Freie Universität Berlin

Trust Aware Social Networking: A Distributed Storage System based on Social Trust and Geographical Proximity

Diplomarbeit

vorgelegt am: 22. Januar 2009

am Fachbereich für Informatik der Freien Universität Berlin

Name:	Oliver Schneider
Matrikelnummer:	3791276
Abgabedatum:	23. Januar 2009
Fachbereich:	Informatik
Erstgutachter:	Prof. Schiller, Freie Universität Berlin
Zweitgutachter:	Dr. Vidales, Deutsche Telekom Laboratories

Eidesstattliche Erklärung zur Diplomarbeit

Hiermit versichere ich, dass ich die Diplomarbeit - bei einer Gruppenarbeit meinen gekennzeichneten teil der Arbeit - selbständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst habe.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

Berlin, den 22. Januar 2009 Oliver Schneider

Contents

List of Figures

List of Tables

Glossary

DHT Distributed hash table, a type of distributed system that provides hash table-like functionality.

P2P Peer-to-peer, a type of ad-hoc computer network.

Node A node a participating system in a P2P network.

Peer A peer is equivalent to a node.

SN Social network, a network made of entities typically persons or organisations linked together through.

UI User interface, graphical interface for user interaction.

WBSN Web-based social network, a social network application that can be accessed via a web site.

Abstract

The popularity of online social network services such as Facebook and StudiVZ has been steadily increasing over the last years. User numbers of Facebook are expected to soon exceed the 150 Million user threshold. Thousands of new users register an account every day and fill the data bases of network providers with new private data. Reports on security breaches and new marketing concepts for user data diminish the trust placed in these providers.

In order to gain back the user's control over private data, Sonja Buchegger et al. [21] proposed to use a community-driven P2P solution for social network services. To date only few P2P implementations have been used. None of them incorporates security measures based social trust or provides strong availability qualities.

The present diploma thesis incorporates this idea and extends it by implementing it on DSL-Routers and adding a special consideration of social trust. I base trust on social relations to support secure data exchanges and storage. The scope of this thesis includes the analysis of the system requirements, the derived software architecture and a fully working prototype that fulfills these requirements. *Clique* is the implementation of this architecture and is presented and evaluated as part of this work. It is shown that the software architecture is feasible to implement and deploy on a DSL-Router using the Mono runtime. I demonstrate the possibility to replace existing online social networks by implementing alternative architectures, such as *Clique*. *Clique* is a complement to current solutions and allows users to manage their private data while keeping benefits such as availability and safe storage of the information.

Zusammenfassung

Die Popularität von Sozialen Netzwerk-Diensten wie Facebook und StudiVZ ist ungebrochen. Es wird erwartet, dass die Nutzerzahl von Facebook schon bald die 150 Millionen Marke durchbricht. Täglich registrieren sich tausende weitere Nutzer, welche die Datenbanken der Anbieter mit immer neuen privaten Daten füllen. Dass das in diese Anbieter gesetzte Vertrauen nicht gerechtfertigt ist, zeigen immer wieder auftretende Sicherheitslücken sowie die Einführung neuer Konzepte zur Vermarktung der Nutzerdaten.

Damit Benutzer die Kontrolle über Ihre Daten zurückerlangen können, schlugen Sonja Buchegger et al. [21] die Idee, zu einer durch Nutzer getragenen P2P Lösung, für einen Sozialen Netzwerk-Dienst, vor. Aktuell gibt es nur wenige Realisierung einer solchen P2P Lösung und keine umfasst erweiterte Schutzmaßnahmen basierend auf Vertrauen oder liefert hohe Verfügbarkeitsgarantien.

Die vorliegende Arbeit greift diese Idee auf und erweitert sie um den Aspekt, die Realisierung dieses Dienstes unter Verwendung eines DSL-Routers und der besonderen Beachtung von Vertrauen durchzuführen. Dieses Vertrauen basiert auf sozialen Bindungen und wird benutzt, um einen sicheren Austausch Daten und deren sicherer Speicherung zu gewährleisten. In diesem Zusammenhang beschreibt die Arbeit, die an ein solches System gestellten Anforderungen sowie die Architektur, die daraus abgeleitet wurde. *Clique*, die Realisierung dieser Architektur, wird vorgestellt sowie gezeigt, dass es Möglich ist eine solche Software-Architektur mit Hilfe der Mono Laufzeitumgebung zu realisieren und auf einem DSL-Router zu betreiben. Ich demonstriere die Möglichkeit einen existierenden Dienst für soziale Netzwerke durch eine alternative Architektur wie *Clique* zu ersetzen. *Clique* stellt dabei eine Ergänzung zu bestehenden Lösungen dar, die es Nutzern erlaubt ihre privaten Daten zu verwalten und dabei die Vorteile von hoher Verfügbarkeit und sicherer Speicherung beizubehalten.

CHAPTER 1

Introduction

Web-based social networks (WBSN) have become ever more popular and are steadily growing in size and interest. With user numbers exceeding tens of millions per network, the data stored within the network provides an economic opportunity for the companies running the network and third parties as well. With risk of exposure through data mining of network owners, or worse through security breaches within the network, users have become more and more aware of the risks of these networks regarding privacy and ownership of social data.

With this newly gained awareness, alternative approaches for creating next-generation online social network (OSN) services were discussed by Buchegger et al. [21] and they introduce the idea of using peer-to-peer infrastructure instead of centralistic client-server approaches as it was used in the past. This seminal paper poses several questions on how to structure the network, how to achieve sufficient performance and most importantly how to gain a high level of trust and security. In the course of this thesis a prototype implementation of such a P2P social network service, named *Clique*, was developed and evaluated.

1.1 Terminology

This work makes frequent use of technical terms which shall be introduced and defined here for a better understanding. The term **peer-to-peer (P2P)** system describes a decentralized network system where every participant contributes resources to the overall system and participants use diverse connectivity to exchange messages. Communication partners are named **peers** or **nodes** within such a network as the system does not have the notion of client and server. A **peer** is simultaneously a server and a client.

Another commonly used term is **availability** which describes the proportion of time a system is in a functioning condition. A functioning condition in the context of P2P systems means that a **node** is connectable and ready for interaction. **Availability** is mostly measured in percent of **uptime** over the period of a year with **uptime** being the time a **node** is operational. Favorable **availability** values begin at an uptime of 99% per year which still allows a **downtime** of more than 3 days per year.

Social networks (SN) are discussed in this work and describe a social structure of interconnected people. These connections stand for interdependencies as for instance friendship, trust or same interest. A special manifestation of a **SN** is the **WBSN** or **OSN**, which is a web application or service that allows people directly to form a social network by creating links to other persons and adding them as friends.

1.2 Motivation

With WBSNs having issues of trust and scalability, next generation OSN services have to eliminate these issues and still offer the same functionality. WBSN services handle the most private and personal data of users and should therefore provide security and integrity of their services. A study by the Frauenhofer Institute [32] examined how data privacy was enforced in major WBSN platforms. Researchers came to the conclusion that none of the tested platforms offers a satisfactory solution to ensure privacy. In addition to security concerns, the trust in companies behind such services is questionable as well. The provided service is connected to a financial interest which seeks to exploit user data in order to make money. This creates the opportunity develop a new service with these aims as a basis.

- Strengthen privacy by removing a central entity providing the service and limit the distribution of social data to a minimum and only trusted persons.
- Provide similar functionalities and quality of services guarantees as current WBSNs,

however, using an always-on networking infrastructure already available in most households: DSL routers.

A possible approach to ensure these qualities is a P2P implementation of an OSN service is to use the advanced uptime and connectivity properties DSL-Routers offer.

1.3 Contribution

Existing WBSN client-server architectures do not cope well with the increasing number of users which arouse scalability issues. This creates the demand for additional hardware that companies find funding for by exploiting their only asset, the users data. The presented thesis proposes a new approach to implement a P2P social network service using social trust that ensures data privacy and still provides comparable performance and availability as experienced in the today's WBSNs.

- My first contribution is the design and deployment of a prototype implementation of a P2P social network application. Different approaches have been discussed in previous publications; however these were focused on offering a client application to be run on mobile devices or desktop computers. This work's implementation is platform independent and can be deployed on a number of devices, but most importantly on DSL-Routers. The use of an embedded device for a more reliable service thus may be provided, based on the previous experience of the author in the field of Wi-Fi metropolitan networks and bandwidth sharing[1].
- Another contribution is the implementation of a replication scheme based on social trust and geographic distance that ensures data privacy and maximizes the failure tolerance. Although trust is discussed in other publications, an automatic direct trust measurement between trusting parties is barely examined.

1 M. Solarski, P. Vidales, O. Schneider, P. Zerfos, and J. Singh, An Experimental Evaluation of Urban Networking using IEEE 802.11 Technology . In Proceedings of the 1st IEEE Workshop on Operator-Assisted (Wireless Mesh) Community Networks (OpComm 2006), September 2006

- A further contribution is the evaluation of feasibility and effectiveness of using the platform independent Mono runtime to develop and deploy a service on top of a DSL-Router. This evaluation includes the assessment of the development process as well as performance measurements compared to native code.

1.4 Outline of the thesis

This thesis is outline as follows. Chapter 2 introduces the theoretical background and research context on which this work is based beginning with the underlying network paradigms and their application.

In chapter 3, a requirement analysis of the given problem is conducted and transformed into a design specification detailing all need components and subcomponents.

Chapter 4 presents details of the conducted implementation. This includes class overviews of all components as well as descriptions of their interaction. Finally, the designed protocol is described.

In chapter 5, the implementation and the development process are examined a) qualitatively by evaluating the effort it took to develop the prototype and b) quantitatively by presenting some benchmarks conducted with the system and the underlying runtime.

Chapter 6 concludes this work and presents ideas for future research and improvements.

CHAPTER 2

Theoretical background

This chapter generally introduces research findings and concepts incorporated into *Clique*. Strengths and limitations of these concepts will be pointed out, alternative solutions will be proposed.

I will specifically cater to the concept of distributed systems (section 2.1), introduce properties of a trustworthy system and discuss the concept of trust in current research (sections 2.2), and present a mean to improve performance and availability of networking system (section 2.3). The last two sections describe the basic principles of embedded systems (section 2.4) and social networks (section 2.5).

2.1 Distributed systems

For the present work, a P2P system is aimed to be implemented. As it is considered to be a distributed system, I will start off giving an overview on distributed systems. Coulouris defines a distributed system as a system "in which components located at networked computers communicate and coordinate their actions only by message passing" [25, p. 1]. Tanenbaum, on the contrary, describes a distributed system as a "...collection of independent computers that appear to the users of the system as a single computer" [50, p. 1]. The main objective of these systems is to share resources, e.g. storage space, computational time or any other action to achieve a common goal. As Tanenbaum's definition states, the distributed system hides the existence of individual entities that form the system by providing an abstraction layer.

Distributed systems and their construction have to cope with the challenges of heterogeneous components, openness, security, scalability, failure handling, concurrency and transparency. *Clique* faces the same challenges and deals with them accordingly. It hides the distributed nature of the system from the user and ensures scalability through the use of P2P methods. Protocols handling failures and concurrency are implemented additionally.

The implementation of *Clique* makes use the two paradigms, client-server and P2P, whose implementations constitute distributed systems. Both approaches will be discussed next.

2.1.1 Client-server paradigm

Tanenbaum [49] describes the *client-server paradigm* as a model with servers being 'powerful computers'. These computers are connected to clients through a network, while clients are 'simpler machines'. Client and server communicate by sending messages over the network. The client initiates the process by sending a request and then waits for a reply message from the server. After having received the request, the server performs the requested work or looks up the requested data and sends back a reply message. The server's main function is to provide a service while the client's purpose is to consume it. The communication used in client-server systems consists of two parts: a request and an answer to that request. Only clients can start this process by sending a request to a waiting server that will in return process the incoming request and answer it accordingly. The number of clients normally exceeds the number of servers which reduces processing load for clients but at the same time increases the server load. The client-server paradigm is an essential part of today's network infrastructure. Basically every network application uses it directly or modified. Well known examples in internet applications are e-Mail services, web services or domain name services. A limitation of this paradigm is the lack of scalability for large numbers of clients. With each new client the server needs more resources which can only be added to a certain extent, i.e. until the number of clients overwhelms a server. *Clique* is a hybrid system for which the lookup part was implemented as a server component. The main system, however, uses P2P methods (see section 2.1.2). These P2P methods offer an alternative approach to the client-server paradigm and avoid problems of scalability. They will be discussed in the following section.

