on servers that are located in different parts of the globe. This way, both heterogeneity of participants and a large number of participants is guaranteed. Although the setup may be time-consuming these tests may provide valuable insight on the performance of the p2p-framework in a realistic large-scale scenario. Before addressing this step we recommend to examine NAT traversal in the pc-lab or small-scale internet setup.

In the preceding chapters, we gave additional insight into the performance of the p2p-framework. We focused on data-rates of the two main components of the p2p-framework, the message-dispatcher, and the storage-dispatcher. In addition, we proposed two directions for further benchmarking - high-performance computing and large-scale internet setup. We close this thesis by providing a template for future benchmarking results. It currently shows extended measurements of the availability testing and can easily switch to the scalability or robustness results.

Part IV.

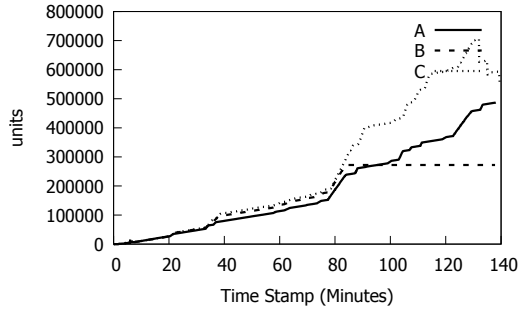
Appendix

Appendix A - Detailed Metric Overview for Availability Benchmarking

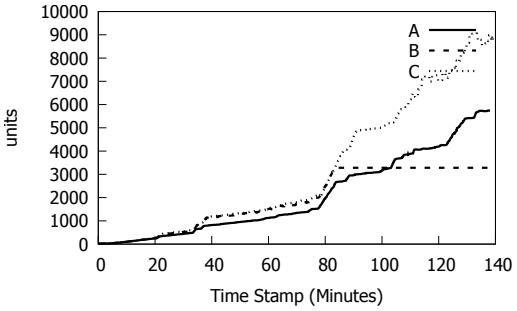
The following pages show an extract of the available statistical data of the availability experiment in a diagram format. This template is organized bottom-up. First, it shows measurement results of the p2p-framework layer, focusing on the core components - the message-dispatcher and the storage-dispatcher. Next, it shows data of the application layer. It goes through the plugins, categorized by messaging impact, storage impact and mixed impact. In addition to the actions performed by the plugin during the test, it shows additional storage information, specific to each plugin. The template, at the moment, is configured to show results of the availability experiment. It can change to show scalability, robustness, and results of regression testing.

7.3. Message Dispatcher

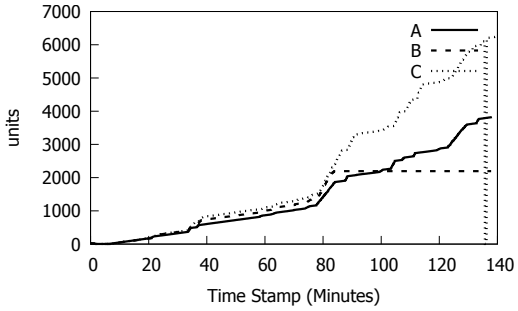
7.3.1. Sent Messages



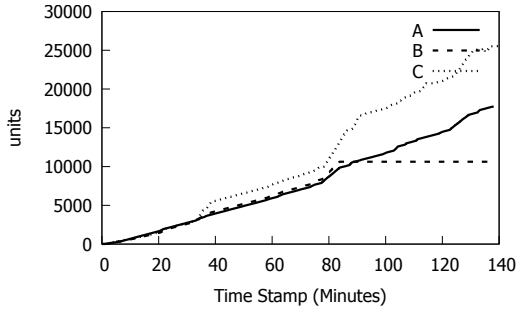
(a) Sent Messages



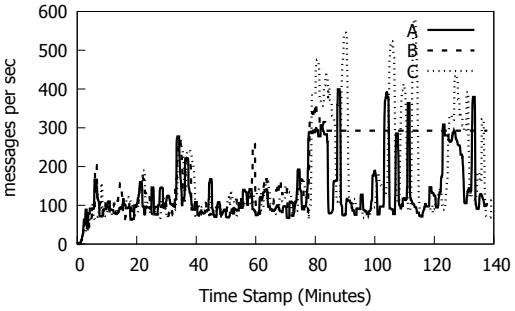
(b) Sent Messages - Average



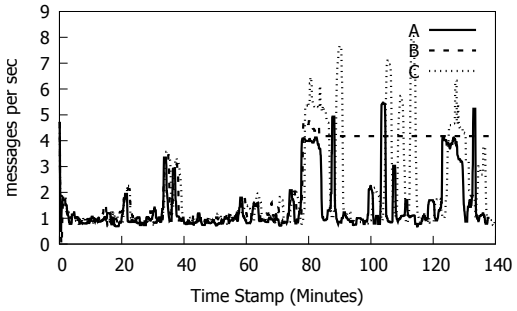
(c) Sent Messages - Minimum



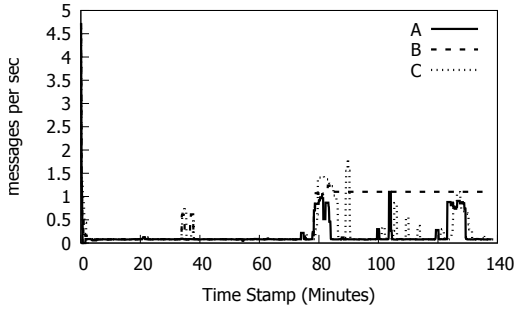
(d) Sent Messages - Maximum



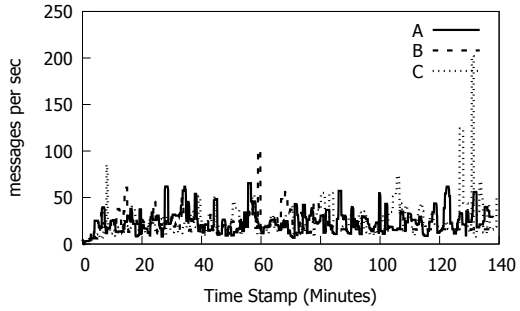
(e) Sent Messages - Rate



(f) Sent Messages - Average Rate



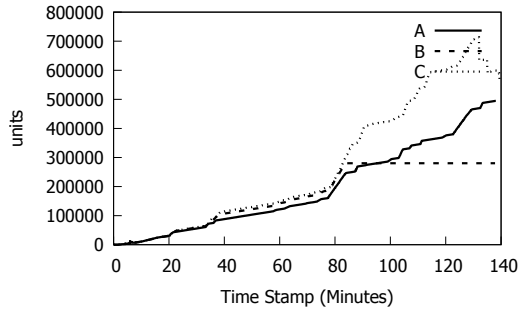
(g) Sent Messages - Minimum Rate



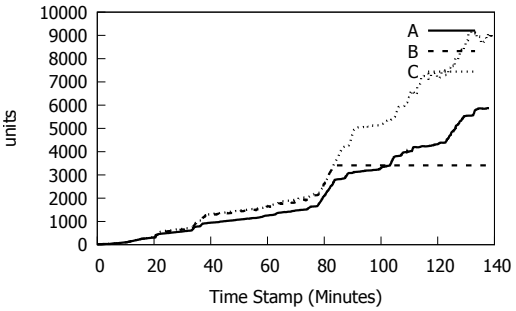
(h) Sent Messages - Maximum Rate

Figure 7.1.: MessageDispatcher - Sent Messages (Count, Rate)

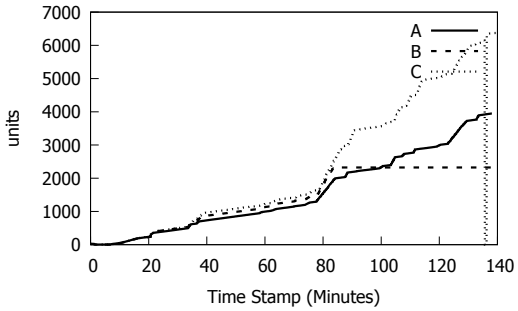
7.3.2. Received Messages



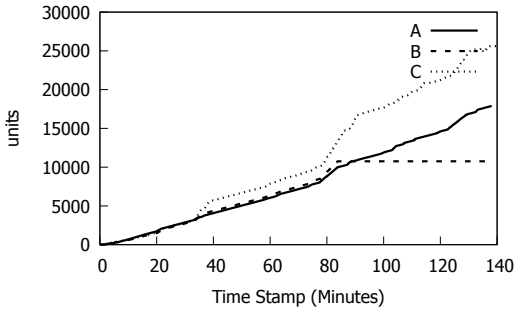
(a) Received Messages



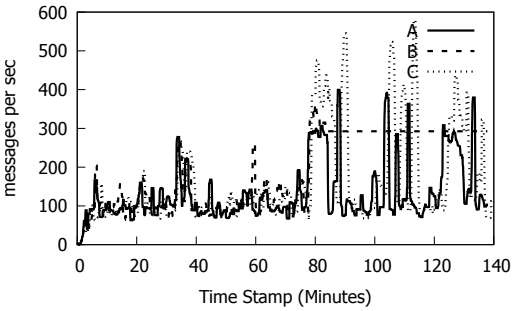
(b) Received Messages - Average



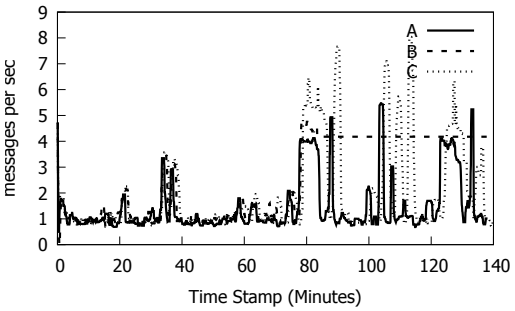
(c) Received Messages - Minimum



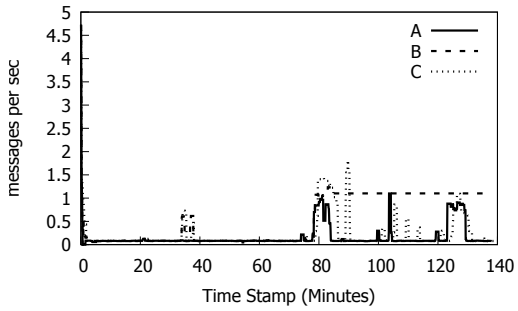
(d) Received Messages - Maximum



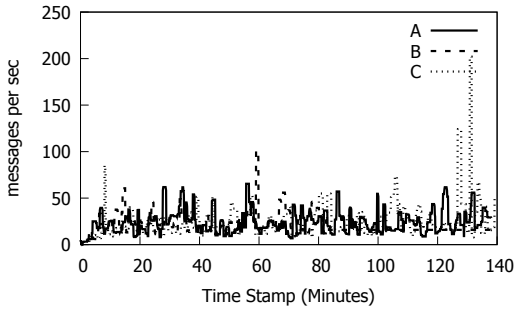
(e) Received Messages - Rate



(f) Received Messages - Average Rate



(g) Received Messages - Minimum Rate



(h) Received Messages - Maximum Rate

Figure 7.2.: MessageDispatcher - Received Messages (Count, Rate)

7.4. Storage Dispatcher

7.4.1. Stored Data

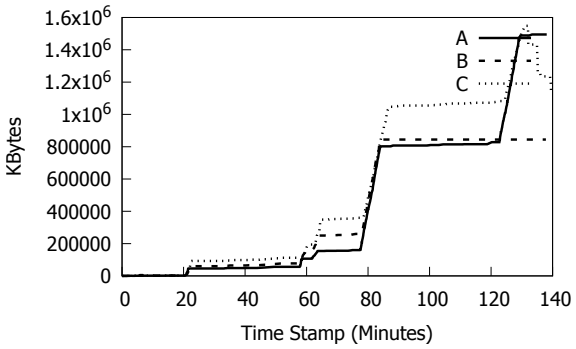


Figure 7.3.: StorageDispatcher - Stored Data

7.4.2. Retrieved Data

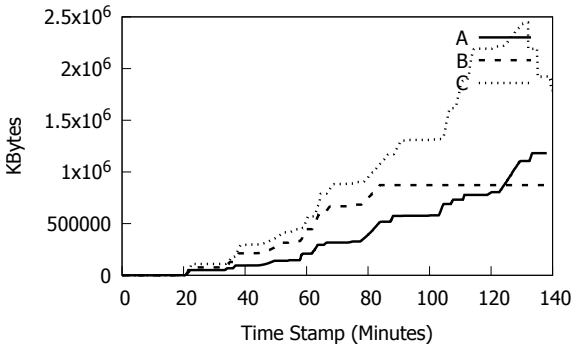
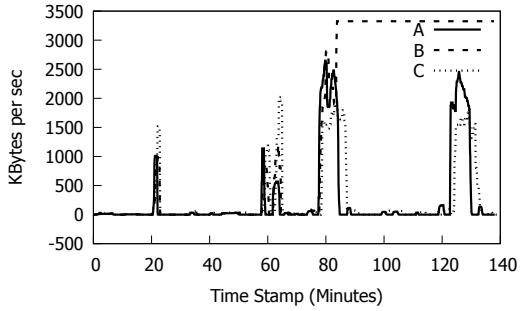
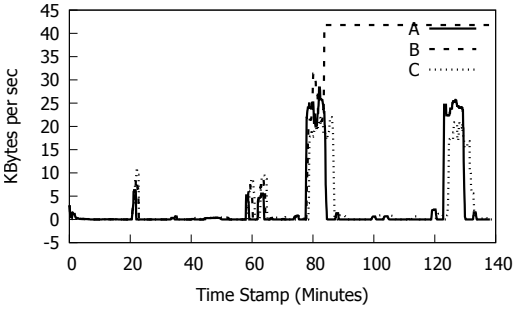