2.1.2 Peer-to-peer paradigm

Clique uses the P2P paradigm and is therefore essential for this thesis. In contrast to client-server networks, in which a small number of servers provide resources and bandwidth for a larger number of clients, P2P systems constitute a network of interconnected peer nodes that combine their resources to achieve a common goal. "A P2P network distributes information among the member nodes instead of concentrating it at a single server" [40, p. 31]. In this setting every node is equal to its peer and acts as client and server simultaneously. Nodes make requests to other peers and answer incoming calls.

The main advantage of P2P systems over central client-server approaches is that, although every client creates additional demands for the system, each client also contributes more resources, thus improving the overall performance of the system. The client-server approach does not provide for clients to contribute resources to the system, therefore new clients only create additional demands. Availability and robustness against failure are increased in P2P systems as many peers equally provide service. P2P systems are still able to provide resources and functionality in case of node failure as other nodes can compensate for this failure. Central servers, on the contrary, remove service for all clients in case of server error and are hence a single point of failure.

P2P systems are mostly noted for application in file-sharing programs (e.g. BitTorrent [24] and eDonkey [31]), but are also used in applications for real-time communication between people (e.g. Voice or text chat).

Clique implements a P2P storage system that distributes social data among nodes and provides additional measures for fault tolerance, while being transparent to the user.

Peer-to-peer lookup

A further function that is needed for *Clique* is lookup. P2P systems consist of a large number of interconnected peers with dynamically changing IP addresses in a largely unstructured network. Keeping track of all changes within the network and providing lookup operations is essential for a P2P system in order to be operational. Peers have to look up the addresses of other peers to establish a connection. Moreover, lookup methods are used to implement search functionality. Searching the profiles of users is a common feature of WBSNs and should be considered when choosing a look up solution.

P2P solutions to this problem exist but cannot fully satisfy the requirements of providing guarantees for look up termination and range queries. Potential candidates are presented next along with thoughts on why they were not suitable for usage in the *Clique* system.

Gnutella [23] is an unstructured P2P system. In this context 'unstructured' means that nodes connect randomly to neighbors and build a random network. Lookup in Gnutella takes place by flooding random neighbors with lookup requests. This technique cannot guarantee that an item will be found. Chances even decrease if an item is unpopular. In a social network profiles will only be shared by a small amount of people which makes all profiles unpopular for Gnutella. A person's profile would therefore not be found.

Chord [47], Tapestry[52], Pastry [44] are implementation of distributed hash table (DHT) protocols. A DHT offers storage and lookup of name value pairs by applying a cryptographic hash like SHA-1 [14] to the name. The DHT implementation uses the resulting hash value to store and find the corresponding value. DHTs offer fast lookup methods and good load balancing properties but are limited to exact match searches. The absence of range queries limits the use of a social network, because looking people up is an essential part of WBSNs. This limitation makes DHTs no viable alternative.

P-Grid [16] is a DHT variant incorporating virtual distributed search tree to provide support for lexicographic key ordering and range queries. Although range queries are supported, the maintenance of the network as well as load balancing are quite complex. Currently, no deployment of P-Grid is available. As a result, the code is not very stable and has not been tested. Implementing this solution would be too strenuous. P-Grid was therefore not chosen to keep the system simpler, but could be a viable solution for future enhancements of *Clique*.

Peer-to-peer storage

Storage for social data, for instance for profiles or messages, is a critical part of *Clique*. The storage solution has to be secure, ensure privacy and be fault-tolerant. P2P storage solutions that present possible options are discussed next.

OceanStore [36] uses a P2P infrastructure to store files and to provide file system operations on them. Data is distributed among peers and can be cached everywhere. That improves latency and accessibility. Data security and integrity are assured by allowing encrypted storage. OceanStore does, however, not provide equal access to clients. The authors envision the network to be run by third party access providers who get paid to guarantee qualities such as availability and security. This semi-centralized design negates one of the main goals of *Clique*: a community-driven system that avoids data distribution to third parties.

Ivy [37] is a multi-user read/write P2P file system that uses DHash [26] to store data. Each user handles file updates by appending changes to their private log. This avoids the need for locking and conflict resolution for simultaneous access of files. Ivy does not handle encryption and distributes files in clear text to other users. This system does not control who accesses the data or where data is stored. Ivy therefore does not meet the privacy requirements of *Clique*.

Pastis [22] is similar to Ivy [37] as it uses a DHT (Pastry [44]) to provide routing and storage. It offers multi-user read/write operations, similar to Ivy, but makes use of the good locality properties of Pastry. Network access latency can thus be minimized, so Pastis shows better performance than OceanStore and Ivy. Pastis is able to regulate write access to a file, yet, does not prevent read access to it. In this scenario, privacy of the stored data cannot be ensured unless encryption is handled on the client side.

Though existing P2P storage solutions may be used to provide storage functionality in general, they involve client side encryption and create additional problems regarding authorization and performance. The present work aims to propose a new approach specializing on the fault tolerant storage of social network data while maintaining privacy.

2.2 Trustworthy systems

Trustworthiness becomes increasingly important in software and computer systems, especially with regard to steadily expanding application domains into all fields of daily life. Software and technical systems influence more and more our daily routines and increasingly handle private data. The success of software and its application depends on the extent to which we can trust it [19]. This section will briefly introduce the term trust (section 2.2.1), followed by an introduction to related trust research in computer sciences (section 2.2.2). Social trust and how it will be employed in the proposed system will be described at last (section 2.2.3).

2.2.1 Trust

The Oxford Dictionary of English defines trust as the "firm belief in the reliability, truth, or ability of someone or something" [46]. Trust is gained from past experience of entities' interactions. Therefore, a trustworthy entity typically brings a history of reliability, security and no failure during the course interaction; it provides service in a timely manner. These qualities also apply for computing systems or services. A trustworthy service will most importantly not disclose confidential information [19].

2.2.2 Trust in computer sciences

During the last years, the focus on trust has been increased in the domain of computer sciences in order to optimize service performance. Trust in this regard refers to security as well as promptness in the execution of processes. A specific algorithm for trust is EigenTrust [35]. It uses a variation of the PageRank algorithm [39] to determine a global trust rating for peers in a P2P network. Trust is inferred by an interaction between peers and their experience based on the ratio of good to bad files received from a host. That way peers without previous interaction experiences can request a trust rating from the network to estimate a nodes future performance.

In the context of social networks, the gained global trust rating displays only a small part of trust ratings. That is because the gained rating of EigenTrust focuses on performances values of peers such as uptime, response time and throughput. Social relations between nodes are not considered. In a social network only a small number of nodes are eligible communication partners, so that a direct measurement of trust seems more suitable than a global cooperative approach. Another concept discussed in computer science research is the inference of trust. Goldbeck [28] offers two algorithms to compute a trust rating from one person to another who are not directly connected in a social network. The proposed algorithms use trust rating requests that traverse to all the trusted persons of a social network and are returned to the request node along with a trust rating. For the presented work inferred trust is not essential, but may be used to extend the circle of trusted people in case it is needed.

2.2.3 Social trust

In this thesis social trust is defined as a trust measurement automatically derived from social data. To the best of my knowledge, the only research in this field was conducted by Motorola Inc., a mobile phone manufacturer, and published in a patent application in 2008 [48]. The patent proposes an index computed from statistical parameters that were derived from social data stored in a mobile phone. The statistical parameters included number of voice calls to a person made and received, call length and many other parameters derived from communication patterns. The parameters were weighted according to their importance. This principle will also be used in this work and will be applied to the social data stored in a WBSN (e.g. number of incoming and outgoing messages, wall entries).

2.3 Replication

The *Clique* system uses replication to improve system properties like availability and performance. Section 2.3.1 defines replication. The following sections 2.3.1 and 2.3.2 describe the kind of replication used in the proposed P2P system. The last section 2.3.3 illustrates two prominent examples for replication in computer sciences.

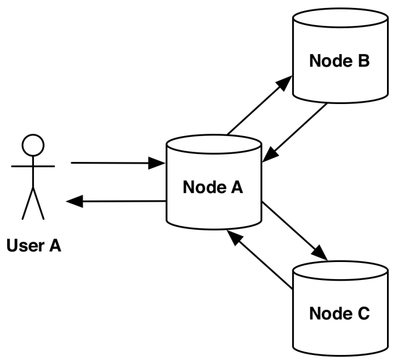
2.3.1 Definition

Replication of data refers to [...the maintenance of copies of data at multiple computers] [25, p. 1]. More general, a replica is an exact copy of an item or a resource with replication being the process of creating and maintaining such a replica. In computer sciences replication is used to provide additional copies of resources by exchanging data in order to ensure consistency between redundant resources. These resources are mainly copies of data items that are stored on different devices (replication in space), but also include resources like computation or services [1]. Besides replication in space, these services support replication in time, i.e. the ability to be run repetitively on the same device. Usually, uniform access to replicas is provided hiding the actual replication. Advantages gained through replication are improved reliability, fault-tolerance, accessibility and performance. Having several copies of an item decreases the chances of failure, as several instances are less likely to fail at the same time. That is also true for performance considering that workload and access the data can be distributed between several copies. Thus faster response times can be provided by choosing geographically closer replicas and a higher throughput can be achieved due to less loaded individual replicas. In order to keep all replicas in a consistent state, in which all replicas share the same data, protocols have to be put in place to synchronize each copy.

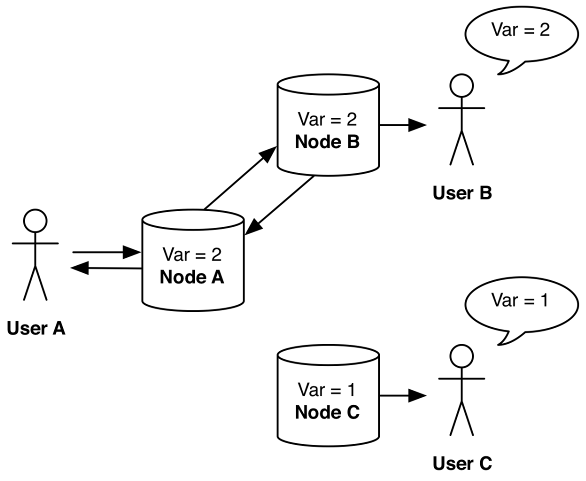
Update models

When choosing a replication scheme it is important to choose an update model that defines who is responsible for creating replicas and keeping the system in sync. While Read-Only operations are autonomic and can be executed in any order, update capability or write operation mandate additional effort to be taken to keep the system in a consistent state. Gray et al. [29] describe two properties that define a replication model: propagation and ownership. Updates to replicas can be propagated either *Eager*, i.e updating every replica directly before returning from an update request, or *Lazy*, i.e. by distributing the updates after a transaction was committed. *Eager* updates guarantee consistency but slow down response times, while *lazy* has to synchronize all updates to keep a consistent state. Propagation mode as well as object ownership influence how updates are organized in a replicated environment. Updates can be propagated in an eager manner by updating all replicas at the same time. Therefore all copies in the network are kept exactly synchronized as updates only succeed if all replicas could be updated. Lazy replication commits all changes locally, thus gaining responsiveness, and communicates local changes separately to other copies in the system. This way eager replication trades slower response times against consistent updates and avoids concurrency problems. Figure 1a illustrates an eager update: User A tries to do an update on Node A. Before User A receives a response on the success of his update, Node A has to contact und update Node B and C. All three nodes have to be successfully updated their data before Node A can confirm the successful update. Nodes A, B, and C have the same data at all times and respond with the same values. To demonstrate lazy propagation, figure 1b shows a possible update problem that may occur doing lazy updates. While User A gets an immediate response after the local update has finished, the changes need to be communicated to Node B and C and keep the system in an inconsistent state until all nodes were updated. In this state, concurrent requests can result in contradicting answers. In the case of figure 1b, User B and C receive different answers for the value of Var.

Object ownership is the second factor for choosing an update model. Two forms exist:



(a) Eager replication



(b) Lazy replication

Figure 1: Replication modes

master or group ownership. With master ownership only a single node has the right to change the data, while all other nodes provide read access only. Every update request has to be sent to the master node that holds the primary copy. With group ownership all node have equal rights on the data and can receive update request. With this equality every node has to communicate the changes to each member of the group.

Clique uses the lazy-master variant of replication. Social data can only be updated using the node that is solely responsible for it. If a node is offline, updates to that node will not be processed. All updates are stored at the primary node and are later lazily propagated to its mirrors in the system. This process ensures that though a node may become outdated it will never differ from the primary node, because simultaneous updates to several nodes are not allowed.

2.3.2 Representation

When designing a replication method it has to be decided how data will be represented in the system. Traditionally replicated files can be represented in three forms:

1. Whole file
2. Per-Block
3. Erasure codes

Whole file and Per-Block representations use original data and create redundancy by copying either the whole file to different nodes or divide the file into blocks that can be stored on different nodes separately. The third option erasure codes is a technique that divides an object into n fragments and encodes those fragments into m additional fragments with $n > m$. The resulting system is resilient to up to m fragment failures [43, 51]. Comparisons between conventional replication and erasure codes show that erasure codes require considerably lower storage costs to meet the desired availability level. Table 1 presents the approximated additional replication space needed to provide availabilities level between 0.800% and 0.999% for an estimated average short term node failure of 50%. The advantage of erasure codes in terms of storage and availability only deploys in an ideal environment that implies evenly distributed blocks on a large number of

Required Availability in %	Replication factor (Whole file replication)	Stretch factor (Erasure codes)
0.800	3	2.13
0.900	4	2.19
0.950	5	2.25
0.990	7	2.36
0.995	8	2.40
0.999	10	2.49

Table 1: Comparison of replica count for whole-file replication and error codes.

nodes. Error codes cannot cope well with a small amount of nodes available for replication. One of the main objectives of *Clique* is to provide as much privacy as possible. To achieve that, the amount of parties with access to a user's social data has to be kept to a minimum. With this in mind whole-file replication offers the best solution for usage in *Clique*.

2.3.3 Examples

RAID

The most common use of the concept of data replication is RAID (Redundant Array of Independent Disks) [41]. This concept describes the parallel use of a larger number of small disks to replace single large disks. In order to replace a single large disk the pool of smaller disks is connected to a RAID controller that transparently presents this pool of drives as a single, large and logical drive. The drive can be accessed as analogous to the original large disk. Incorporating RAID increases throughput of the storage system, because the parallel use of several drives distributes the load among the drives. That allows simultaneous access to data. Reliability can also be improved as RAID was designed to anticipate failure and offers several tradeoffs between additional storage and the capability to recover from failure.

Content delivery networks

Content delivery networks (CDNs) pose another important use of replicas to improve performance of a system. The purpose of CDNs is to distribute a selected type of data over a network. Enhancement of performance is approached by moving data from an origin server to replication servers on the edge of the network [42]. Data retrieved from local replica server typically has advantages in terms of latency and transfer rates compared to remote servers. Key problems are the placement of replication servers and object placement on these servers.

2.4 Embedded systems

DSL-Routers will be the main deployment target of *Clique* and belong the hardware category of embedded devices. This section describes the developments that made it possible to use Mono instead of highly optimized native code.

2.4.1 Evolution

Noergaard defines an embedded system as "[. . .] an applied computer system, as distinguished from other types of computer systems such as personal computers (PCs) or supercomputers." [38, p. 5] This definition doesn't provide much details by only stating that embedded systems are different from general purpose computers. Barr offers more details by describing embedded systems as "[. . .] a combination of computer hardware and software - and perhaps additional parts, either mechanical or electronic - designed to perform a dedicated function" [18, p. 1]. PCs or supercomputers are designed to be general purpose machines, able to tackle all kinds of problems while embedded systems are specifically designed to perform a special dedicated function. This restriction to certain problems is used to minimize production costs by only incorporating as much hardware resources as needed to solve the given problem. Hardware resources are processing power, memory or other hardware functionality. Confronted with such hardware limitations software was adapted. Software was scaled down in memory usage, code size and processing time to be effectively run on an embedded device.

Technological advances in production and product design made the term embedded device become more fluid. To date embedded devices have to solve more tasks and process more generic problems. This prompted a need for more hardware resources and higher layers of abstraction to efficiently implement all the required features. With increased resource supply, embedded platforms allowed to run new software that was considered to inefficient before.

2.4.2 Development

Developing software for an embedded system creates additional requirements to the development process as embedded devices are limited in memory and CPU performance. Therefore additional effort is required to minimize code size and memory usage. For this kind of optimization, native code (section 2.4.2) is the best choice as it offers most possibilities for optimizations but at the price of being very complex. When memory and performance are not the main problem, managed code (section 2.4.2) can offer a more easy way of developing software for embedded devices. The main arguments that go into a decision on what to use will be discussed now.

Native code

When developing native code for an embedded platform a complex development environment has to be set up in order to create native code for a foreign hardware platform. The process normally involves writing code in C/C++ and translating this code with the help of a cross compiler to the specific machine language used by the embedded system. Afterwards the machine code has to be deployed on the embedded device and run to see if it works.

Advantages:

- Optimizations for size and speed possible
- No VM overhead
- Real-time capability

- Direct hardware access
- Large number of development tools

Disadvantages:

- Complex debugging
- Code can be complex due to complexity of language
- Not platform independent
- Different compiler behavior

Native code offers mostly advantage hardware system engineers by saving money on hardware resources. Software developers on the contrary have to cope with a difficult development environment. Systems can be produced cheaper as native code can be optimized to consume less memory and CPU time but this comes with the price of complex development processes and platform dependence to a specific hardware.

Managed code

Managed code describes machine code that is independent from actual hardware and run in a virtual machine. Prominent examples of programming languages that use managed code are Microsoft .Net [2] and Sun Java [3]. Both Java and .Net runtime are also available as scaled down versions to better fit on embedded devices. Advantages:

- Portability
- Easy maintenance
- Large number of standard libraries
- Large number of development tools

Disadvantages:

- VM Overhead CPU and Memory
- No direct hardware access

Managed code improves the development process by offering a large amount of standard libraries and hiding error prone concepts such as memory management from the developer. Additionally the written code can be easily reused on different platforms. When memory and performance are not the main problem managed coded is a good alternative for producing reliable software.

Mono

The *Clique* system was implemented using C# [4] and Mono [5] as runtime. Some reasons for Mono are given below. Reasons for mono:

Portability: While managed code is platform independent the virtual machines running the code are not. Before one can use theses virtual machines they have to be ported to each new platform. In order to port such a virtual machine the source codes have to be available. These sources are then recompiled for the specific platform. At the time a virtual machine had to be ported to a DSL-Router platform both Sun and Microsoft didn't offer an open source version of their virtual machines. As an alternative Mono [5] offered an open source implementation of the Microsoft .Net runtime which is highly portable. Officially supported platforms for Mono are Linux, Mac OS X, BSD, and Microsoft Windows, including x86, x86-64, ARM, s390, PowerPC and more.

Multi language support: Many languages and runtimes qualify to be used on an embedded device but most support only to run one specific language. The .Net runtime on the contrary was aimed to support multiple languages from the start. Today more than 50 languages can be used to compile programs for .Net runtime [6, 7]. Include are languages such as C++, Java, Delphi and script languages like Python and Ruby.

IDE support: Integrated development environments can support software development a lot by offering functions like syntax highlighting, debugging, refactoring support. Visual Studio 2008 [8] was used in the development of *Clique*. It is known to be one of the best IDEs available, which specifically supports the C# language *Clique* was written in.

2.5 Social networks

The term social network was introduced 1954 [17] and describes a social structure made of nodes connected by edges that represent one or more specific types of interdependency. In this social context node represent individuals, groups or organizations while the connecting edges can be common relations like values, ideas, friendship, trade or other.

2.5.1 Web-based social networks

Web-based social networks are a special form of social networks that has grow in numbers and scope since the mid-1990s [28]. Today more than one billion user accounts have been registered to WBSN. This has created large opportunities as well as challenges for research in SNs. With the emergence of WBSNs large SNs could be studied for the first time without relying on randomized models or simulations to created large networks like this.

A user estimation made in 2005 [9] accounted for over 1 billion registered users in social networking sites. The top 10 social networks are shown in table 2.

Table 2: Top 10 of WBSN sites

No.	Name	# of Users
1.	MySpace	300.000.000
2.	Facebook	90.000.000
3.	Orkut	64.000.000
4.	ChinaRen Xiaonei	60.000.000
5.	Hi5	60.000.000
6.	Neopets	45.000.000
7.	Bebo	40.000.000
8.	Windows Live Spaces	40.000.000
9.	Xanga	40.000.000
10.	zoominfo	35.000.000
	Sum:	774.000.000

Facebook one of the biggest WBSNs announced December 2008 that they reached 140

Million users and 600.000 new users join the site everyday [10]. Even with it's over 800 servers supplying over 28 terabytes of memory [11] this massive growth creates scalability problems for Facebook.

Clique is meant to replace these systems and provide similar availability and better scalability properties.

2.5.2 Peer-to-peer social networks

Clique is not the first implementation of a P2P social network system. Two other approaches are presented next.

PeerSON

PeerSon focuses on providing a P2P social network infrastructure with special attention to security and privacy concerns [45] [21]. Main goal was to remove centralized entities and servers from the network owned by a single organization. This was achieved using a distributed hash table (OpenDHT [12]) as a lookup service and direct node communication afterwards. In order to increase availability implicitly replication is used. Clients cache all the entries received and make them available to other peers. To further increase the systems availability properties and uptime Schioeberg proposes the future use of DSL-Routers as additional network nodes.

Clique is a complement to this architecture by deploying the software on DSL-Routers and providing additional measures to increase availability and security based on trust.

MyNet

MyNet [34] provides a secure P2P social network service on top the Unmanaged Internet Architecture (UIA) [27] overlay. The focus is on providing pervasive access to social data by providing easy ways to share this data and resources among friends and devices uniformly and without delay. Each user imprints a device or cluster of devices with his profile and personal data and MyNet enables him to share this data in a secure fashion with his friends or colleagues.

In [33] Kalofonos et al. extended the MyNet design to a hybrid solution to eliminate of the some P2P limitations. With the device focused approach of MyNet no availability guaranties could be made as devices can be turned off a lot or not connected to the network. Moreover the solution was not integrated into existing Internet services. To overcome this virtual device were introduced that can be hosted on dedicated infrastructure that offers uptime and connectivity guarantees.

CHAPTER 3

System design

This chapter describes complete process of designing *Clique* starting with Section 3.1 detailing requirements analysis that went into creating *Clique* followed by Section 3.2 which contains the System architecture and a description of all its components that were derived from the requirements.

3.1 Requirements analysis

The software requirements analysis yields specifications that help in the software design process to create the underlying architecture and its components. Within this analysis process, functional requirements as well as non-functional are described. Functional requirements depict actual functionality that can be described as pairs of input sets, behavior and outputs while non-functional requirements specify criteria that can be used to judge a systems operation like performance, security, or reliability.

3.1.1 User requirements

Clique is intended to replace a normal WBSN consequently it has to offer the same basic functionalities. The use case depicted in Figure 2 covers the main functionalities that must be present in the system.

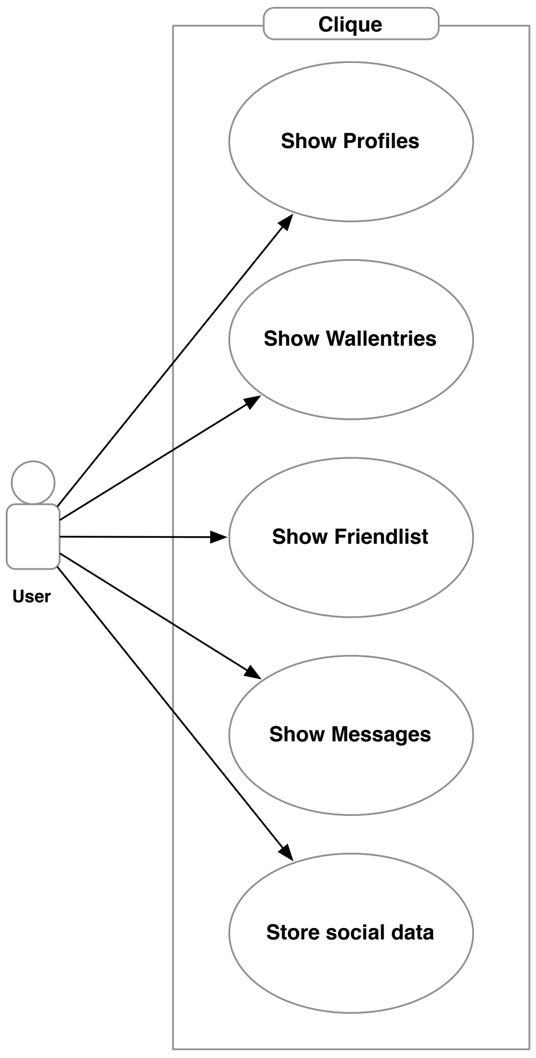


Figure 2: Use case user

3.1.2 Functional requirements

Show Profiles: User profiles store all the information a user wants to be known by his friends including data like home city, E-Mail address or relationship status. This information is not public and has to be accessible only by friends of the user.