Figure 7.4.: StorageDispatcher - Retrieved Data

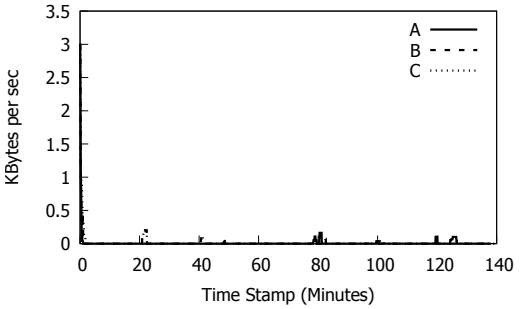
7.4.3. Stored and Retrieved Data Rate



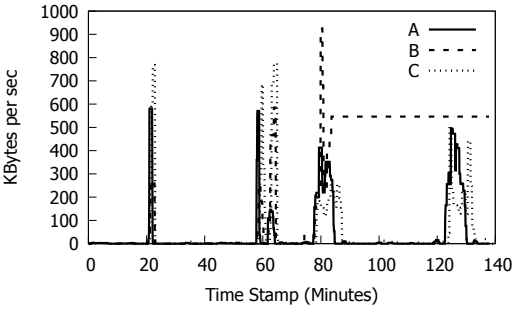
(a) Stored Data - Rate



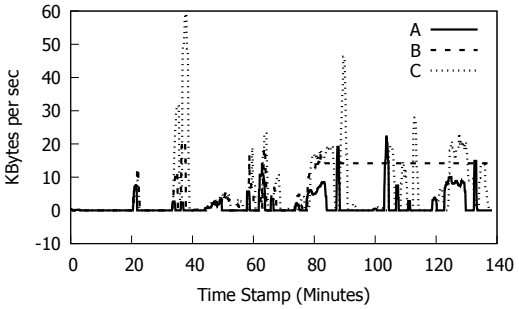
(b) Stored Data - Average Rate



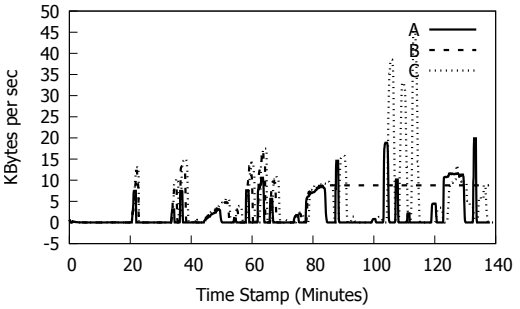
(c) Stored Data - Minimum Rate



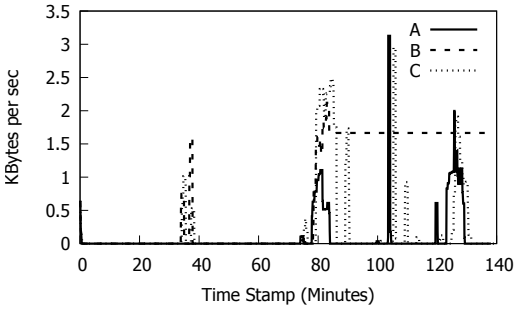
(d) Stored Data - Maximum Rate



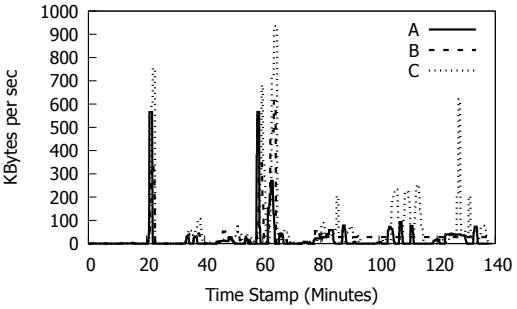
(e) Retrieved Data - Rate



(f) Retrieved Data - Average Rate



(g) Retrieved Data - Minimum Rate



(h) Retrieved Data - Maximum Rate

Figure 7.5.: StorageDispatcher - Stored and Retrieved Data Rate

7.4.4. Stored DDS Data

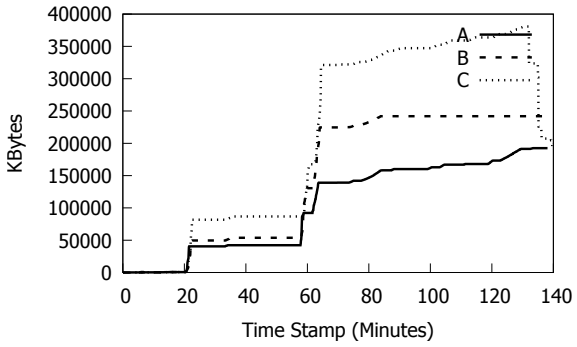


Figure 7.6.: StorageDispatcher - Stored DDS Data

7.4.5. Retrieved DDS Data

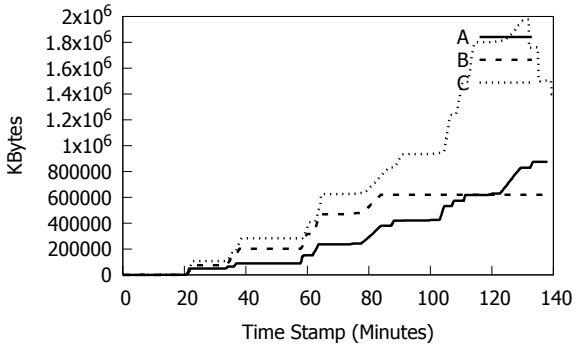
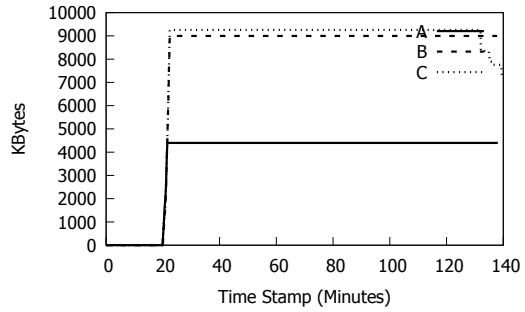
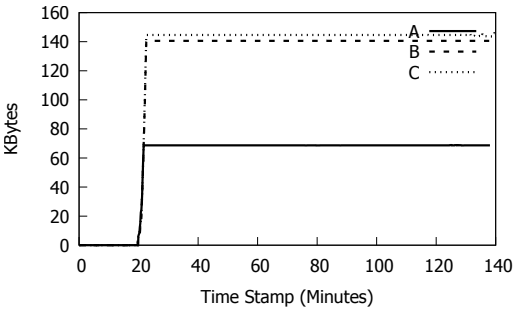


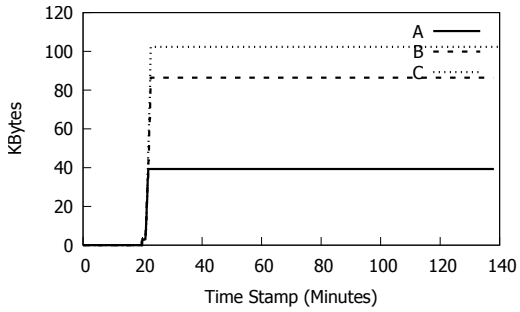
Figure 7.7.: StorageDispatcher - Retrieved DDS Data



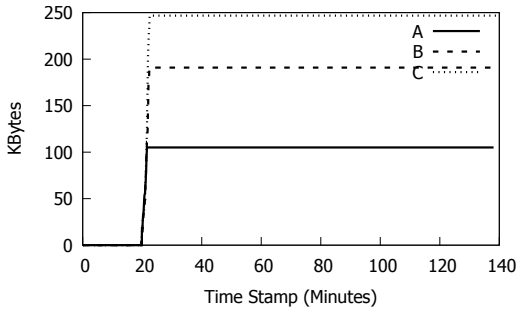
(a) Data Stored



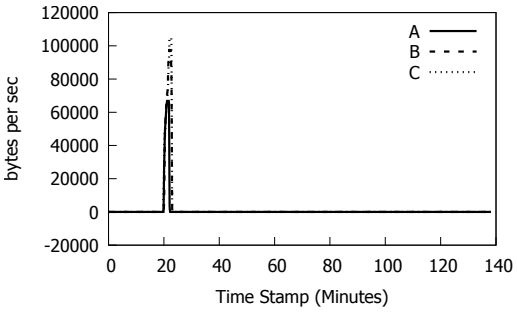
(b) Data Stored - Average



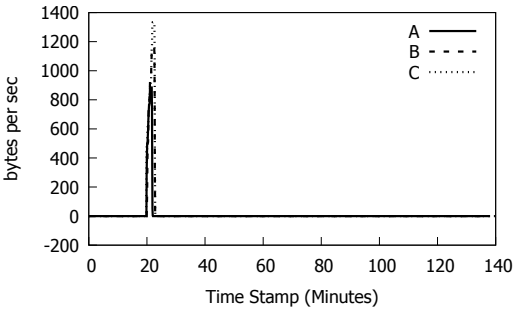
(c) Data Stored - Minimum



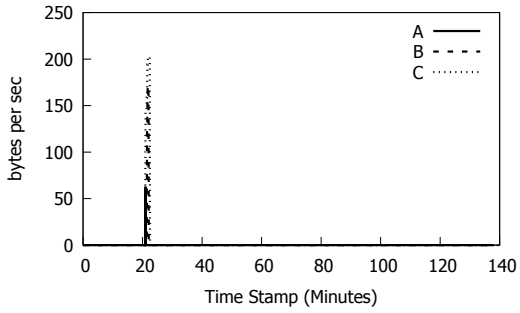
(d) Data Stored - Maximum



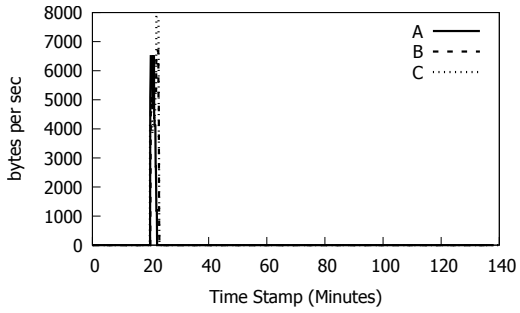
(e) Data Stored Rate



(f) Data Stored Rate - Average



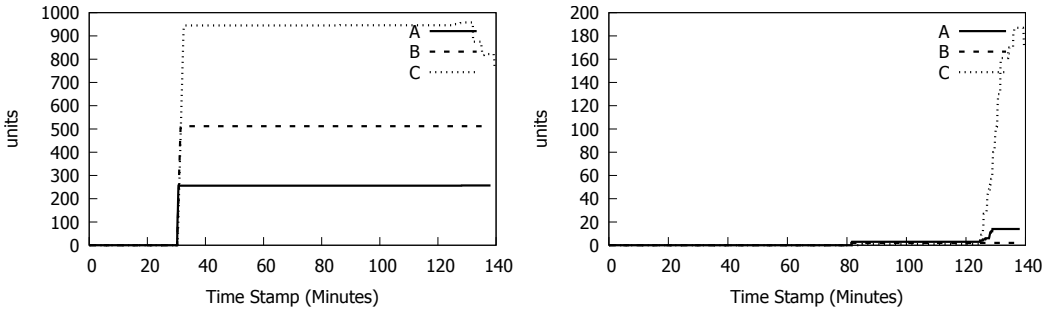
(g) Data Stored Rate - Minimum



(h) Data Stored Rate - Maximum

Figure 7.8.: StorageDispatcher - Stored and Retrieved DDS Data Rate

7.5. Null Pointer and Other Exceptions



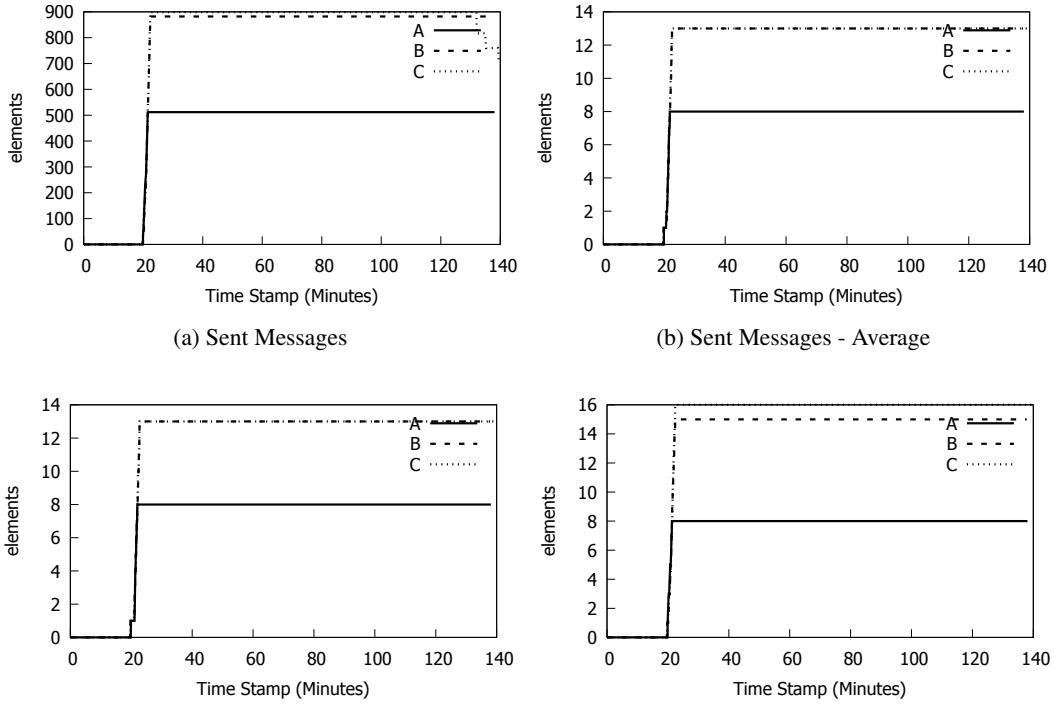
(a) Null Pointer Exception

(b) Other Exceptions

Figure 7.9.: Null Pointer Exceptions vs. Other Exceptions

7.6. Plugins with high Messaging Impact

7.6.1. Message Plugin



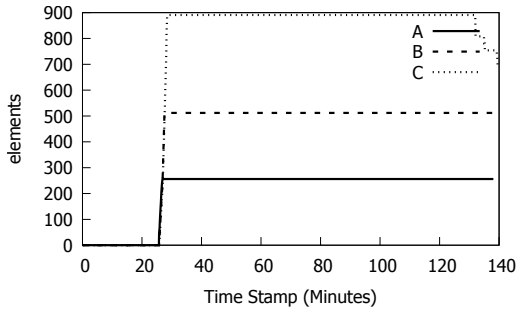
(a) Sent Messages

(b) Sent Messages - Average

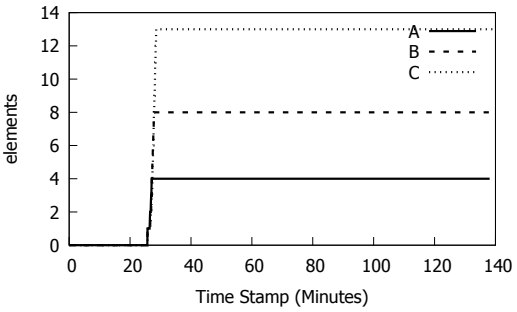
(c) Sent Messages - Minimum

(d) Sent Messages - Maximum

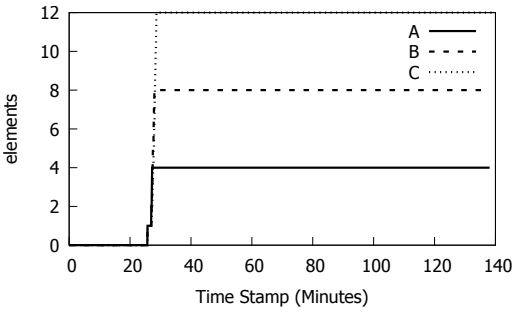
Figure 7.10.: MESSAGING plugin - Sent Messages