Show Wallentries: Stored within a social profile are short messages that other users post on the profile page of a user. Wall entries are public and must be accessible for other users.

Show Friendlist: Connected to every user profile is a list of friends. Only friends have access to a users profile and wall entries. Friend lists are public and must be accessible for other users.

Show Messages: User profiles contain messages that were sent or received in the past. A user has to have access to these messages. Each user can only read his personal messages; access to other users must be prohibited.

Store social data: All social data consisting of user profile, wall entries, friend list and messages have to be stored permanently and be available for retrieval.

3.1.3 Non-functional requirements

Usability: Clique has to provide an intuitive and easy to use user interface which helps the user to interact with the underlying social network. This includes preventing the user from executing invalid actions and providing meaningful error messages in case of failure.

Availability: The system is composed of a large number of devices connected via the same number of network connections that could fail independently or simultaneously. Failure decreases system uptime and endangers system operations. This results in an extra demand for additional measures to counteract such failures.

Reliability: Besides availability *Clique* must prevent malfunction and defects with appropriate error handling and input validation. These actions have to anticipate unintentional as well as maliciously malformed input.

Performance: With software running on embedded devices extra attention to performance and efficiency has to be paid to performance as only limited resources are available. All operations should avoid complex computations as well as excessive memory consumption. Additionally bandwidth usage should be kept to a minimum. In cases of peaks in demand for a specific user profile, a node could be overwhelmed and therefore needs means to distribute load among nodes within the network.

Privacy: *Clique* handles most intimate and private user data and therefore privacy has to be ensured at all times. Access to data must be limited to valid peers that are allowed to request these data and distribution among other peers must be avoided if possible.

3.2 System Architecture

3.2.1 Overview

Clique is divided into three separate parts:

Backend: a lookup services for network nodes.

Client: a user interface for potential users.

Storage node: a service node providing the social network service and providing replication for fault tolerance.

While P2P networks can be designed to run without any central backend *Clique* is designed as a hybrid solution that incorporates a centralistic lookup service in order to keep the overall system simpler and avoid complex P2P lookup mechanisms. This way it is possible to concentrate on the storage node itself. The client was kept separated from the storage node to keep the implementation independent and flexible. A client could be written in any language or form as long as it implements the underlying protocol of the *Clique* social network. Each storage node is part of the overall P2P network infrastructure and is providing resources for whole system. All three parts are illustrated in figure 3. Shown is the network structure of the including its two communication pattern. Communication can be started by either clients or storage nodes. The backend only receives requests and does not initiate any communication on its own.

3.2.2 Backend

The backends main objective is to provide a lookup service. *Clique* requires a lookup service to acquire information on a specific node including its IP address, service port and location. This information is necessary to establish communication in an IP network

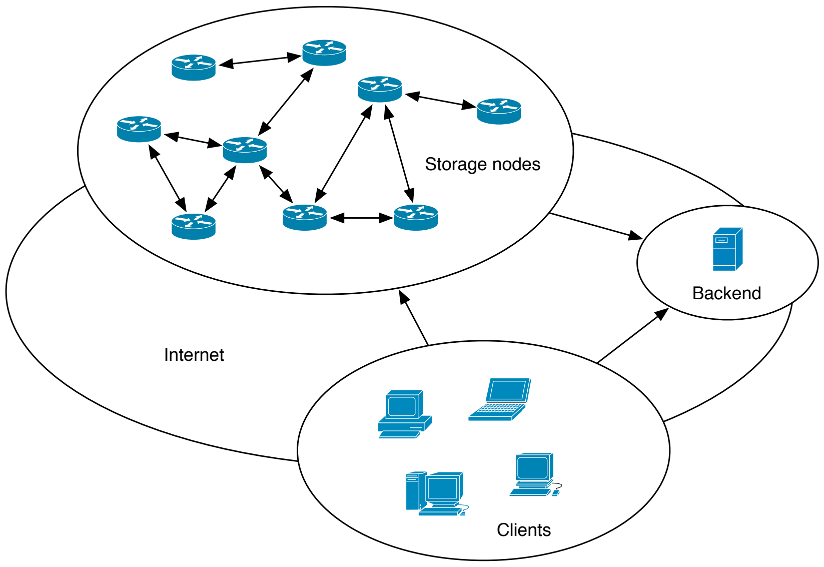


Figure 3: System overview

as communication partners have to know each other's IP addresses. This is necessary in order to contact each other. In a centralistic WBSN scenario this is done with the use of DNS servers which is appropriate when servers are used where IP addresses rarely change but due to the dynamic nature of P2P systems, where quick changes of IP addresses are common, a lightweight solution is more suitable. Additionally DNS was no designed to easily provide arbitrary data like location and service port to clients along with IP addresses.

The alternative of using a P2P lookup service was not pursued as although solutions for this problem exist they add unnecessary complexities.

Request dispatcher: *Client* and *Storage node* entities can send three different request types to the backend. The *Request dispatcher* analyzes incoming request and chooses the appropriate component to deal with it. Once analysis is done the *Request dispatcher* forwards the relevant information to the responsible handler which starts processing the request from thereon.

Login: Nodes need to log into the backend in order to register their current IP and Port configuration. This done be findable and provide other nodes with connection information. The login handler is responsible for storing this connection information

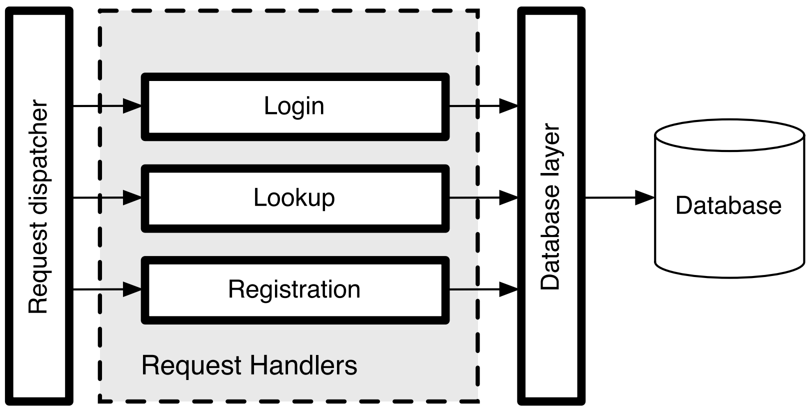


Figure 4: Backend architecture

in the database and replacing existing old information.

Registration: This component handles all requests regarding the registration of replicas. Requests are validated and if valid persisted into the database.

Lookup: When a node or a client wants to find another node the lookup component is used to retrieve the needed connection information stored by the *Login* component. Once an incoming request reaches the *Lookup* component it fetches the necessary information from the database and returns it to the requesting party.

Database layer: This component handles all the store and retrieve requests coming from the *Login*, *Registration* or *Lookup* component.

Figure 4 presents a diagram of the backend components.

3.2.3 Client

To provide the user with means of interaction with the system and a representation of the networks data a client is necessary. The client is responsible for retrieving data from the social network and feeding modifications back into it. Clients therefore have offer a graphical user interface that offers methods to communicate with the underlying social network layer to retrieve and display data.

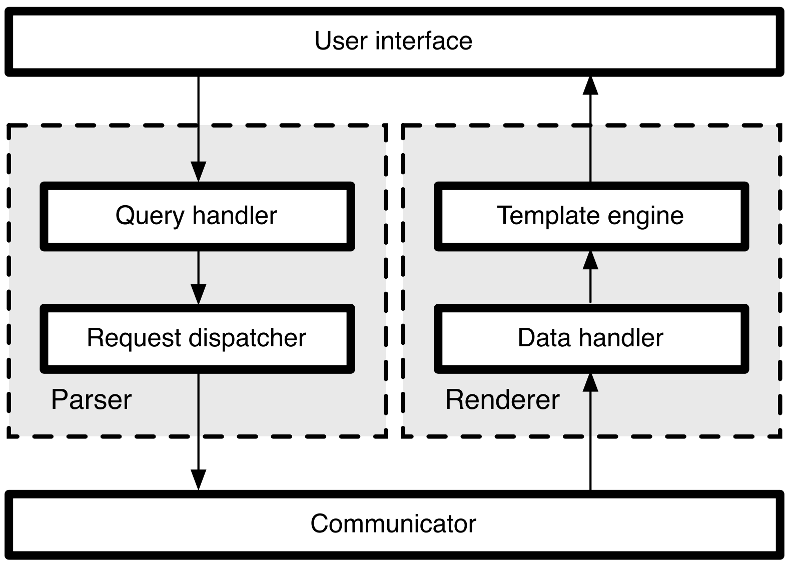


Figure 5: Client architecture

User interface: User interaction with the system can only be done with the *User interface* (UI). The UI presents the data by drawing it on a screen for the user to see. Moreover the user can interact with it by using input devices such as mouse or keyboard to start new requests. Possible requests are named in the User requirements (see section 3.1.1).

Query handler: User interface inaction results in requests that are send to the client. When such a request is formulated and sent to the client the *Query handler* will analyze the request and decided what information is needed. Once analysis is done the *Request dispatcher* forwards the extracted relevant information to the *Request dispatcher*.

Request dispatcher: Taking the prepared incoming information, the *Request dispatcher* has to retrieve the information necessary to connect a *Storage node* that can answer the Users request. For this the *Request dispatcher* queries the backend for the connection information of the node responsible for this information. This information contains the IP and Port of the master copy as well as information on possible mirrors. The *Request dispatcher* then translates the data into a format that is ready for transmission and forwards this package of connection information and data to the *Communicator*.

Communicator: Actual network traffic is handled by the *Communicator* component which is responsible for sending and retrieving data on the network level. It takes incoming data and translates it to a binary representation useable in network communication.

Data handler: Data retrieved from the *Communicator* gets forwarded to a registered *Data handler* which in return process it the incoming data. Upon the incoming data type a template is chosen for displaying a Profile or Messages. The template and data is then forwarded to the *Template engine*.

Template engine: Templates have specific placeholders for inserting data. The *Template engine* takes these placeholders an replaces them with the provided data from the *Data handler*. The resulting representation is returned to the *User interface*.

Figure 5 illustrates the client architecture.

3.2.4 Storage node

Storage nodes are the components responsible for storing profile data, regulate and grant access to the profiles and enhance system reliability and performance by actively creating replicas. Each storage node is responsible for a single user profile and its related data such as messages and wall entries. The proposed architecture and its components can be seen in figure 6.

Service Worker: The *ServiceWorker* is the server part of the P2P design and handles all incoming requests. It provides a network interface for incoming requests and answers those requests accordingly. Those requests are either store and retrieve request for social data posted by clients or replication requests from other storage nodes. To handle both types of request the *ServiceWorker* will use the *StorageManger* to store or retrieve the requested data.

Replication Worker: In order to improve the systems availability replicas have to be distributed among the storage nodes to compensate for failure. The *ReplicationWorker*

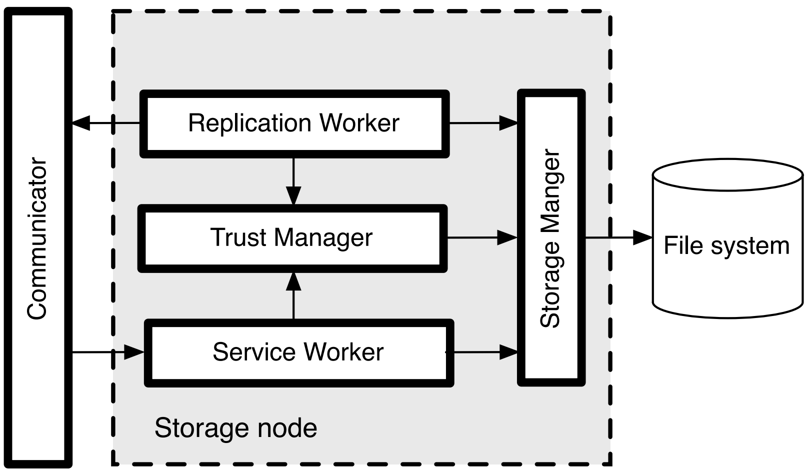


Figure 6: Storage node architecture

handles this task by requesting trust assessments for all friends in the user's friend list from the *TrustManager* and choosing the most trusted friends as mirrors for the local profile. Once the nodes are chosen the *ReplicationManager* starts sending replicas to theses nodes and after success registers the nodes as replicas in the backend system.