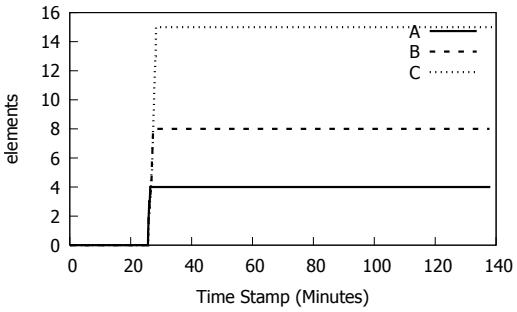
(a) Inbox Views



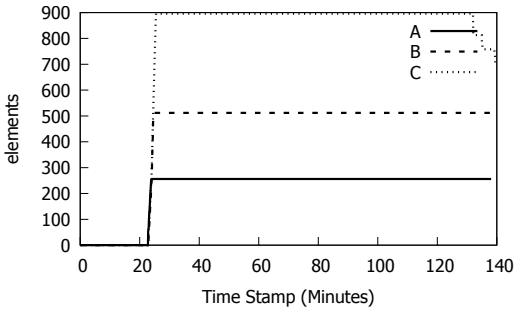
(b) Inbox Views - Average



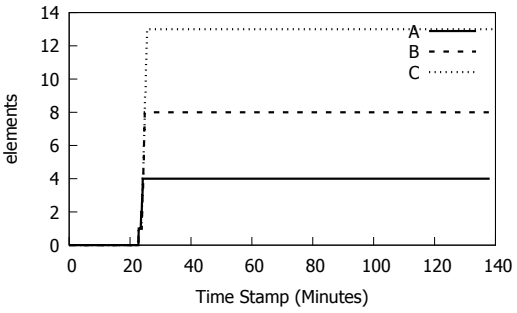
(c) Inbox Views - Minimum



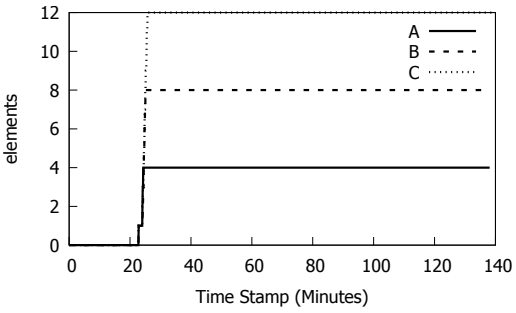
(d) Inbox Views - Maximum



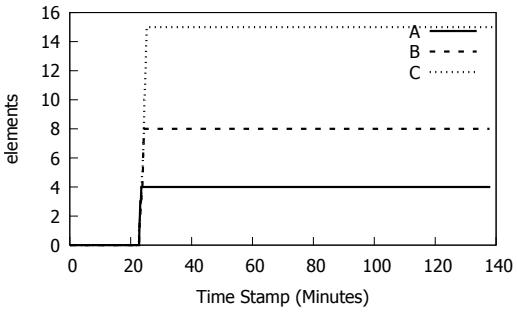
(e) Outbox Views



(f) Outbox Views - Average

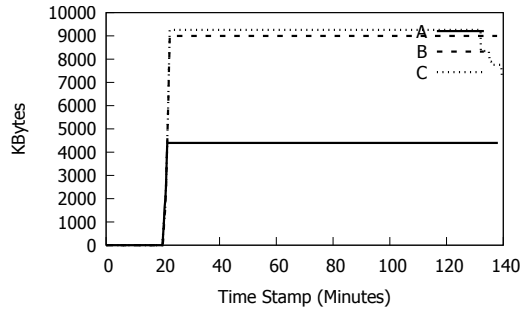


(g) Outbox Views - Minimum

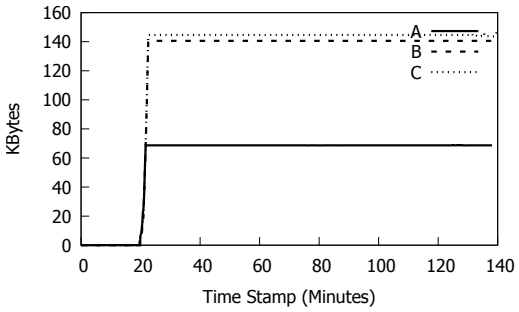


(h) Outbox Views - Maximum

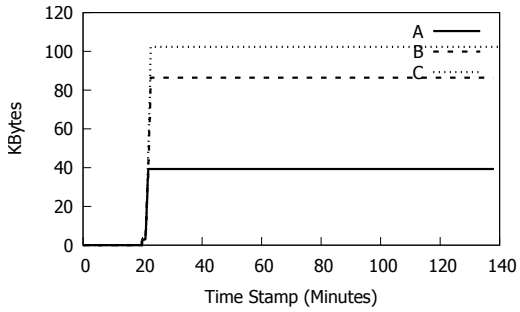
Figure 7.11.: MESSAGING plugin - Inbox and Outbox Views



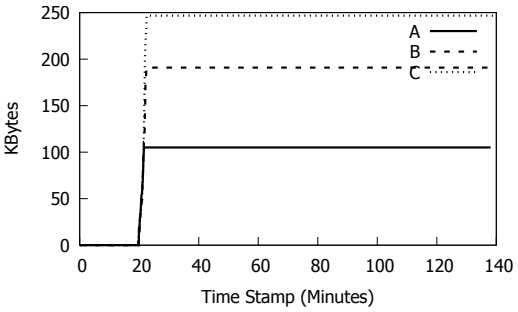
(a) Data Stored



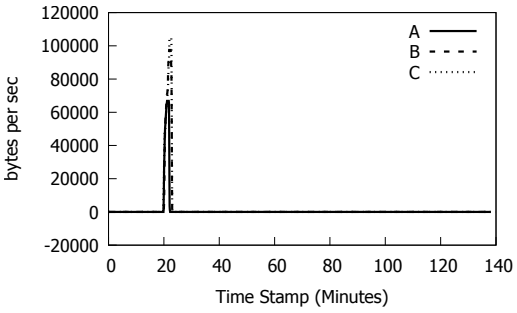
(b) Data Stored - Average



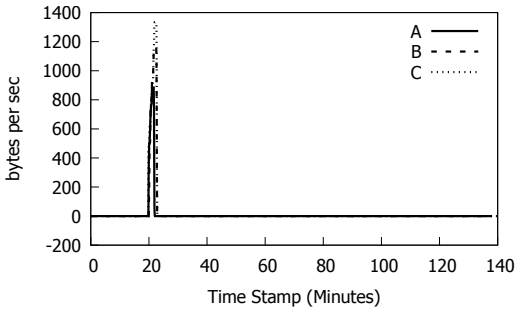
(c) Data Stored - Minimum



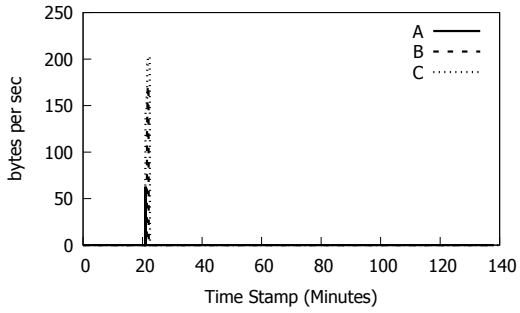
(d) Data Stored - Maximum



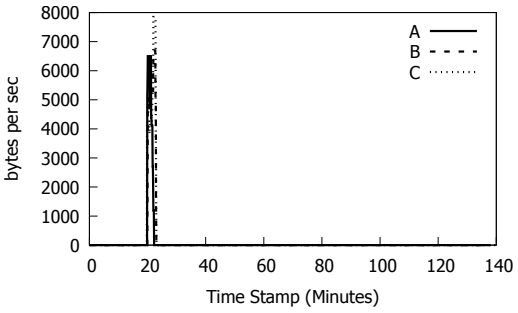
(e) Data Stored Rate



(f) Data Stored Rate - Average

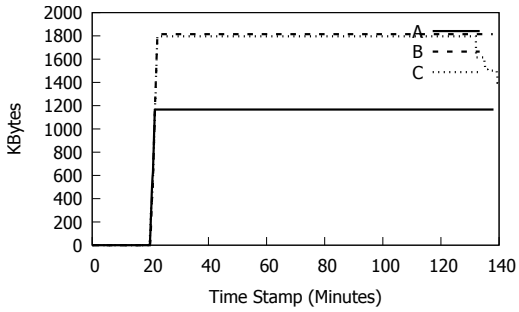


(g) Data Stored Rate - Minimum

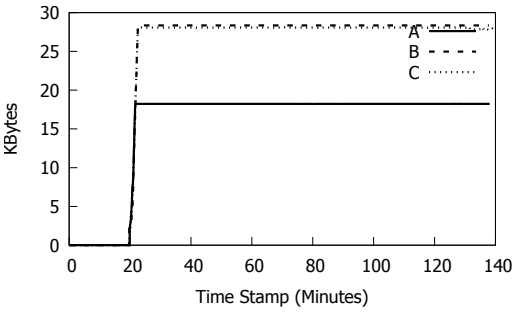


(h) Data Stored Rate - Maximum

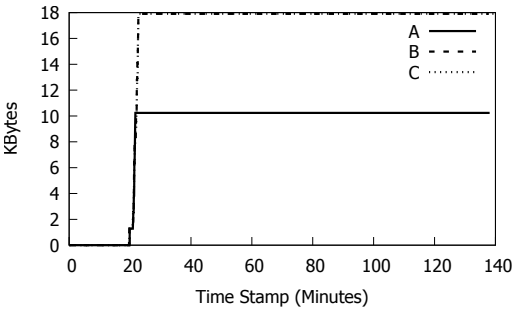
Figure 7.12.: MESSAGING plugin - Data Stored Amount and Rate



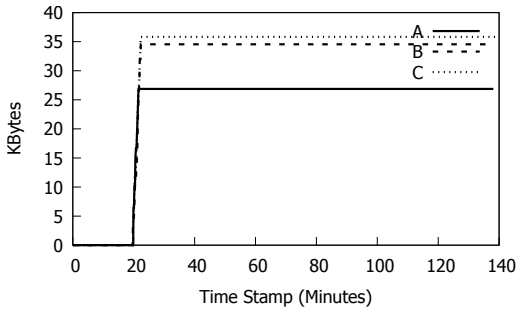
(a) Data Retrieved



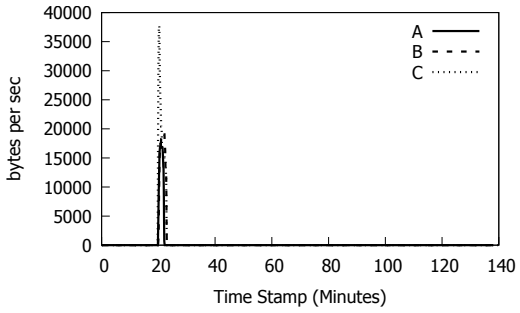
(b) Data Retrieved - Average



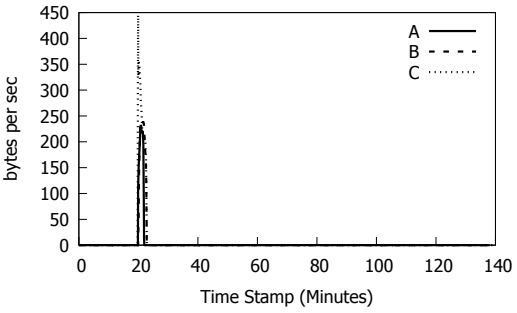
(c) Data Retrieved - Minimum



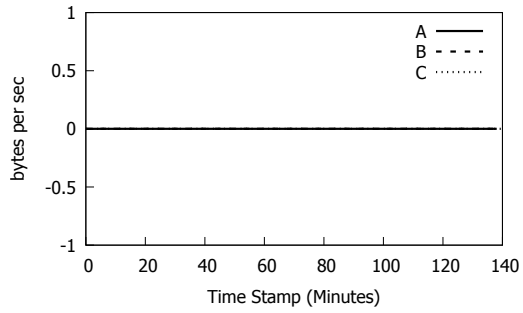
(d) Data Retrieved - Maximum



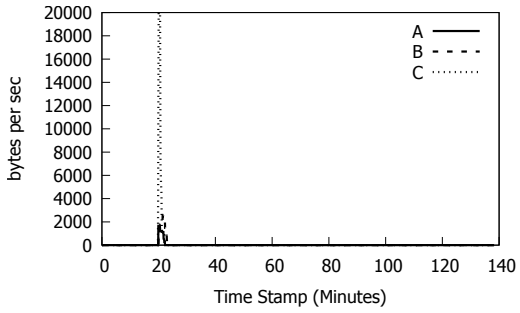
(e) Data Retrieved Rate



(f) Data Retrieved Rate - Average



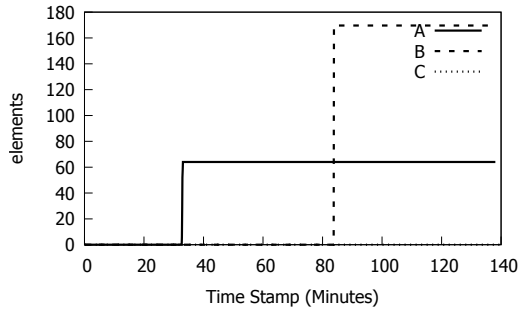
(g) Data Retrieved Rate - Minimum



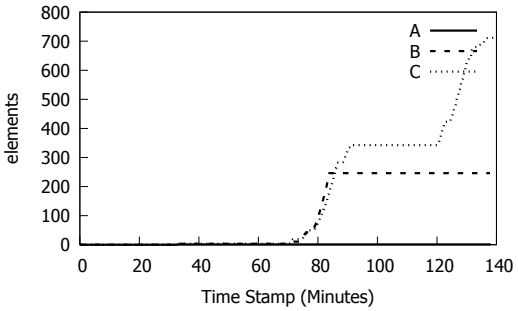
(h) Data Retrieved Rate - Maximum

Figure 7.13.: MESSAGING plugin - Data Retrieved Amount and Rate

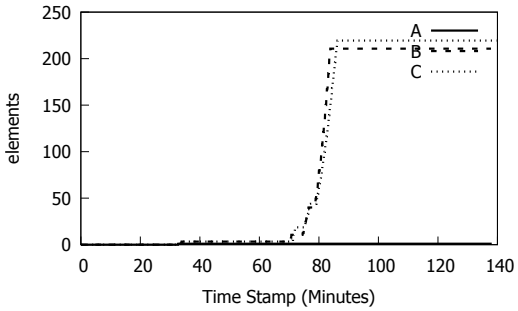
7.6.2. Forum Plugin



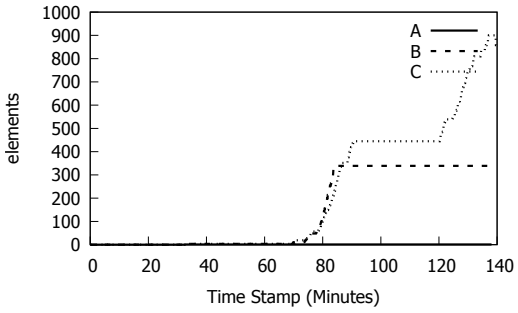
(a) Created Forum Threads



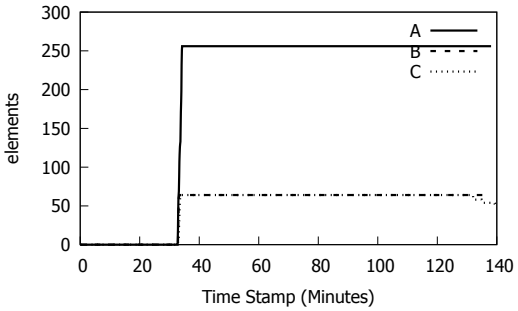
(b) Created Forum Threads - Average



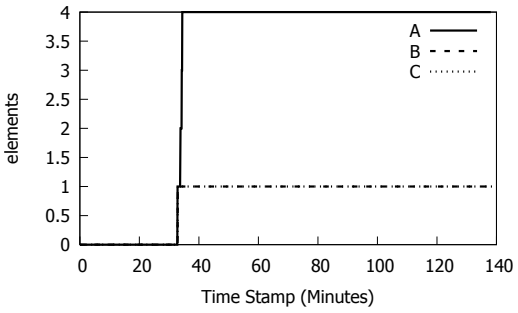
(c) Created Forum Threads - Minimum



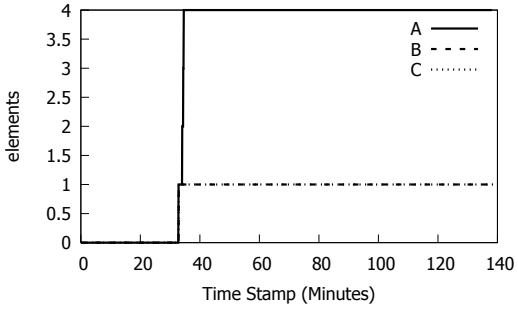
(d) Created Forum Threads - Maximum



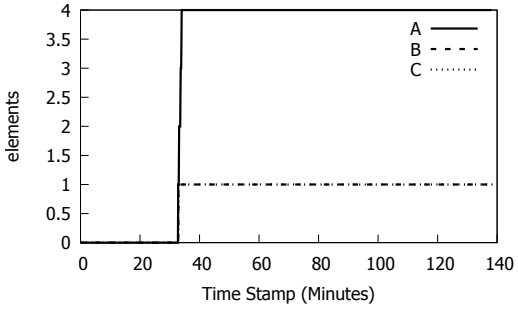
(e) Created Forum Posts Rate



(f) Created Forum Posts Rate - Average

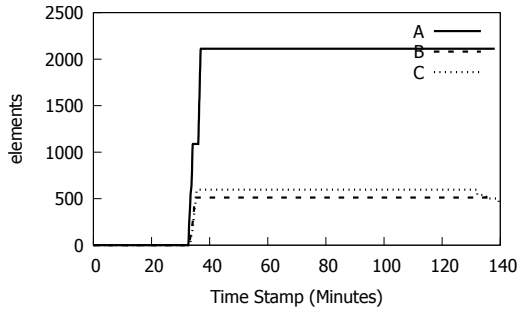


(g) Created Forum Posts Rate - Minimum

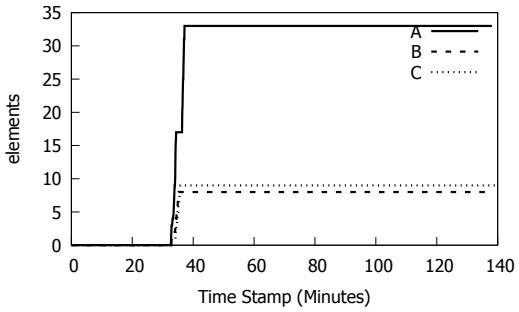


(h) Created Forum Posts Rate - Maximum

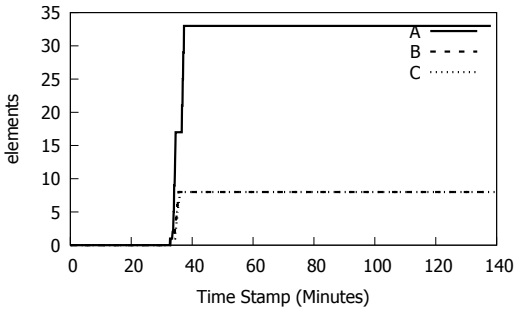
Figure 7.14.: FORUM plugin - Number of Created Forum Threads and Posts



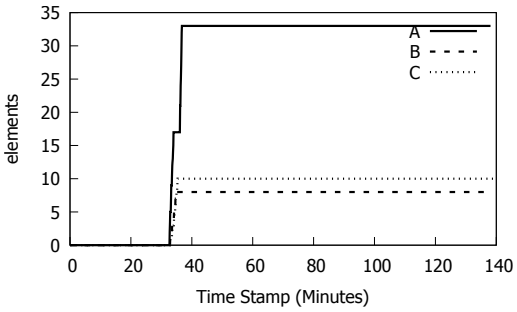
(a) Forum Views



(b) Forum Views - Average

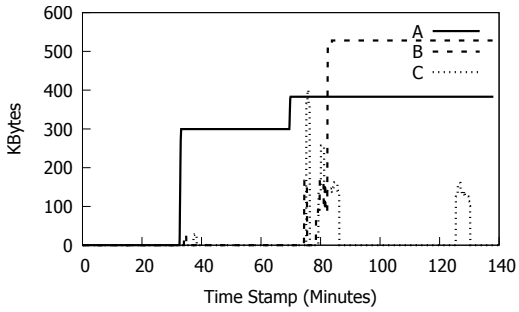


(c) Forum Views - Minimum

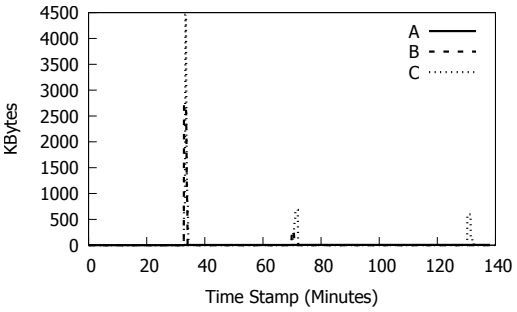


(d) Forum Views - Maximum

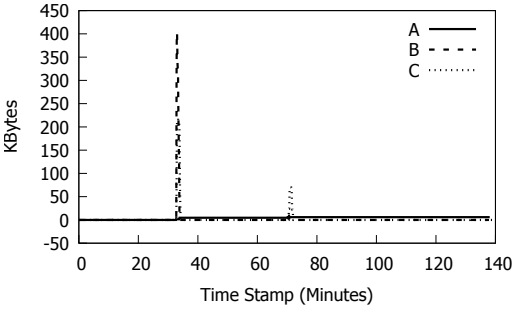
Figure 7.15.: FORUM plugin - Number of Forum Views



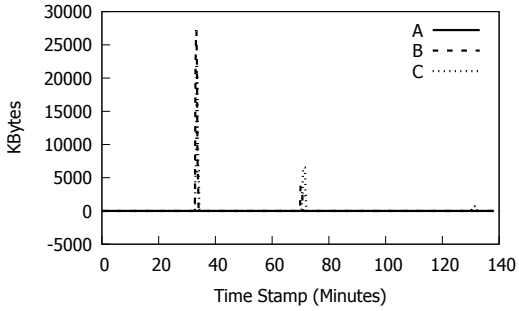
(a) Data Stored



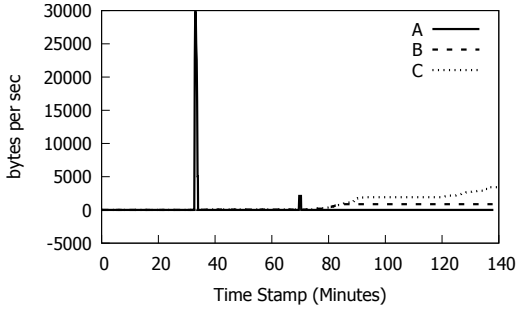
(b) Data Stored - Average



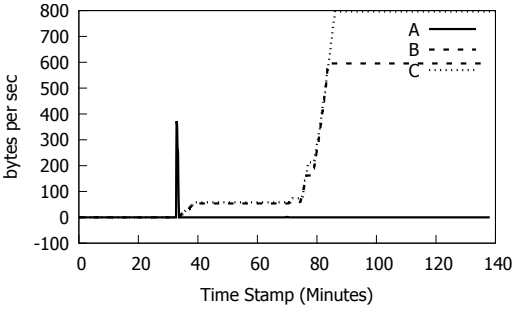
(c) Data Stored - Minimum



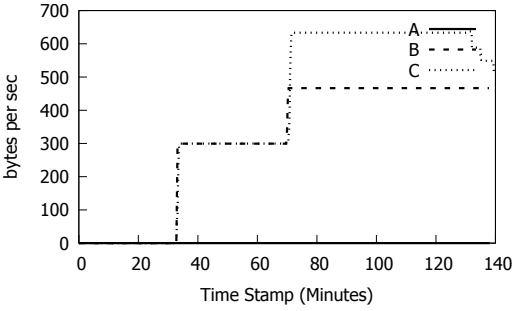
(d) Data Stored - Maximum



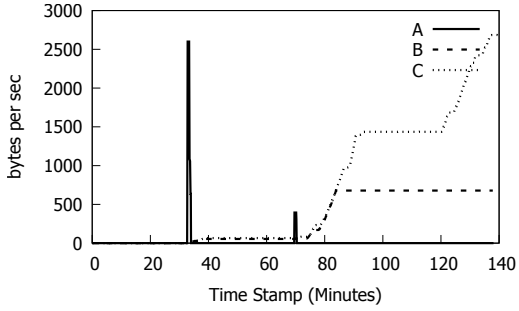
(e) Data Stored Rate



(f) Data Stored Rate - Average

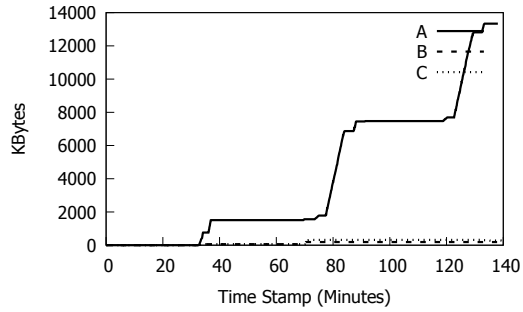


(g) Data Stored Rate - Minimum

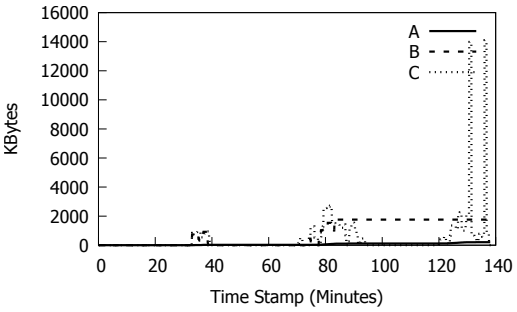


(h) Data Stored Rate - Maximum

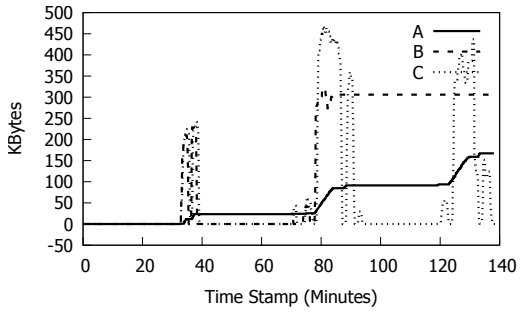
Figure 7.16.: FFORUM plugin - Data Stored Amount and Rate