Trust Manager: Analyzing locally store social data and calculation a trust index is the main task of the *TrustManager*. Additionally to the locally stored data the trust manager requests the geographical location of a user and calculates the distances between the main user's location and a friend as supplementary information to better assess closeness between two users. This assessment is uses by the*ReplicationWorker* to choose the most trusted replication location.

Storage Manager: The *StorageManager* acts as an abstraction layer between objects like a user profile and the local file system. While all components work with objects those objects have to store when they are currently not needed. The *StorageManager* offers methods for this, methods to store and retrieve objects to and from the local files system. Receiving an object for storage the *StorageManager* will serialize the object to a byte stream and write it to disk.

CHAPTER 4

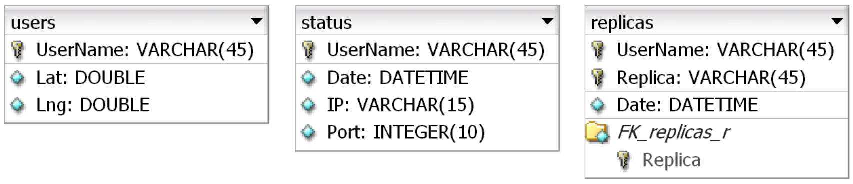
Implementation

This chapter will detail the implementation work that went into *Clique*. Section 4.1 describes the components included in *Clique* and the classes used to build them. Section 4.2 describes the main processes of *Clique* that are involved in providing the P2P social network service. Section 4.3 closes this chapter by explaining the application protocol underlying the *Clique* service.

4.1 Components

4.1.1 Backend

The backend was implemented as a Representational state transfer (REST) web service using the PHP programming language. A REST web service is executed by accessing a specific URL on a web server, in this case Apache. Each parameter that needs to be passed on to the function is embedded into the URL. The web service then returns a XML file as a response. Figure 7 shows the underlying database structure consisting of three tables users, status and replicas. The database only stores the minimal amount of data necessary to provide the lookup functions described in section 3.2.2. For each user a username combined with a geographic location is stored. These values for location have to be predefined and could be provided by an ISP or users themselves. The stored username is connected to a single Clique storage node. This username is used by storage nodes to log into the backend and registering on startup. The information sent at login is stored in the status table which thereafter contains the current IP and Port configuration. When a storage node registers a replica this information is stored as a pair of usernames in the replica table. To make the database access easier two table views were created, shown in



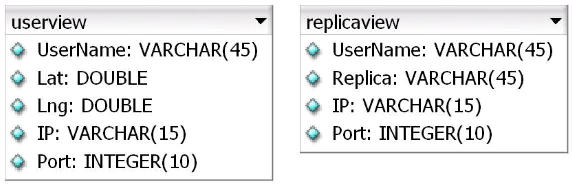
users
UserName: VARCHAR(45)
Lat: DOUBLE
Lng: DOUBLE

status
UserName: VARCHAR(45)
Date: DATETIME
IP: VARCHAR(15)
Port: INTEGER(10)

replicas
UserName: VARCHAR(45)
Replica: VARCHAR(45)
Date: DATETIME
FK_replicas_r
Replica

Figure 7: Backend tables

Figure 8. Both views use the user table and join it with either status or replicas table. This union simplifies selection statements by avoiding an additional join operation in each selection statement.



userview
UserName: VARCHAR(45)
Lat: DOUBLE
Lng: DOUBLE
IP: VARCHAR(15)
Port: INTEGER(10)

replicaview
UserName: VARCHAR(45)
Replica: VARCHAR(45)
IP: VARCHAR(15)
Port: INTEGER(10)

Figure 8: Backend views

4.1.2 Social client

Generally DSL-Routers already provide an embedded web server to give users a web interface for configuration, taking this as an opportunity the social client was implemented as a small web application that can be run in any web server capable of executing an external program. The Client application itself is a console application that takes arguments from the web server and output html code that is in returned displayed by the web server.

The actual classes that compose the social client are presented in Figure 9. As can be seen all the presented classes reflect the components described the design section 3.2.3. It contains Parser, RequestDispatcher, DataHandler and TemplateEngine.

In the process of executing the social client, the main program passes the incoming arguments to the ParseArguments() function of the Parser class, which will then check and extract the arguments. Depended on the arguments found, the responsible request method from the RequestDispatcher is called. The RequestDispatcher prepares a request and registers a DataHandler method for handling the response that will follow the request. After sending the request and receiving the answer the DataHandler method will extract the incoming data and call a render method from the TemplateEngine, that responsible for the received type of data. The implementation itself was kept simple and follows the

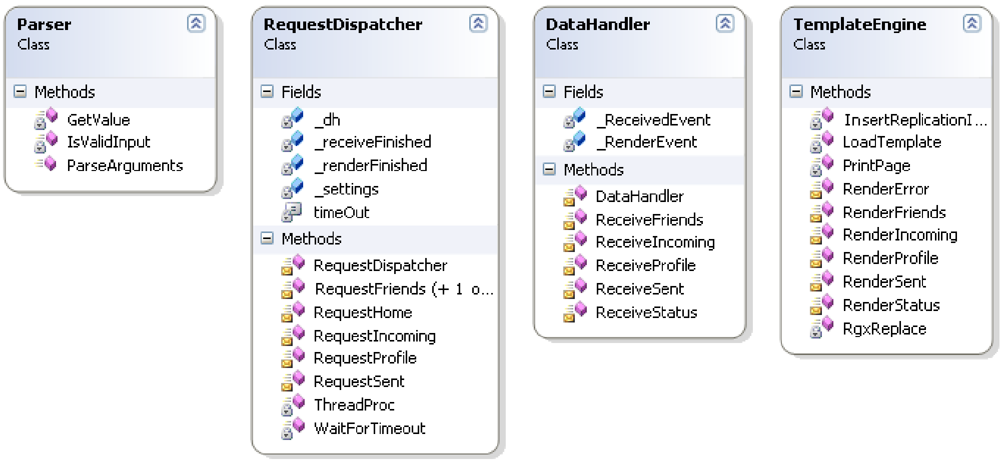


Figure 9: Class diagram of the Client implemenation

design described earlier. Socio clients receive requests, parse those requests, and decide which node to contact in order to retrieve the requested data.

4.1.3 Storage node

Storage nodes are more complex compared to social clients as they have to combine client and server capabilities at the same time. Each component defined in the design chapter was mapped to a class in the implementation and is shown in figure 10. The following classes are illustrated: ServiceWorker, TrustManager, ReplicationWorker and StorageManger.

Both worker classes utilize the manager classes when executing their individual code paths.

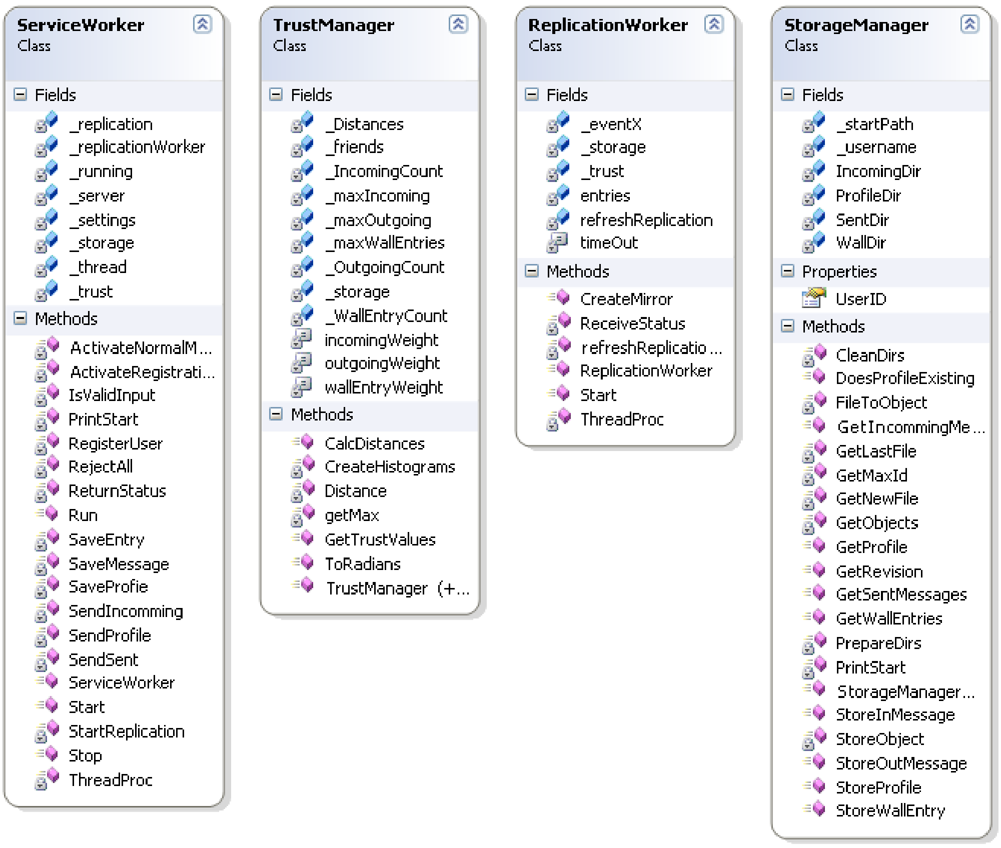


Figure 10: Class diagram of the storage node implementation

4.1.4 Communicator

The communicator component bundles all classes responsible for network communication. This includes inter-node TCP/IP communication in the P2P network part but also HTTP connections for the backend. Figure 11 shows the main communicatin classes. The represented ConnectionWorker class is responsible for inter-node and client-node communication while the HttpUtils class offers methods to access the backend over HTTP.

Objects

The Communicator contains objects that serve as a common base for exchanging data between client components and storage nodes. Having those objects bundle in the communication layer makes sure, that every component uses the same data representation and is understood by all other components. Included are objects such as Profile, Message,

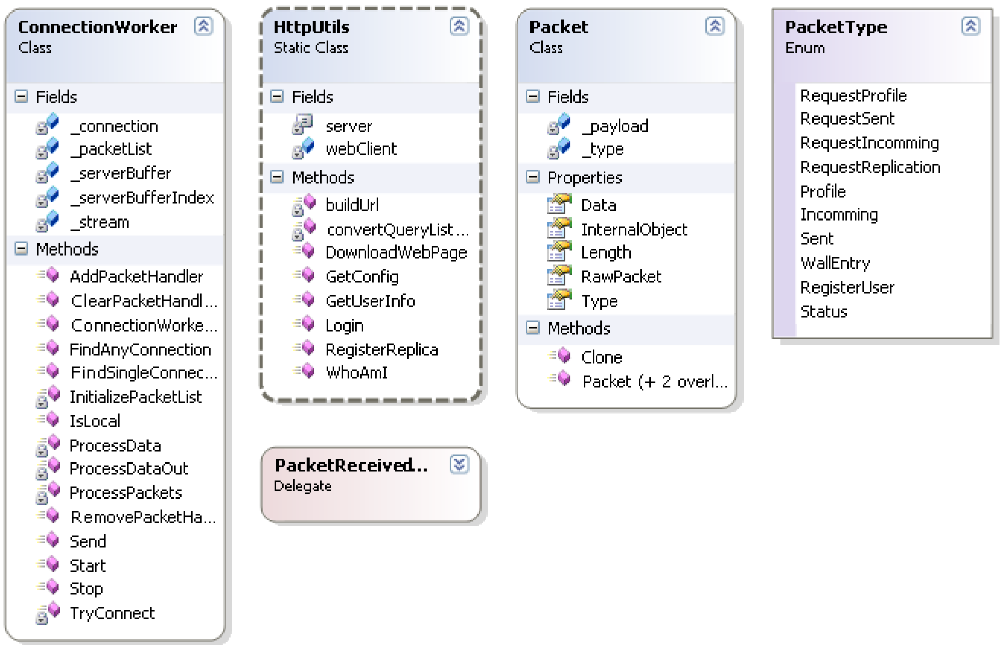


Figure 11: Class diagram of the communicator implementation

WallEntry.

Packet

Next to the classes responsible for communication Figure 11 shows the Packet class. The packet class is used as an abstract layer. Each packet stores an object and provides methods to convert this object into a binary representation. This representation can be sent over a TCP connection later be restored back into an object. Another field in the packet class is the packet type which is used by the DataHandler to identify the serialized object stored within the packet.

The binary representation has a structure of its own and was intentionally kept simple. It is illustrated in figure 12. As can be seen the packet structure only contains the packet

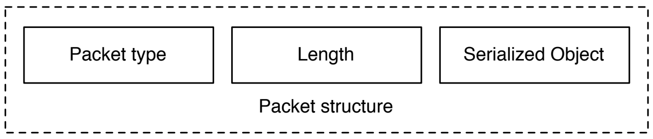


Figure 12: Structure of the utilized packet format

type, the packet length and the binary representation of an object. The length is important for handling the incoming packets in order to recognize the end a packet transmission. Serializing data objects to a binary representation enables the packet structure to be generic and transport any kind of information by utilizing only one kind of packet. Only the packet type has to be changed while the structure stays the same. As an alternative it would have been possible to create a generic packet and use C# inheritance to create a special packet for each data type. This way packet and data would be combined and only useful for network communication. The separation of data objects and packets user here made it possible to reuse the data objects within the storage manager implementation. This way communicator and storage manger use the same data objects for communication and to save information to the file system.

4.2 Processes

This sections details the main processes of each component and how it was implemented. Backend processes will not be included as these cannot be triggered by the backend itself.

4.2.1 Social client processes

Startup

Social clients are started by a web server which was previously triggered by a user's browser. Figure 13 presents the complete process of user, web server, client interaction. The browser triggers the web server by accessing an URL. The triggered web server will start the social client and forward the submitted arguments. Based on these arguments the client will fetch data return this data as a HTML page, which is returned to the browser.

Client communication

A client pure purpose is to retrieve and display data from the social network to the user. All communication requests originating from the client are user initiated. Exemplary for the general communication patter of a client the process of retrieving a user profile is depicted in Figure 14. Upon initiation the client will ask the backend for the current

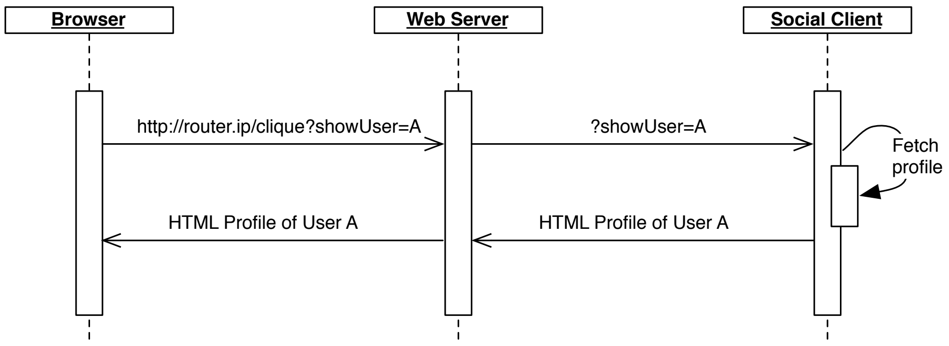


Figure 13: Sequence diagram of the Client Webserver interaction

connection information of the node responsible for the data the user wants to retrieve. In case of Figure 14 the Profile of user "A" is requested. When the connection information was received this information is used to connect the actual node and retrieves the social data. The process is analog for all *client* actions.

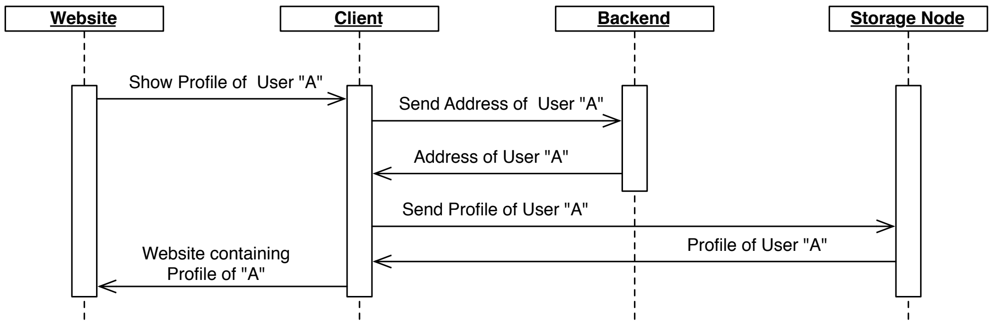


Figure 14: Sequence diagram for requesting a user profile

4.2.2 Storage node processes

Storage node communication

Next to its passive role of answering requests from clients, storage nodes take an active role in distributing replicas of their main social profile. Similar to clients, nodes have to retrieve a nodes connection information from the backend, before being able to establish a connection. Once this information was retrieved the replica is send to the other node. Upon success the node registers the new replica in the backend for other nodes to see and use when needed. This process is shown in Figure 15.

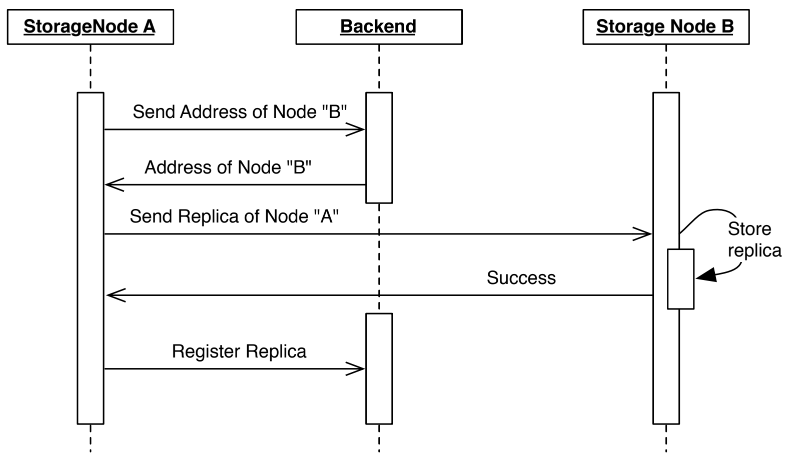


Figure 15: Sequence diagram for creating a replica on another node

Server functions

Storage nodes are capable to accept several simultaneous connections enabling storage nodes to service a multitude of incoming requests at the same time. This is done by the Service Worker who accepts incoming connections and places the handling routines for each individual connection into its own thread. In order to keep a node from being overrun by incoming connections a thread pool was employed that only accepts incoming connections when enough threads are available. Figure 16 depicts this process. Incoming connections are forwarded to the thread pool which will automatically put incoming connections into a waiting list in case no threads are available at the time.

Trust Calculation

As describes in Section 3.2.4 the analysis of locally stored social data is done by the Trust Manger. With after this analysis the Trust Manager calculates a trust index that other components can retrieve and use for their decision making. The algorithm works as follows:

1. Trust Manger creates histograms of Wall entries, Incoming and Outgoing messages per user.

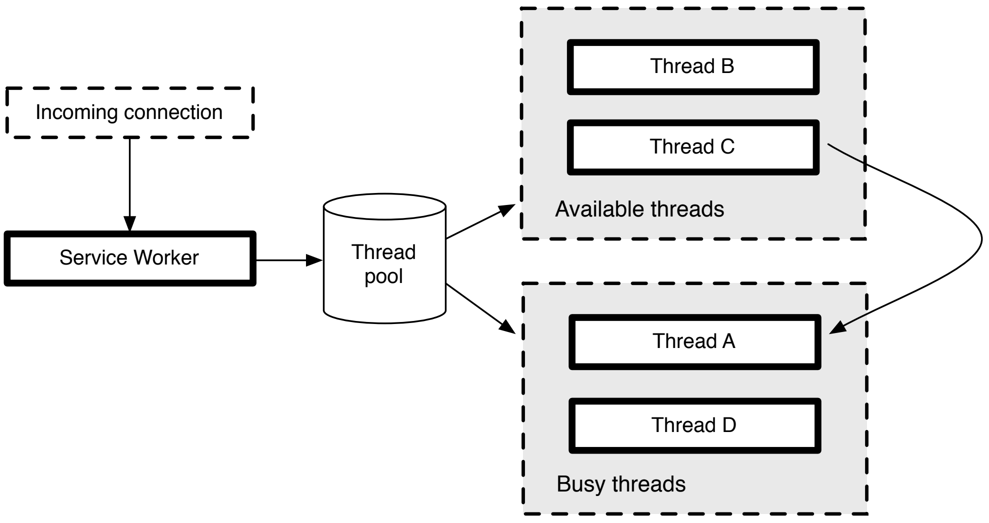


Figure 16: Diagram of the thread pool work

2. Trust Manger finds the overall maximum number of wall entries, incoming and outgoing messages send or created by a single user.
3. For each user a ratio of wall entries created by an user divided by maximum number of wall entries created is calculated and multiplied with its weight of 0.2.
4. For each user a ratio of incoming messages sent by this user and the maximum number of incoming messages sent is calculated and multiplied with its weight of 0.4.
5. For each user a ratio of outgoing messages sent by this user and the maximum number of outgoing messages sent is calculated and multiplied with its weight of 0.4.
6. The resulting trust index per use is the sum of all three ratio calculated in Step 3-5.

The proposed weights of 0.2 for wall entries and 0.4 for private messages are based on the assumption that private message indicate a higher sense of privacy and therefore need to be valued more in calculation of the trust index. Wall entries are publicly readable and used generally in a more casual context than private messages.

Additionally the weights are used to limit gained trust values to the range of $0 - 100$ which makes them more comparable.

Replication

The Replication Worker is responsible for distributing the local data to additional nodes to provide for extra fault tolerance in case the main node fails. The replication process is started immediately after the application is loaded. The process works like this:

1. A list of friends and trust indices is retrieved from the Trust Manger.
2. The list of friends is sorted according to their trust value.
3. The number needed replicas is calculated by dividing the number of friends by three and rounding the result up.
4. For the number of calculated replicas, replicas are created, starting with the most trusted friends from the list. With each round of the loop, it is tracked if a predefined threshold distance of 5km was reached.
5. If the calculated amount of replicas was created but the threshold distance was not exceeded the list of friends is traversed to the end, looking for a friend residing outside the 5km threshold. If found, an additional replica is created.

The presented algorithm makes sure, that only the most trusted nodes are chosen as a replication endpoint. Exceptions are only made in case the threshold criterion was not met. The threshold of 5km was chosen because most DSL-Failures are locally related. This grants the assumption that storage nodes outside of this 5km rage are less likely to fail simultaneously and offer increased availability. This allows the search to be widened to find an additional node that meets the criterion although it might not be the most trusted friend.

4.3 Protocol

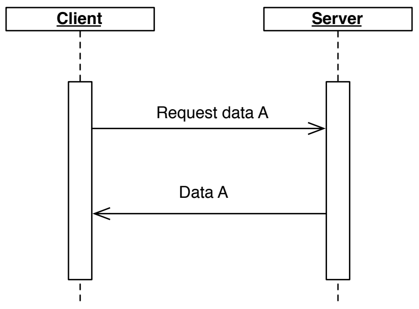
As underlying transport protocol, used by the Communicator, TCP/IP was chosen. This way the implemented application protocol on top of TCP/IP could be kept simple, TCP/IP ensuring that transmitted packets are received properly or an error will be given in case of failure. This way additional save guards that would have been necessary an UDP protocol implementation could be omitted.

The protocol only has two types of communication scenarios:

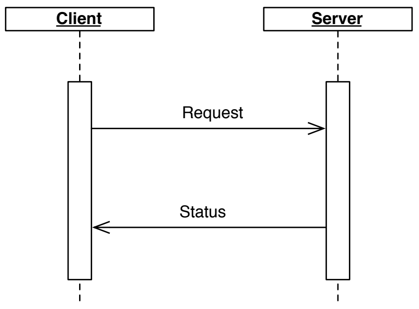
- Requests
- Replication

4.3.1 Requests

Requests are initiated from a social client and normally meant to retrieve data like a user profile or messages. After a connection was made the client sends a request to the server and waits for the requested data. If this request can be successfully served by the server the data is return as seen in Figure 17a. In case of failure like the requested profile is not stored at the connected server a status message is returned. Instead of the data the status message contains the reason of the failure. This is depicted in Figure 17b. Normally a connection is closed after a request was answered.



(a) Successful request



(b) Failed request

Figure 17: Sequence diagram of both request variants

4.3.2 Replication

The replication process is started the same way as described in section 4.3.1 by issuing a request to the server but contrary to a normal request, the established connection is kept open. After the replication request is answered with a positive status message the client begins to send the data expecting an affirmative status message after every item that was send. After all items were sent the connection is closed. This process is presented in Figured 18.