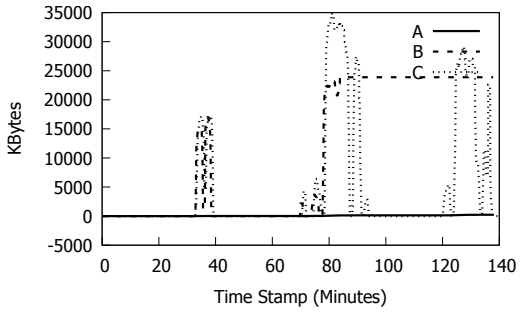
(a) Data Retrieved



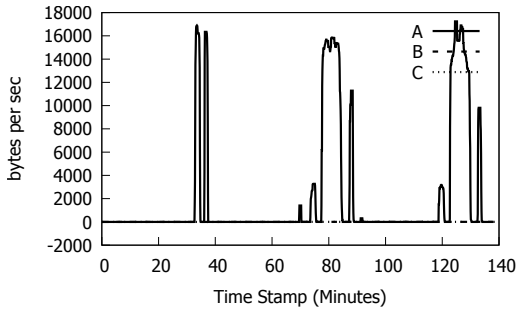
(b) Data Retrieved - Average



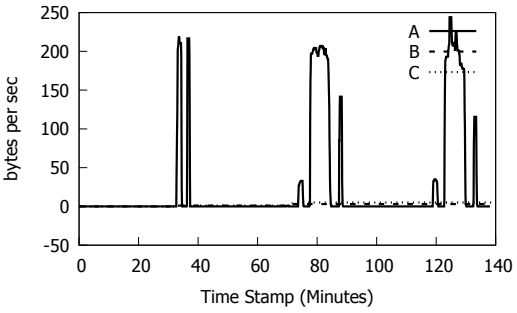
(c) Data Retrieved - Minimum



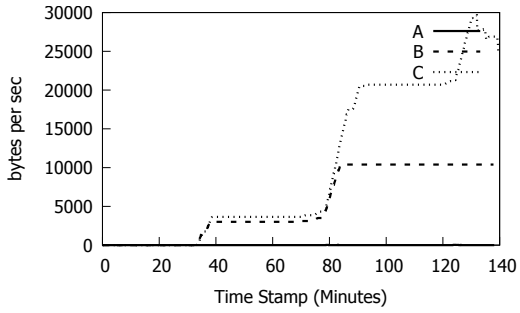
(d) Data Retrieved - Maximum



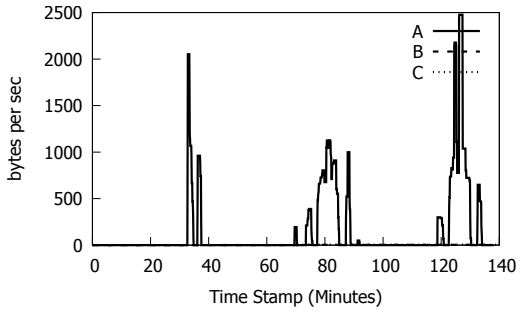
(e) Data Retrieved Rate



(f) Data Retrieved Rate - Average



(g) Data Retrieved Rate - Minimum

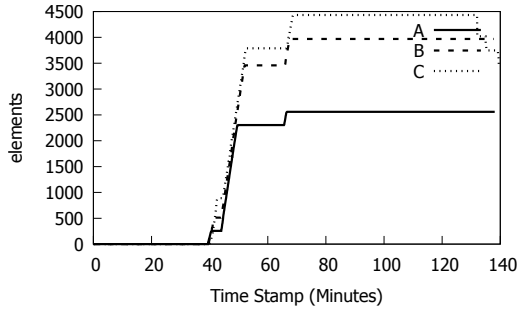


(h) Data Retrieved Rate - Maximum

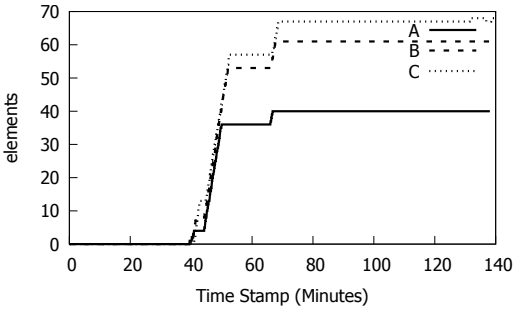
Figure 7.17.: FFORUM plugin - Data Retrieved Amount and Rate

7.7. Plugins with high Storage Impact

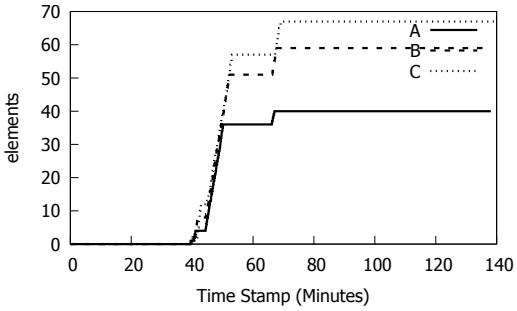
7.7.1. Photo Plugin



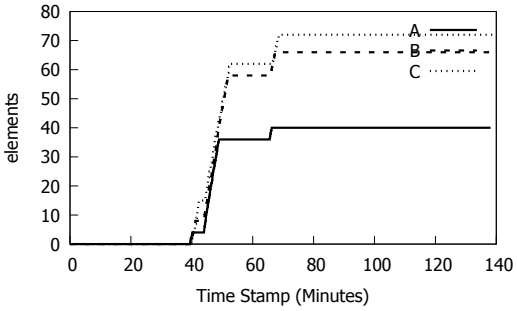
(a) Created Albums



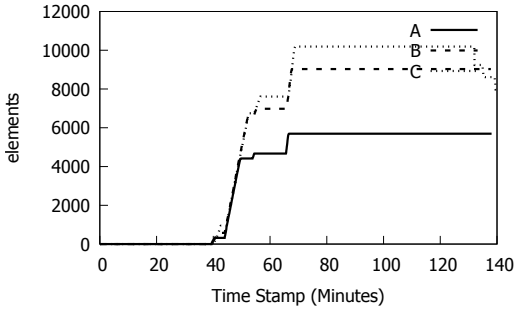
(b) Created Albums - Average



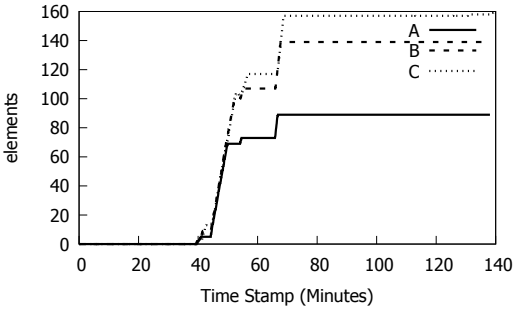
(c) Created Albums - Minimum



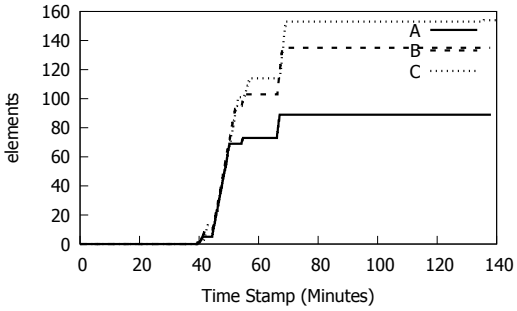
(d) Created Albums - Maximum



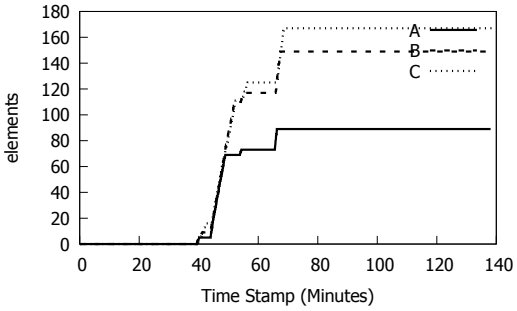
(e) Album Views



(f) Album Views - Average

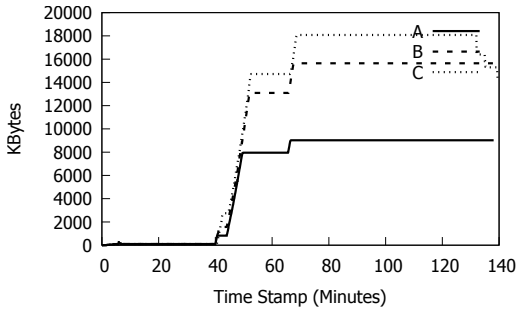


(g) Album Views - Minimum

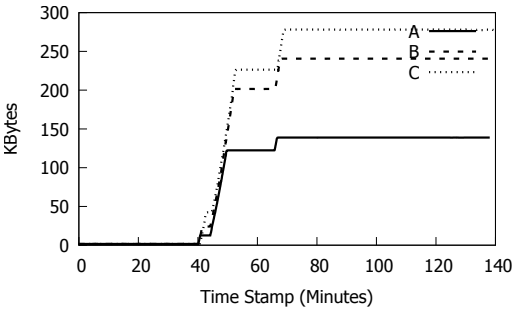


(h) Album Views - Maximum

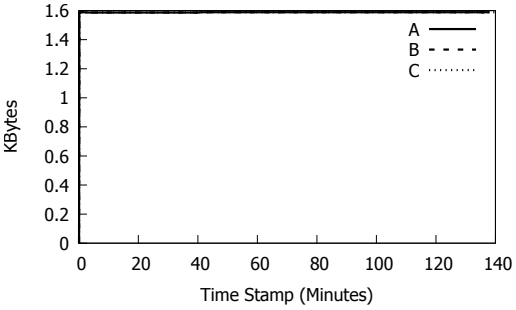
Figure 7.18.: PHOTO plugin - Created Album and Views



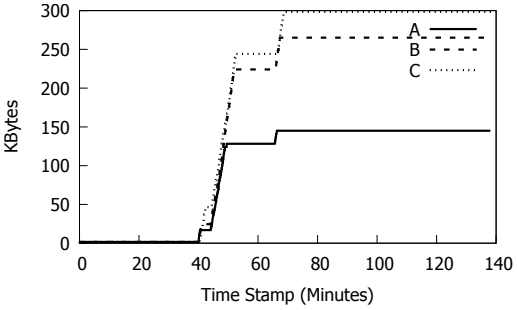
(a) Data Stored



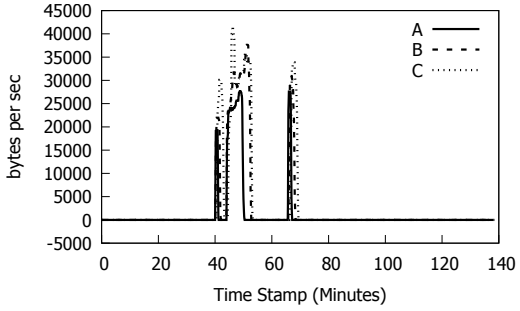
(b) Data Stored - Average



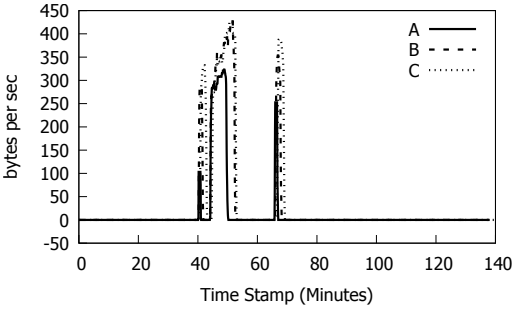
(c) Data Stored - Minimum



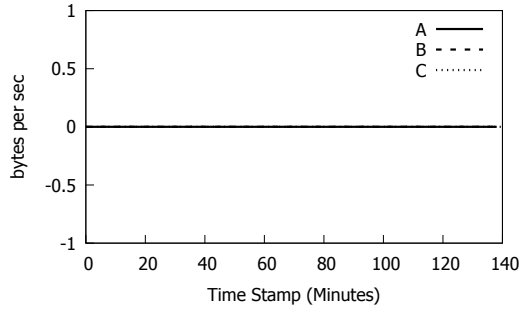
(d) Data Stored - Maximum



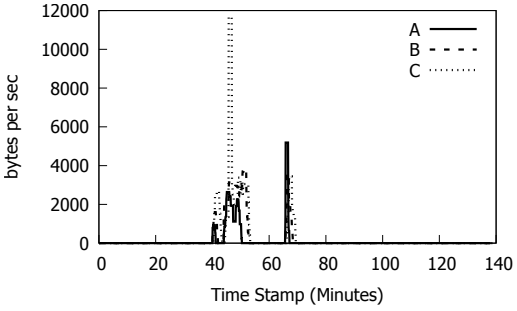
(e) Data Stored Rate



(f) Data Stored Rate - Average

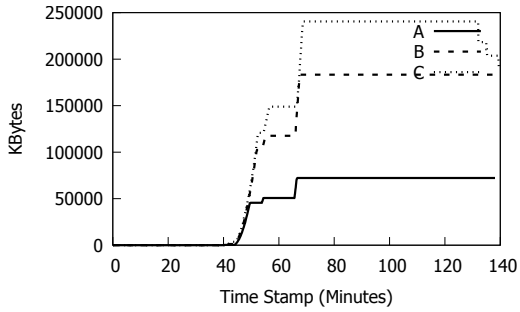


(g) Data Stored Rate - Minimum

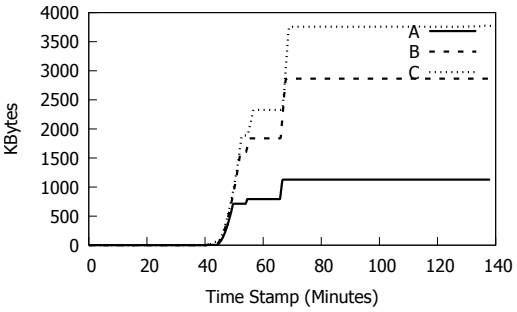


(h) Data Stored Rate - Maximum

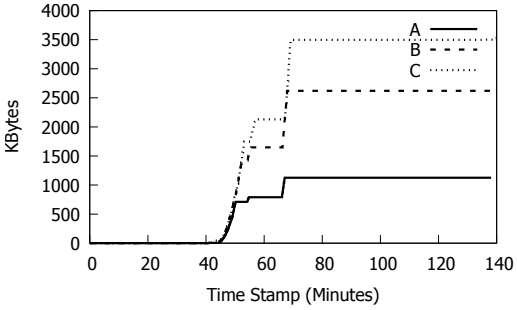
Figure 7.19.: PHOTO plugin - Data Stored Amount and Rate



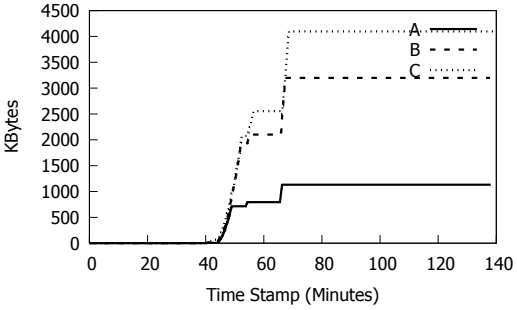
(a) Data Retrieved



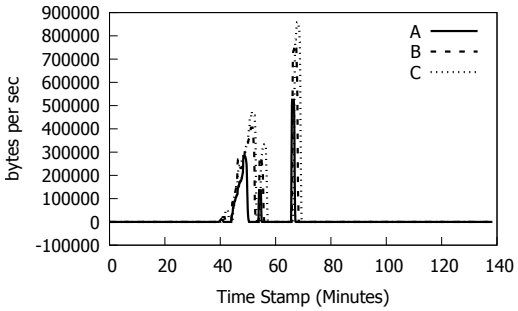
(b) Data Retrieved - Average



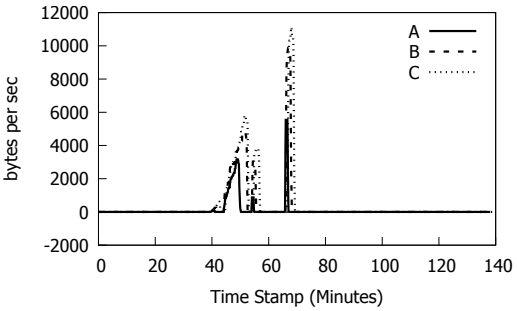
(c) Data Retrieved - Minimum



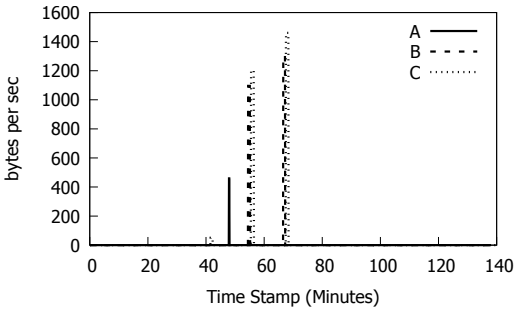
(d) Data Retrieved - Maximum



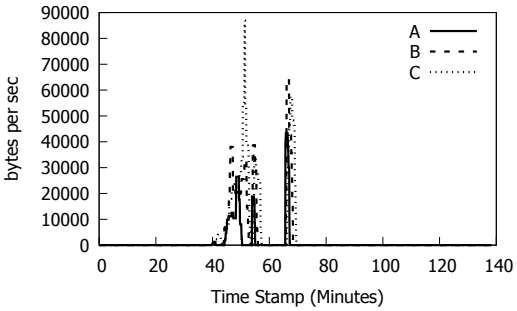
(e) Data Retrieved Rate



(f) Data Retrieved Rate - Average



(g) Data Retrieved Rate - Minimum

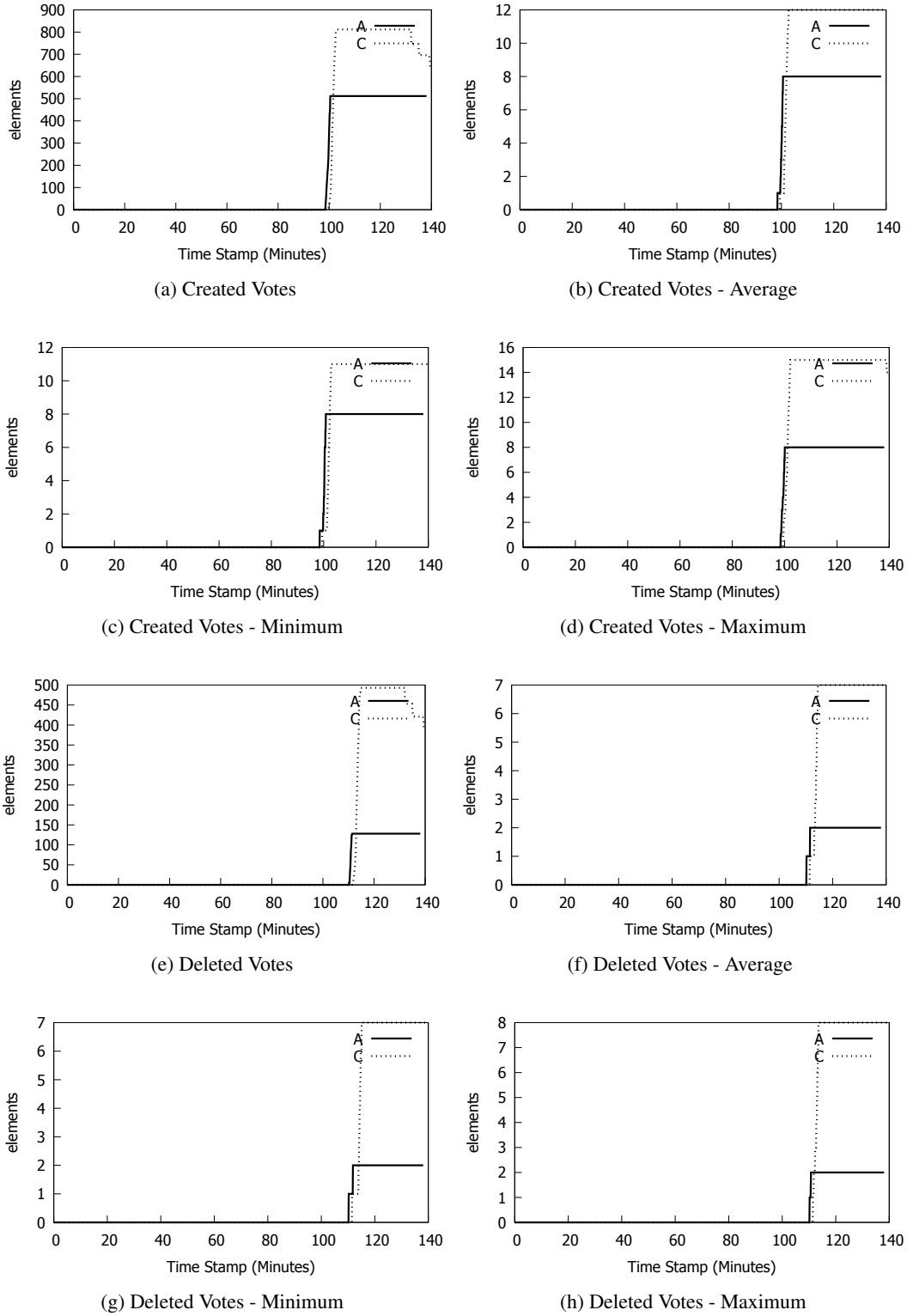


(h) Data Retrieved Rate - Maximum

Figure 7.20.: PHOTO plugin - Data Retrieved Amount and Rate

7.8. Plugins with Mixed Impact

7.8.1. Vote Plugin



(a) Created Votes

(b) Created Votes - Average

(c) Created Votes - Minimum

(d) Created Votes - Maximum

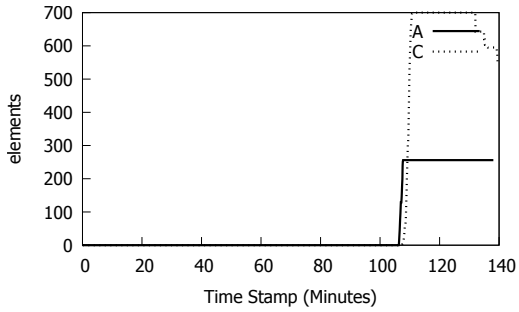
(e) Deleted Votes

(f) Deleted Votes - Average

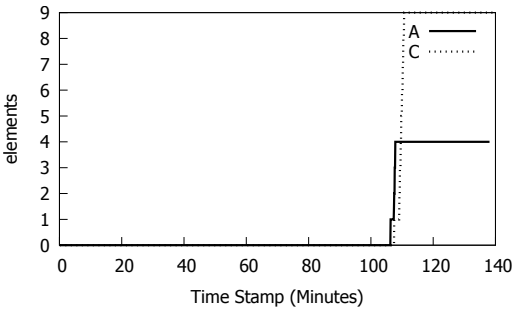
(g) Deleted Votes - Minimum

(h) Deleted Votes - Maximum

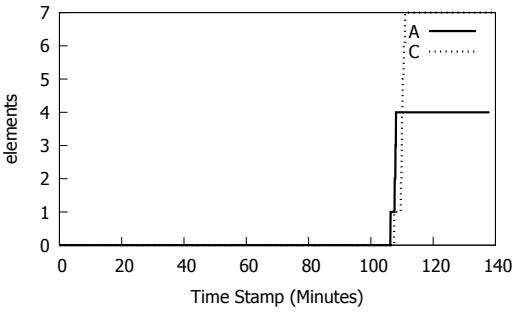
Figure 7.21.: VOTE plugin - Created and Deleted Votes



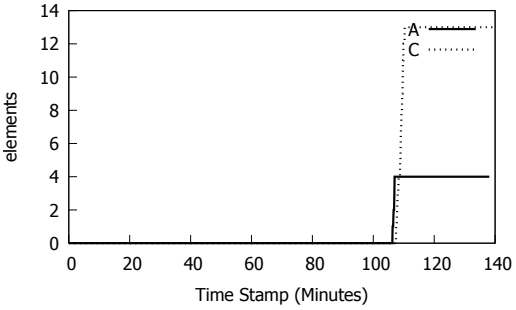
(a) Retrieved Voting Results



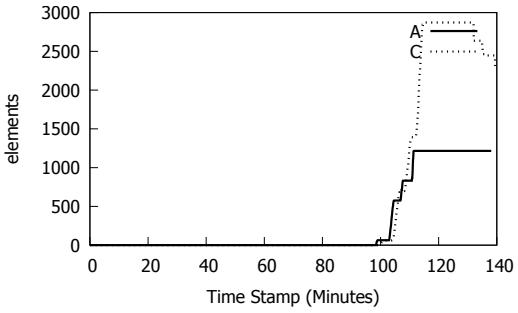
(b) Retrieved Voting Results - Average



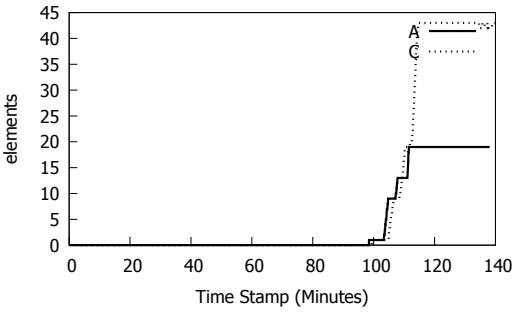
(c) Retrieved Voting Results - Minimum



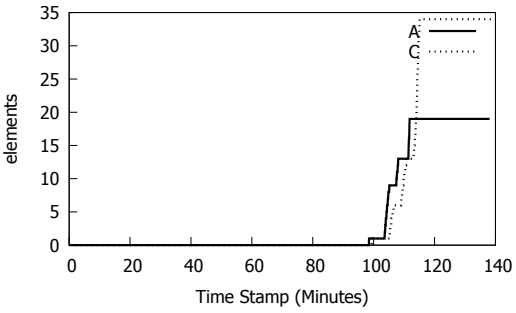
(d) Retrieved Voting Results - Maximum



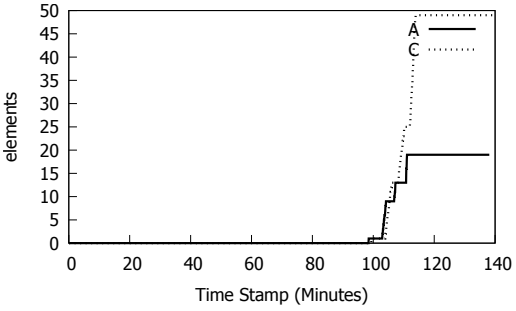
(e) Voting List Views



(f) Voting List Views - Average

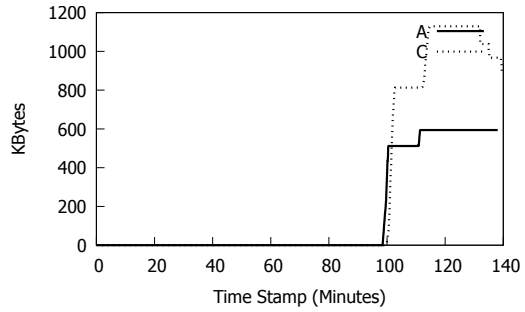


(g) Voting List Views - Minimum

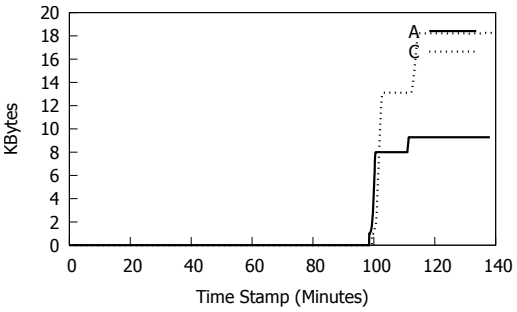


(h) Voting List Views - Maximum

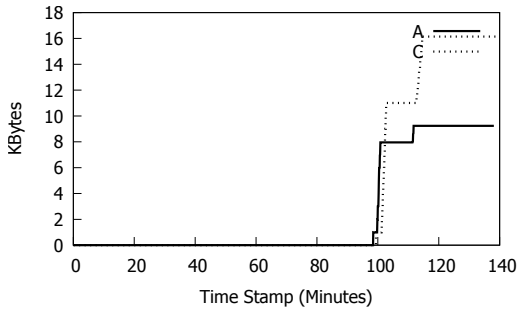
Figure 7.22.: VOTE plugin - Retrieved Voting Results and Voting List Views