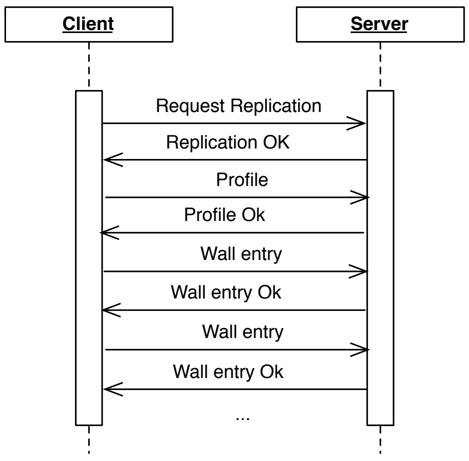


Figure 18: Sequence diagram of a replication process

CHAPTER 5

Evaluation

This chapter presents a comprehensive evaluation of the *Clique* system and the development process that went into it.

5.1 Feasibility of deployment

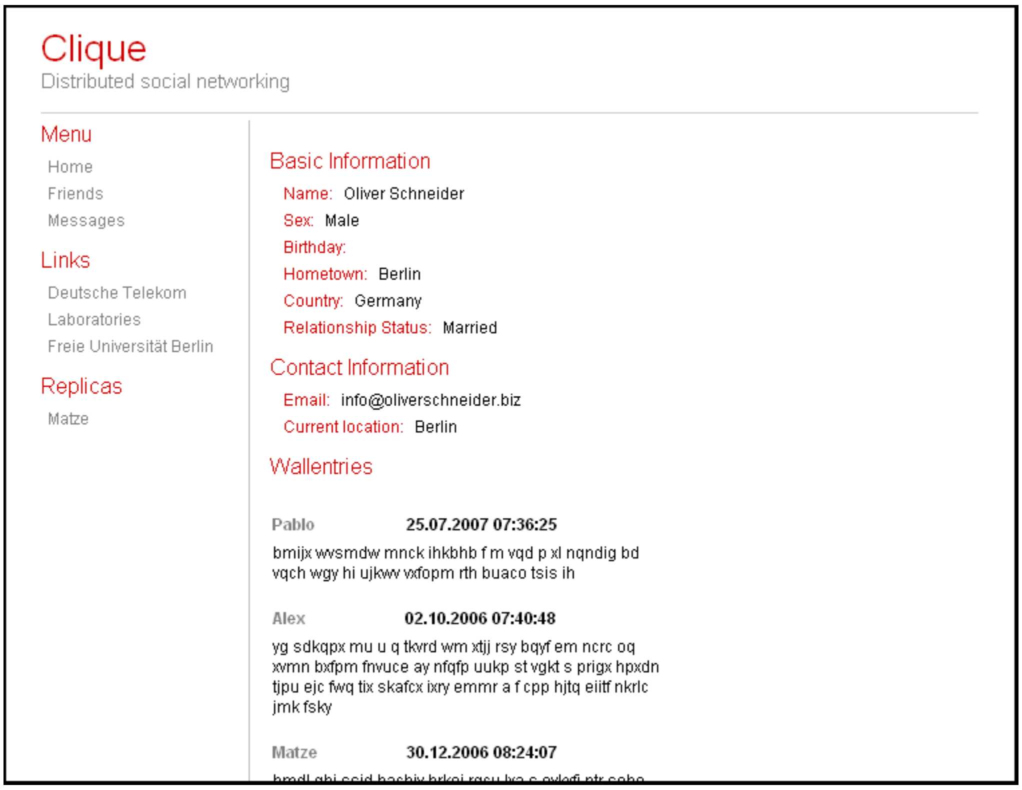
One part of this work was to evaluate the feasibility of deploying software on embedded devices such as DSL-Routers using a managed code runtime. After developing an application and testing it on a consumer DSL-Router I am able conclude that such a deployment is a viable option.

When performance and real time responses are not of the essence, Mono offers a simpler development process compared to tradition C/C++ projects with comparable performance. Mono can enable third party developers to develop embedded applications who wouldn't have considered writing programs or services for such a platform before.

5.1.1 Demo

As part of the evaluation a demo case was designed to present the implemented aspects in a real environment. The proposed scenario requires four components to be present: a backend, two devices A and B (running *Clique*) and a computer to access *Clique*.

In the initial state of the demonstration the backend and device A *Clique* are started. Once device A is fully operational and has registered in the backend, the computer is used access the profile stored on device A. Figure 19 shows an example profile created for this demonstration. As described before the socio client is implemented as a web application hence the web page presenting the profile. By clicking on existing links such as 'messages '



Clique
Distributed social networking

Menu
Home
Friends
Messages

Links
Deutsche Telekom
Laboratories
Freie Universität Berlin

Replicas
Matze

Basic Information
Name: Oliver Schneider
Sex: Male
Birthday:
Hometown: Berlin
Country: Germany
Relationship Status: Married

Contact Information
Email: info@oliverschneider.biz
Current location: Berlin

Wallentries

Pablo 25.07.2007 07:36:25
bmijx wvsmdw mnck ihkbhb f m vqd p xl nqndig bd
vqch wgy hi ujkwv vxfopm rth buaco tsis ih

Alex 02.10.2006 07:40:48
yg sdkqpx mu u q tkvrd wm xtjj rsy bqyf em ncrc oq
xvmn bxfpm fnvuce ay nfqfp uukp st vgkt s prigx hpxdn
tjpu ejc fwq tix skafcx ixry emmr a f cpp hjtq eiitf nkrlc
jmk fsky

Matze 30.12.2006 08:24:07

Figure 19: Social client profile view

or 'friends' the profile can be explored. At this point only the profile stored on device A will be accessible as device B is still turned off. Trying to access the profile on device B will result in an error message.

Now the second device can be turned on in order to demonstrate the interaction of two nodes. After device B is finished starting, a new try to access the profile of device B will succeed. Moreover device B will start to replicate its data to device A. After this is done device B can be turned off again, ensuring that profile B cannot be accessed directly. Refreshing profile B will still show the profile as the replicated copy is used instead of the original. The profile will still show although device B is not available.

This demonstration case can be used to presents the features of storing and retrieving social data from *Clique* as well as the replication and fail-over process.

5.2 Usability (ease of programming)

5.2.1 Maintenance

Maintenance can be improved by reducing the number of lines of code (LOC), as this has a positive impact on the understandability and usability of code [30]. Each line of additional code increases the chance of error. Therefore the reduction of LOC is likely to improve an applications quality and maintainability by reducing complexity and chance of faults.

Visual Studio 2008 was used to calculate the LOCs of all *Clique* components. Table 3 presents these results. Each component has less than 400 LOCs resulting in a sum of only 1.000 LOCs. These values show that *Clique* can be considered to be a small project which can be easily maintained and extended.

5.2.2 Debugging

Debugging is an essential part of software development and clearly shows the advantages of using Mono over cross compiled native code. While native code is depended on special debugging servers or expensive hardware debuggers, *Clique* code was able benefit from be being platform independent. *Clique* code was debugged solely on the development system itself without running on the embedded device, still it proved to work on the embedded device as well. The Visual Studio debugger used in the process is far more helpful and intuitive than any hardware debugger or debugging server available for an embedded Linux.

Project	Lines of Code
Storage Node	387
Social client	277
Communicator	367

Table 3: Code metrics of all Clique components

5.2.3 Platform independence

Platform independence gives developers the freedom to deploy a software product on a multitude of platforms and operation systems after having it developed only once without the effort of porting it to another platform. This aspect was already detailed in the Section 2.4.2 but with developing for an embedded system this advantage proved to be even more beneficial.

In the process of beta testing the *Clique* application, it was impossible and unreasonable to provide every user with a DSL-Router for testing. Although it is favorable to test an application on the actual platform, insisting on it, would have created too many problems. The process of deploying software on a router is just too complicated and time consuming for most people. The platform independence of Mono allowed *Clique* to be run on basically any system. The deployment process could be reduced to simply sending an E-Mail with the attached program and testers were ready to go.

5.3 Performance

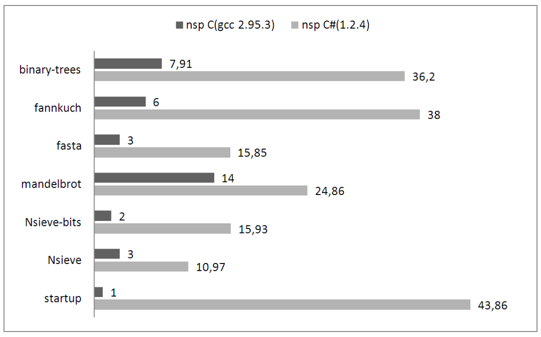
5.3.1 Scalability

Scalability is one of the main advantages of a P2P system such as *Clique* with each new user increases the system's capacity as each user introduces its share of resources. While most P2P systems distribute the network load uniformly onto all nodes the social network overlay used by *Clique* regulates itself. While popular profiles will have lots of access requests, they also have more friends to distribute replicas to. This will decrease the load to the node automatically. For each node the amount of friends and replicas will automatically scale to the needs of the local friend cluster.

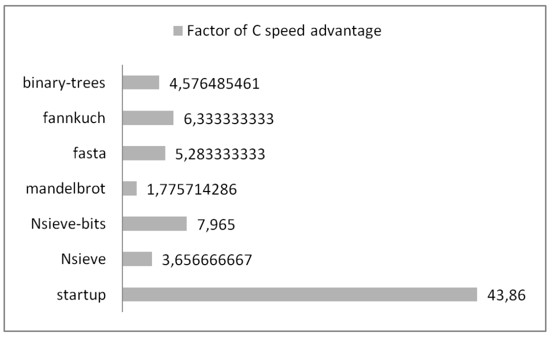
As *Clique* uses a hybrid approach, the central lookup server is more vulnerable to scalability issues. Although this is true the lookup server is to be expected to have far less network load than an ordinary web server providing a complete social network service. IP addresses are likely change only every 24h per router. This makes it possible for nodes to cache connection data for a long period of time and avoid unnecessary lookups.

5.3.2 C vs. Mono

In order to assess the performance deficiencies of Mono compared to a native C implementation several benchmarks from the Computer Language Benchmarks Game Project [13] were performed. These benchmarks were used to evaluate the actual performance difference between both approaches. In figure 20a the exact results of each benchmark are shown. The results show the measured execution time in seconds lower values mean faster execution. To provide a better comparison figure 20b presents the same benchmark results as a factor of how many times C was faster than the C# version. As can be seen in figure 20b every benchmark performed by Mono takes at least 1.8 times longer to execute. In the worst case of Nsieve-bits Mono is even 8 times slower. The startup time can be ignored as the imagined services on a router would only be started once and kept in memory. The performance advantage of native C code was never argued and these benchmark show that Mono, although slower, is capable of performing similar to C code.



(a) Benchmarks



(b) Factor representation

Figure 20: Performance comparassion C vs. Mono

5.3.3 Replication

To assess the performance of the replication process, three test runs were performed ranging from 1000 to 3000 wall entries to be stored on a remote storage node. The sink part of this scenario was performed by a Linux server connected to the internet with direct 100Mbit/s Ethernet connection while client performing the replication was connected over a 2Mbit/s consumer DSL-Line. Both parties were running an unaltered version of the storage node software. Table 4 presents the gained results. It is shown that this setup of storage nodes can perform an average of 20 remote store operations per second. The amount of text included in each entry was uniformly distributed between 3 to 100 words per entry which is more than most ordinary wall entries store.

Performance could be vastly improved by combining wall entries into bulk sets of larger packets. Right now each entry is send separately which creates a large amount of unnecessary overhead slowing the replication. Bulk sets could reduce this overhead considerable. As an example the 3000 wall entries amounted to a file size of 1.2MB but created network traffic of over 8MB combining the actual data sent and acknowledgement packets received for successful save operations.

# of messages	time in s	avg remote store operations per s
1000	42	23,8
2000	104	19,2
3000	176	17,0

Table 4: Code measurements

CHAPTER 6

Conclusion

6.1 Summary

The presented work aimed to develop a distributed system for storing and distributing social data, focusing on security aspects in the handling of this data. In summary, a prototype system was designed, implemented and testes. I was able to confirm the feasibility of developing and deploying a trust based P2P social network service on top of an embedded platform with the help of the Mono runtime. The presented *Clique* solution allows users to store, retrieve and present social data based on P2P technologies. Data distribution is secured by restricting access to trusted users and avoiding data distribution to third parties. This shows that *Clique* is an adequate solution to replace an existing WBSN. The essential feature set of WBSNs can thus be kept, added by important data privacy guarantees.

Strengths of this diploma thesis are the incorporating of trust to improve security as the deployment of a distributed software system in an embedded system using the Mono runtime. Furthermore, Mono proved to be a well suited runtime for platform independent development, even in a restricted environment such as DSL-Routers. The achieved performance was sufficient and the developmental process benefited to a great extent from the given platform independence and development tools.