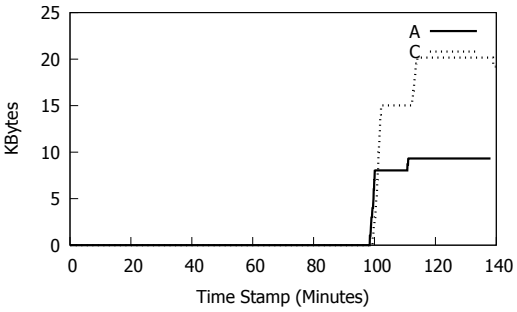
(a) Data Stored



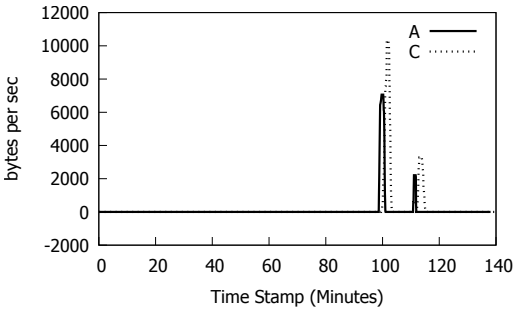
(b) Data Stored - Average



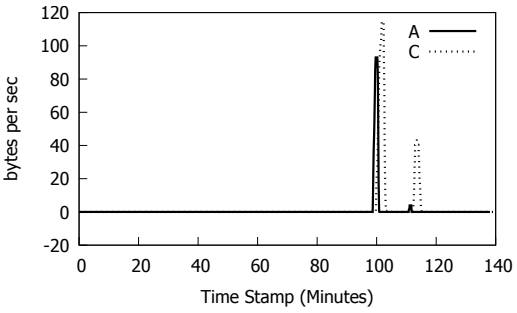
(c) Data Stored - Minimum



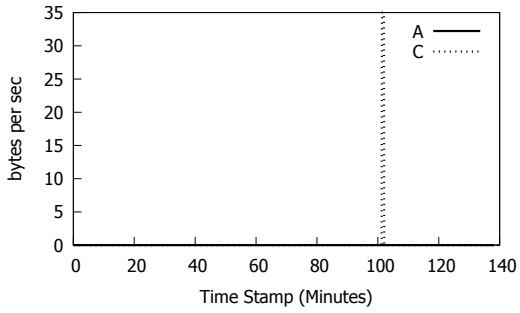
(d) Data Stored - Maximum



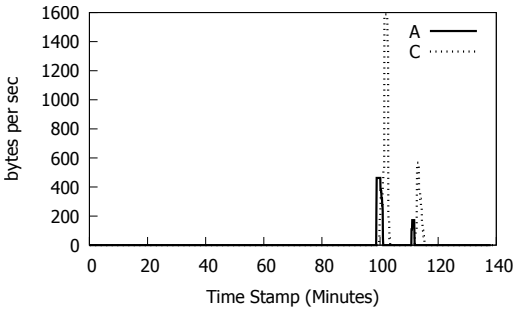
(e) Data Stored Rate



(f) Data Stored Rate - Average

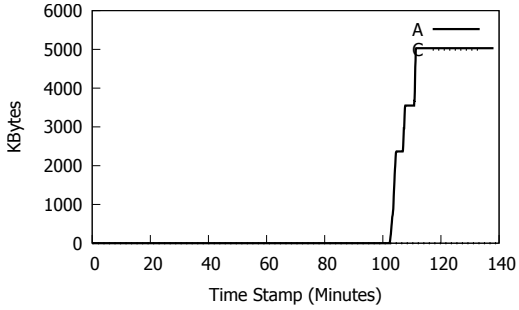


(g) Data Stored Rate - Minimum

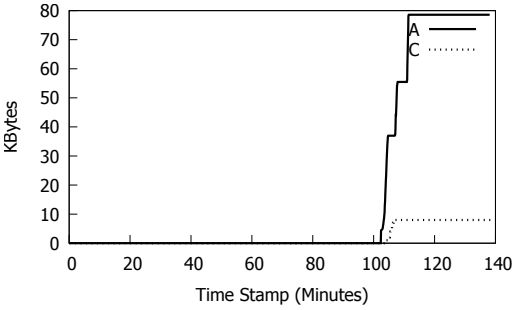


(h) Data Stored Rate - Maximum

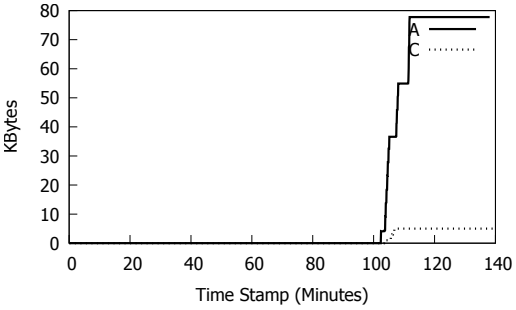
Figure 7.23.: VOTE plugin - Data Stored Amount and Rate



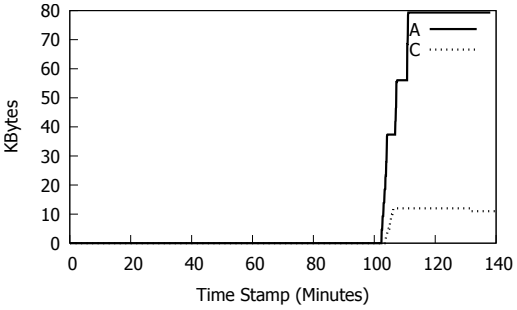
(a) Data Retrieved



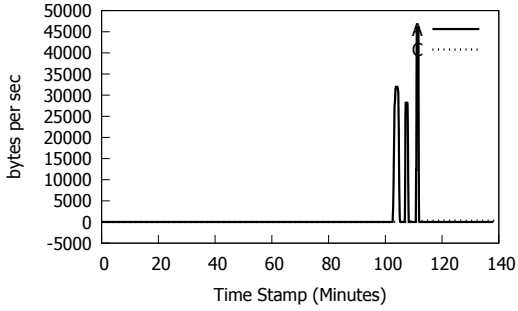
(b) Data Retrieved - Average



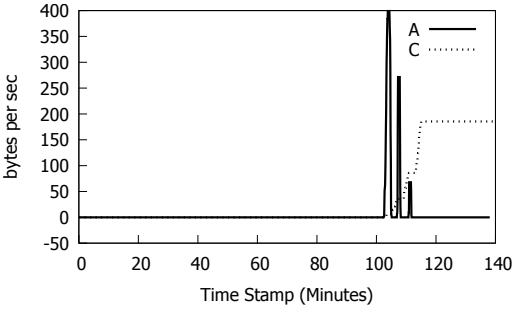
(c) Data Retrieved - Minimum



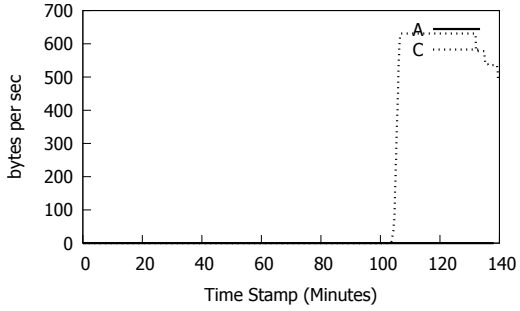
(d) Data Retrieved - Maximum



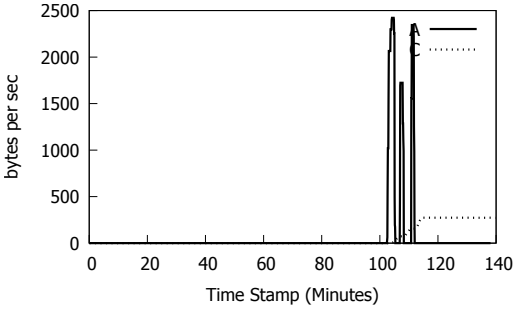
(e) Data Retrieved Rate



(f) Data Retrieved Rate - Average



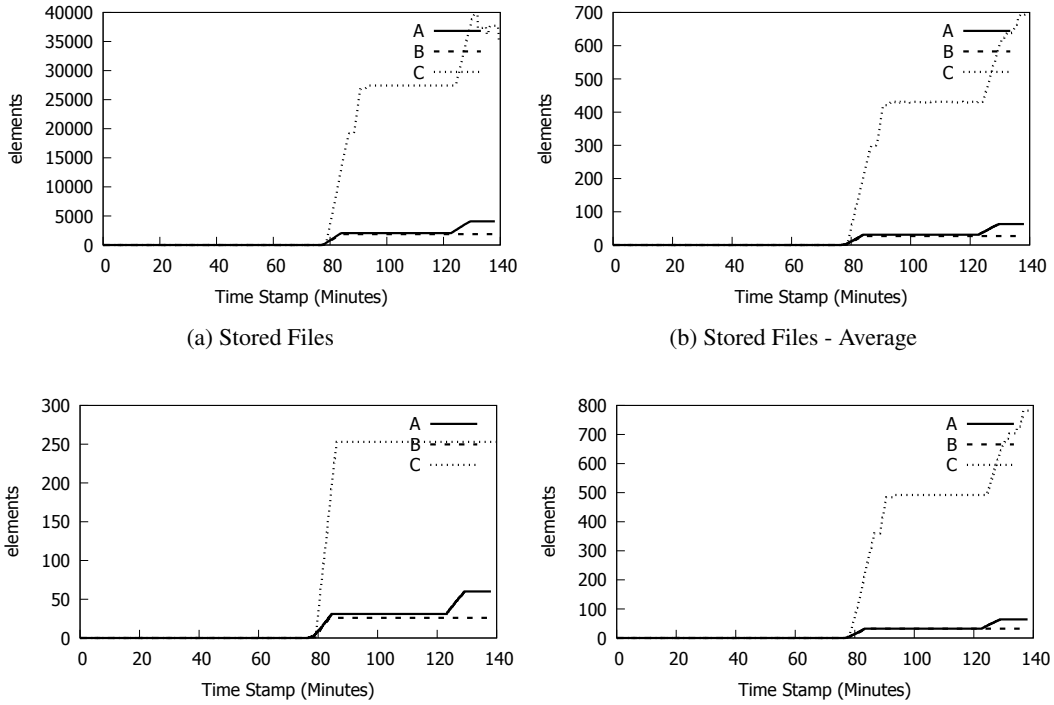
(g) Data Retrieved Rate - Minimum



(h) Data Retrieved Rate - Maximum

Figure 7.24.: VOTE plugin - Data Retrieved Amount and Rate

7.8.2. Filestorage Plugin



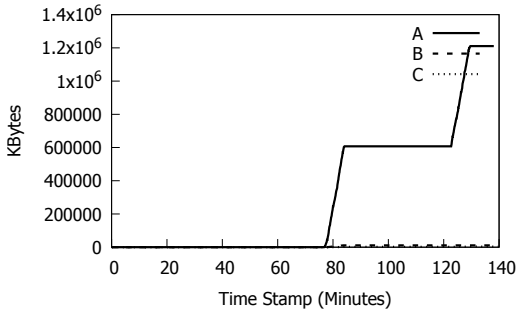
(a) Stored Files

(b) Stored Files - Average

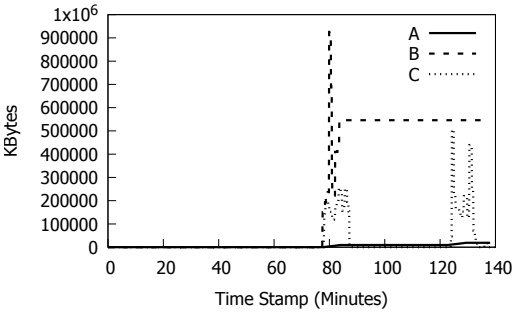
(c) Stored Files - Minimum

(d) Stored Files - Maximum

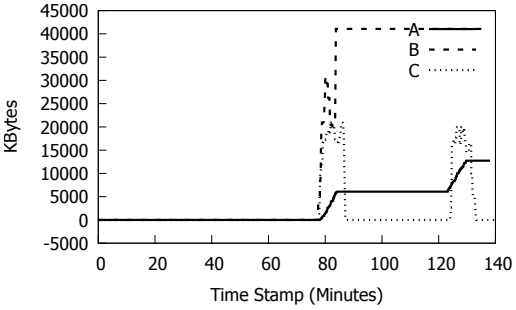
Figure 7.25.: FILESTORAGE plugin - Stored Files



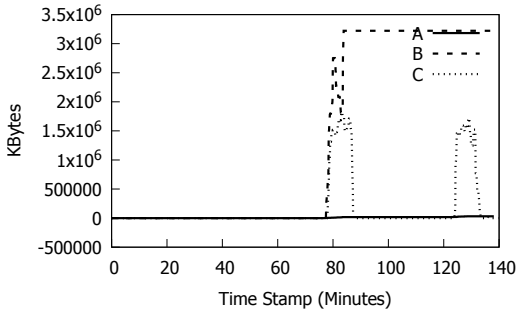
(a) Data Stored



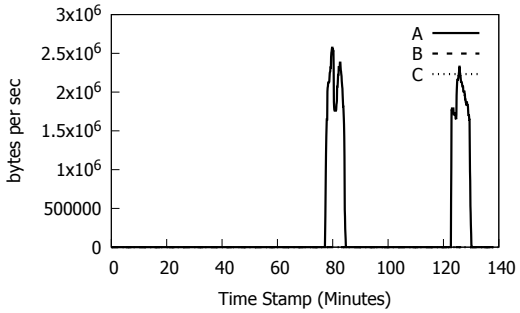
(b) Data Stored - Average



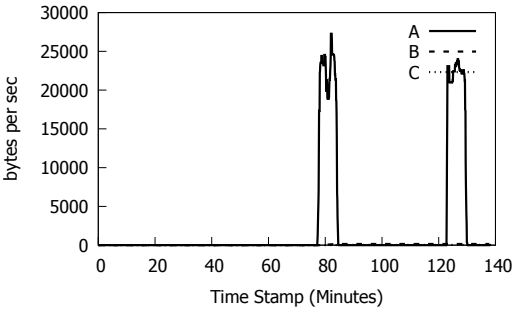
(c) Data Stored - Minimum



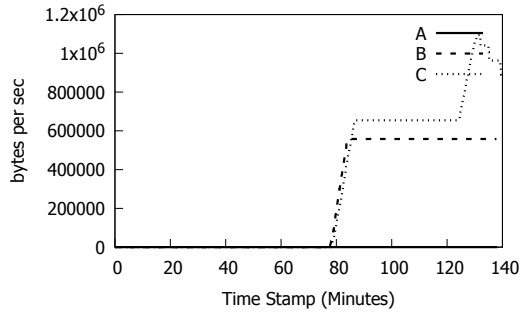
(d) Data Stored - Maximum



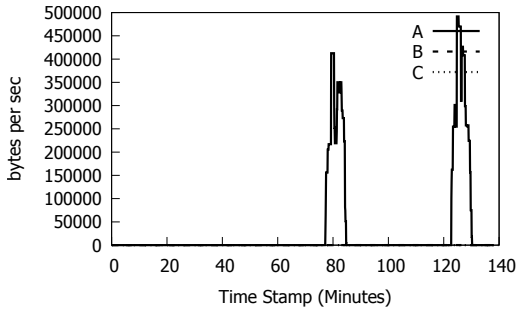
(e) Data Stored Rate



(f) Data Stored Rate - Average

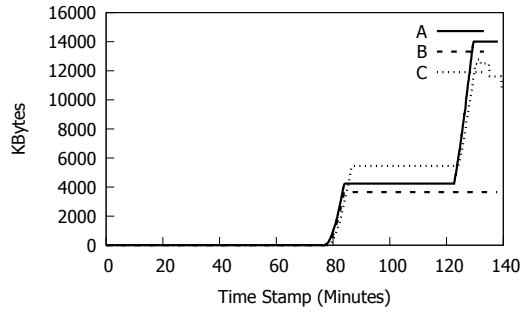


(g) Data Stored Rate - Minimum

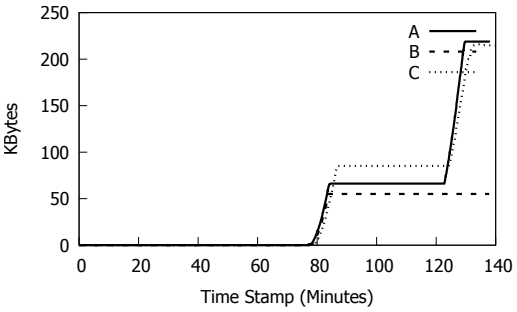


(h) Data Stored Rate - Maximum

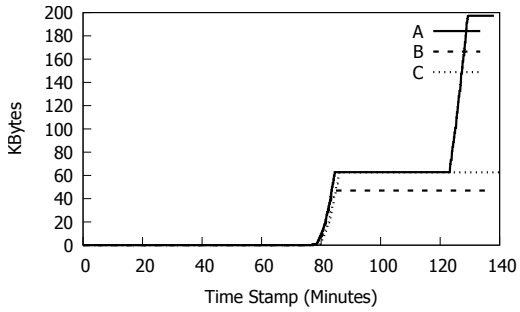
Figure 7.26.: FILESTORAGE plugin - Data Stored Amount and Rate



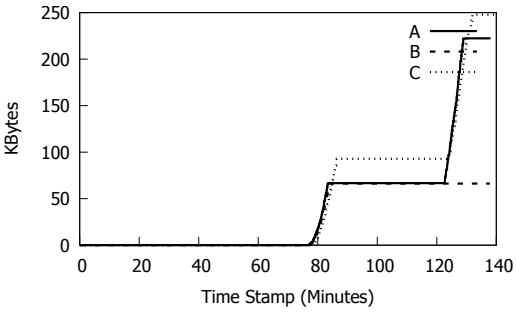
(a) Data Retrieved



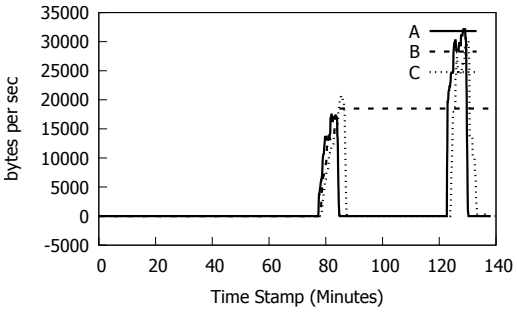
(b) Data Retrieved - Average



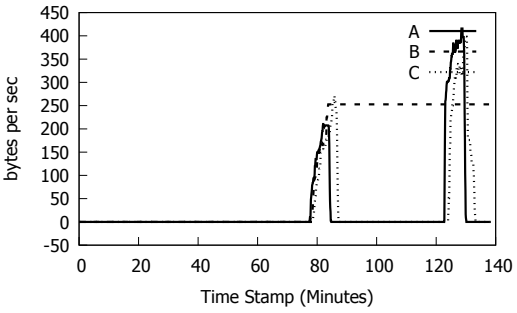
(c) Data Retrieved - Minimum



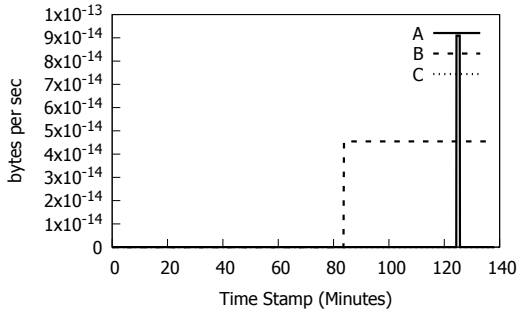
(d) Data Retrieved - Maximum



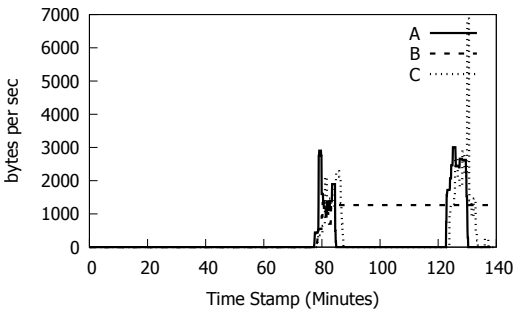
(e) Data Retrieved Rate



(f) Data Retrieved Rate - Average



(g) Data Retrieved Rate - Minimum



(h) Data Retrieved Rate - Maximum

Figure 7.27.: FILESTORAGE plugin - Data Retrieved Amount and Rate

Bibliography

[1] C. Baun and G. Bengel, *Masterkurs Parallele und Verteilte Systeme*, 2015.

[2] K. Graffi, "Monitoring and management of peer-to-peer systems," Ph.D. dissertation, Technische Universität, 2010.

[3] D. Landesbeauftragte für den Datenschutz, "Facebook in der verantwortung," 2018.

[4] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," *arXiv preprint cs/0412017*, 2004.

[5] K. Graffi and D. Stingl, "Monitoring and Management of Structured Peer-to-Peer Systems," in *IEEE P2P '09: Proc. of the Int. Conf. on Peer-to-Peer Computing*, 2009.

[6] K. Graffi, "Project Group 2011 A Distributed Framework for Social Networks," 2011.

[7] A. Payman, "Project Group 2012 A Peer-to-Peer Framework for Social Networks," 2012.

[8] A. Christy, "A Peer-to-Peer Framework for Social Networks Project Group 2013 ," 2013.

[9] P. Weifeng and B. Dheeraj, "Systematic Evaluation of a Decentralized Social Network," 2015.

[10] L. Kangping, "Benchmarking a Distributed Framework for Social Networks," 2013.

[11] K. Graffi and C. Groß, "LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks," in *IEEE CCNC '11: Proc. of the IEEE Consumer Communications and Networking Conf.*, 2011.

[12] K. Graffi and C. Groß, "LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks," in *IEEE P2P '10: Proc. of the Int. Conf. on Peer-to-Peer Computing*, 2010.

[13] A. Butterfield and G. E. Ngondi, "distributed system." [Online].

Available: http://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975-e-1499

[14] B. Carlsson and R. Gustavsson, "The Rise and Fall of Napster - An Evolutionary Approach," in *Active Media Technology*, 2001.

[15] J. Pouwelse and P. Garbacki, "The bittorrent p2p file-sharing system: Measurements and analysis," in *International Workshop on Peer-to-Peer Systems*, 2005.

[16] 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: Proc. of the Int. Conf. on Distributed Systems Platforms*, ser. LNCS, vol. 2218. Springer, 2001.

[17] A. Ippisch and K. Graffi, "Infrastructure mode based opportunistic networks on android devices," in *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, 2017.

[18] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[19] A. Schill and T. Springer, 2012.

[20] B. Leuf, *Peer to peer: Collaboration and sharing over the internet*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[21] R. Kwok, *Peer-to-peer computing: Applications, architecture, protocols, and challenges*. Crc Press, 2011.

[22] Rice University, Houston, USA, "FreePastry," http://www.freepastry.org/FreePastry/.

[23] 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: Proc. of the Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001.

[24] K. Loesing and S. Kaffille, "OpenChord," http://sourceforge.net/projects/open-chord/, 2006.

[25] P. Druschel and A. I. T. Rowstron, "PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility," in *IEEE HotOS '01: Proc. of the Workshop on Hot Topics in Operating Systems*, 2001.

[26] K. Graffi and A. Kovacevic, "Skyeye.kom: An information management over-overlay for get-

ting the oracle view on structured p2p systems," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, 2008.

[27] R. S. Hall and K. Pauls, "Osgi in action," *Creating Modular Applications in Java*, 2011.

[28] T. Jennings, "Real-time communication in web-browsers (rtcweb)," 2017.

[29] B. Bergkvist, "Webrtc, real-time communication between browsers," 2017.

[30] K. Graffi and A. Disterhöft, "Skyeye: A tree-based peer-to-peer monitoring approach," *Pervasive and Mobile Computing*, 2017.

[31] M. Garrels, *Bash Guide for Beginners*, 2010.

[32] M. Garrels, *Introduction to Linux*, 2010.

[33] I. Ross, "R Programming Language," 2018.

[34] F. Schulte, "Public testing of a peer-to-peer based social network application," 2015.

[35] HILBERT, "High Performance Computing," 2018. [Online]. Available: https://wiki.hhu.de/display/HPC/

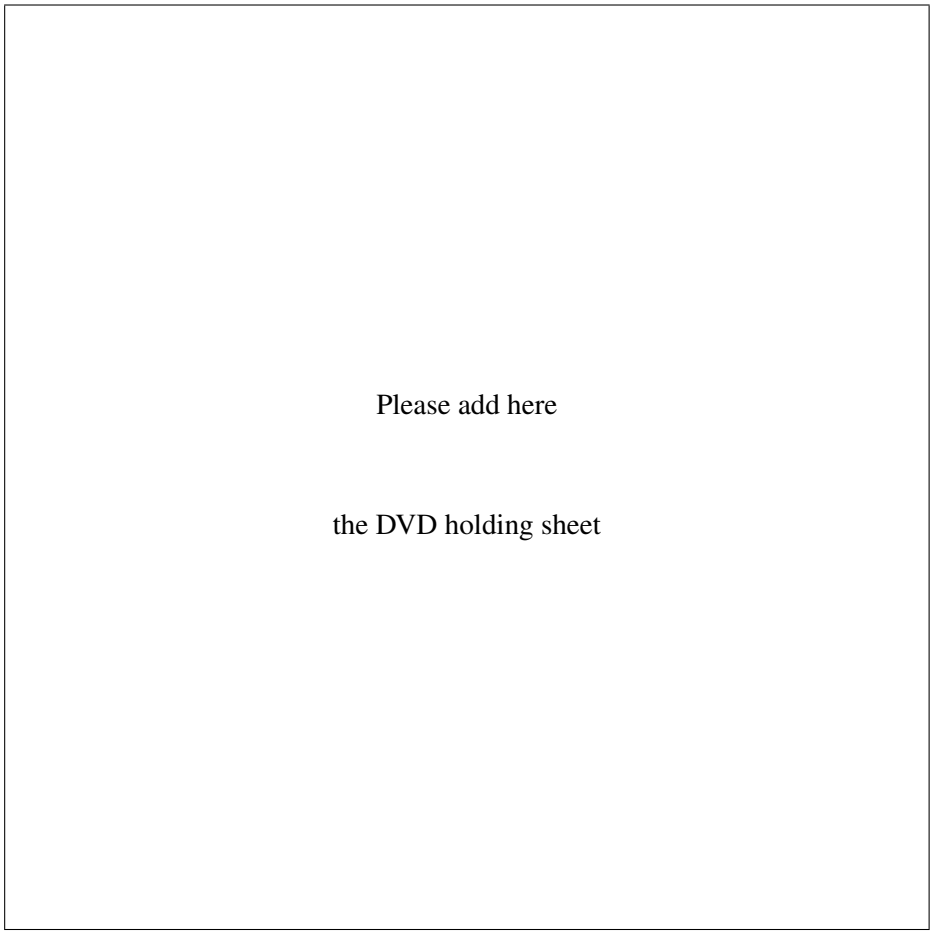
[36] Stackoverflow.com, "OSGi connection with ssh," 2018. [Online]. Available: https://stackoverflow.com/questions/30590137/apache-felix-shell-with-ssh

[37] J. Turner and P. Crowley, "Supercharging planetlab: a high performance, multi-application, overlay network platform," *ACM SIGCOMM Computer Communication Review*, 2007.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 8.Juni 2018 Dlikman, Iakov Arkadjevic



This DVD contains:

- A *pdf* Version of this master thesis
- All LaTeXand graphic files that have been used, as well as the corresponding scripts
- **[adapt]** The source code of the software that was created during the master thesis
- **[adapt]** The measurement data that was created during the evaluation
- The referenced websites and papers