Besides the achievements of this work, some limitations and issues have to be considered. Decoupling the social client from the storage node turned out to pose some problems when fetching the personal profile of a user. The social client lacked knowledge about its designated user name therefore could not contact his local storage node in case a different configuration was used. A limitation with regard to Mono was the startup delay of Mono applications. Frequently starting applications, e.g. the presented socio client, were affected. This may be solved by using different socio client designs that keep the client persistent in memory and avoid a new start. Another issue of Mono was the defective implementation of two methods originally used by *Clique*. The resulting error search was complicated by the fact that the first method was not defective to the extent of raising an exception but required an execution time of more than a minute instead of seconds. This way the application just stalled instead of breaking.

Personally, I found it most difficult to structure my work and find a reasonable balance between actual implementation work and the writing process. Planning work packages and trying to estimate the time I would need, I constantly underestimated time and effort.

6.2 Future research

Additional questions arose during the process of completing this work. The most promising questions and potential research paths are presented here.

- *Clique* provides a stable backbone infrastructure for P2P social networking but is very strict in regulating access to the network. It is an interesting challenge to extend the current *Clique* design to a system that allows ubiquitous access from external networks and different devices. Of special interest are the possible means of authentication between external clients and a *Clique* node.

- Having fuzzy search capabilities with range query support is one of the main features of WBSNs. But even today range queries pose a complex problem to P2P systems. Although systems exists that claim to support range queries [15, 20], it has yet to be examined how well these perform are compared to traditional database queries and if it would be feasible to integrate them into *Clique*.

- WBSN users have already accumulated a certain amount of social data that could be used as a repository for filling the *Clique* network with data. In order to do this, possible ways of data migrating must be explored. Possible solutions would be browser plug-ins or an synchronization application integrated into *Clique*. This application could automatically retrieve and sync existing social network accounts with the *Clique* infrastructure.

Bibliography

[1] http://en.wikipedia.org/wiki/Replication_(computer_science) (Zitiert auf Seite 16)

[2] http://www.microsoft.com/NET/ (Zitiert auf Seite 22)

[3] http://java.sun.com/ (Zitiert auf Seite 22)

[4] http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx (Zitiert auf Seite 23)

[5] http://www.mono-project.com/ (Zitiert auf Seite 23)

[6] http://en.wikipedia.org/wiki/.NET_Languages (Zitiert auf Seite 23)

[7] http://www.dotnetpowered.com/languages.aspx (Zitiert auf Seite 23)

[8] http://msdn.microsoft.com/en-us/vstudio/products/default.aspx (Zitiert auf Seite 23)

[9] http://trust.mindswap.org/cgi-bin/relationshipTable.cgi (Zitiert auf Seite 24)

[10] http://www.datacenterknowledge.com/archives/2008/12/17/facebook-adding-600000-users-a-day/ (Zitiert auf Seite 25)

[11] http://www.facebook.com/note.php?note_id=39391378919&id=9445547199&index=0 (Zitiert auf Seite 25)

[12] http://www.opendht.org/ (Zitiert auf Seite 25)

[13] http://shootout.alioth.debian.org/ (Zitiert auf Seite 52)

[14] *Secure Hash Standard.* National Institute of Standards and Technology. – Federal Information Processing Standard 180-2. http://csrc.nist.gov/publications/fips/ (Zitiert auf Seite 12)

[15] Aberer, Karl ; Cudré-Mauroux, Philippe ; Datta, Anwitaman ; Despotovic, Zoran ; Hauswirth, Manfred ; Punceva, Magdalena ; Schmidt, Roman: P-Grid: a self-organizing structured P2P system. In: *SIGMOD Rec.* 32 (2003), Nr. 3, S. 29–33. http://dx.doi.org/http://doi.acm.org/10.1145/945721.945729. – DOI http://doi.acm.org/10.1145/945721.945729. – ISSN 0163–5808 (Zitiert auf Seite 55)

[16] Aberer, Karl ; Punceva, Magdalena ; Hauswirth, Manfred ; Schmidt, Roman: Improving Data Access in P2P Systems. In: *IEEE Internet Computing* 6 (2002), Nr. 1, S. 58–67. http://dx.doi.org/http://dx.doi.org/10.1109/4236.978370. – DOI http://dx.doi.org/10.1109/4236.978370. – ISSN 1089–7801 (Zitiert auf Seite 12)

[17] Barnes, JA: Class and committees in a Norwegian island parish. In: *Human Relations* 7 (1954), S. 39–58 (Zitiert auf Seite 24)

[18] Barr, Michael ; Massa, Anthony: *Programming Embedded Systems: With C and GNU Development Tools.* O'Reilly Media, Inc., 2006. – ISBN 0596009836 (Zitiert auf Seite 20)

[19] Becker, Steffen ; Hasselbring, Wilhelm ; Paul, Alexandra ; Boskovic, Marko ; Koziolek, Heiko ; Ploski, Jan ; Dhama, Abhishek ; Lipskoch, Henrik ; Rohr, Matthias ; Winteler, Daniel ; Giesecke, Simon ; Meyer, Roland ; Swaminathan, Mani ; Happe, Jens ; Muhle, Margarete ; Warns, Timo: *Trustworthy software systems: a discussion of basic concepts and terminology.* 2006. – 1–18 S (Zitiert auf Seite 14)

[20] Bharambe, Ashwin R. ; Agrawal, Mukesh ; Seshan, Srinivasan: Mercury: supporting scalable multi-attribute range queries. In: *SIGCOMM Comput. Commun. Rev.* 34 (2004), Nr. 4, S. 353–366. http://dx.doi.org/http://doi.acm.org/10.

1145/1030194.1015507. – DOI http://doi.acm.org/10.1145/1030194.1015507. – ISSN 0146–4833 (Zitiert auf Seite 55)

[21] Buchegger, Sonja ; Datta, Anwitaman: A Case for P2P Infrastructure for Social Networks - Opportunities & Challenges. In: *The Sixth International Conference on Wireless On-demand Network Systems and Services* (2009) (Zitiert auf Seiten 3, 4, 5 und 25)

[22] Busca, Jean michel ; Picconi, Fabio ; Sens, Pierre: Pastis: A highly-scalable multi-user peer-to-peer file system. In: *in Euro-Par 2005*, 2005 (Zitiert auf Seite 13)

[23] Clip2: *The Gnutella Protocol Specification v0.41*. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf (Zitiert auf Seite 12)

[24] Cohen, Bram: *Incentives Build Robustness in BitTorrent*. 2003 (Zitiert auf Seite 11)

[25] Coulouris ; Dollimore, Jean ; Kindberg, Tim: *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2005. – ISBN 0321263545 (Zitiert auf Seiten 9 und 16)

[26] Dabek, Frank ; Kaashoek, M. F. ; Karger, David ; Morris, Robert ; Stoica, Ion: Wide-area cooperative storage with CFS. In: *SOSP '01: Proceedings of the eighteenth*
ACM, 2001. – ISBN 1–58113–389–8, S. 202–215 (Zitiert auf Seite 13)

[27] Ford, Bryan ; Strauss, Jacob ; Lesniewski-Laas, Chris ; Rhea, Sean ; Kaashoek, Frans ; Morris, Robert: Persistent personal names for globally connected mobile devices. (2006), S. 233–248. ISBN 1–931971–47–1 (Zitiert auf Seite 25)

[28] Golbeck, Jennifer A.: *Computing and applying trust in web-based social networks*. College Park, MD, USA, Diplomarbeit, 2005. – Chair-James Hendler (Zitiert auf Seiten 15 und 24)

[29] Gray, Jim ; Helland, Pat ; Shasha, Dennis: The dangers of replication and a solution. (1996), S. 173–182 (Zitiert auf Seite 16)

[30] Grimstad, Stein ; Sjøberg, Dag I. K. ; Atkinson, Malcolm P. ; Welland, Ray: Evaluating Usability Aspects of PJama Based on Source Code Measurements. In: *Proceedings of the 8th International Workshop on Persistent Object Systems (POS8) and Proceedings of the 3rd International Workshop on Persistence and Java (PJW3)*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1–55860–585–1, S. 307–321 (Zitiert auf Seite 50)

[31] Heckmann, Oliver ; Bock, Axel: The eDonkey 2000 Protocol / Multimedia Communications Lab, Darmstadt University of Technology. Version: Dec 2002. `ftp://ftp.kom.e-technik.tu-darmstadt.de/pub/papers/HB02-1-paper.pdf` (KOM-TR-08-2002). – Forschungsbericht. – Elektronische Ressource (Zitiert auf Seite 11)

[32] Informations-Technologie, Fraunhofer Institut S. ; Informations-Technologie, Fraunhofer Institut S. (Hrsg.): PrivatsphÃ¤renschutz in Soziale-Netzwerke-Plattformen. Version: 2008. `http://www.sit.fraunhofer.de/Images/SocNetStudie_Deu_Final_tcm105-132111.pdf`. – Forschungsbericht. – Elektronische Ressource (Zitiert auf Seite 6)

[33] Kalofonos, Dimitris N. ; Antoniou, Zoe: A Hybrid P2P/Infrastructure Platform for Personal and Social Internet Services. In: *Proceedings of the 19th IEEE Personal, Indoor, and Mobile Radio Communications Symposium - Applications, Services & Business Track (PIMRC '08)* (2008) (Zitiert auf Seite 25)

[34] Kalofonos, Dimitris N. ; Antoniou, Zoe ; Reynolds, Franklin D. ; Van-Kleek, Max ; Strauss, Jacob ; Wisner, Paul: MyNet: A Platform for Secure P2P Personal and Social Networking Services. (2008), S. 135–146. `http://dx.doi.org/http://dx.doi.org/10.1109/PERCOM.2008.40`. – DOI http://dx.doi.org/10.1109/PERCOM.2008.40. ISBN 978–0–7695–3113–7 (Zitiert auf Seite 25)

[35] Kamvar, Sepandar D. ; Schlosser, Mario T. ; Garcia-Molina, Hector: The Eigentrust algorithm for reputation management in P2P networks. In: *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–680–3, S. 640–651 (Zitiert auf Seite 14)

[36] Kubiatowicz, John ; Bindel, David ; Chen, Yan ; Czerwinski, Steven ; Eaton, Patrick ; Geels, Dennis ; Gummadi, Ramakrishna ; Rhea, Sean ; Weatherspoon, Hakim ; Wells, Chris ; Zhao, Ben: OceanStore: an architecture for global-scale persistent storage. In: *SIGARCH Comput. Archit. News* 28 (2000), Nr. 5, S. 190–201. http://dx.doi.org/http://doi.acm.org/10.1145/378995.379239. – DOI http://doi.acm.org/10.1145/378995.379239. – ISSN 0163–5964 (Zitiert auf Seite 13)

[37] Muthitacharoen, Athicha ; Morris, Robert ; Gil, Thomer M. ; Chen, Benjie: Ivy: a read/write peer-to-peer file system. In: *SIGOPS Oper. Syst. Rev.* 36 (2002), Nr. SI, 31–44. http://dx.doi.org/http://dx.doi.org/10.1145/844128.844132. – DOI http://dx.doi.org/10.1145/844128.844132. – ISSN 0163–5980 (Zitiert auf Seite 13)

[38] Noergaard, Tammy: *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005. – ISBN 0750677929 (Zitiert auf Seite 20)

[39] Page, Lawrence ; Brin, Sergey ; Motwani, Rajeev ; Winograd, Terry: The PageRank Citation Ranking: Bringing Order to the Web / Stanford Digital Library Technologies Project. 1998. – Forschungsbericht (Zitiert auf Seite 14)

[40] Parameswaran, Manoj ; Susarla, Anjana ; Whinston, Andrew B.: P2P Networking: An Information-Sharing Alternative. In: *Computer* 34 (2001), July, Nr. 7, 31–38. http://dx.doi.org/http://dx.doi.org/10.1109/2.933501. – DOI http://dx.doi.org/10.1109/2.933501. – ISSN 0018–9162 (Zitiert auf Seite 11)

[41] Patterson, David A. ; Gibson, Garth ; Katz, Randy H.: A case for redundant arrays of inexpensive disks (RAID). (1988), S. 109–116. http://dx.doi.org/http://doi.acm.org/10.1145/50202.50214. – DOI http://doi.acm.org/10.1145/50202.50214. ISBN 0–89791–268–3 (Zitiert auf Seite 19)

[42] Peng, Gang: CDN: Content Distribution Network. (2003) (Zitiert auf Seite 20)

[43] Plank, J. S.: *Erasure Codes for Storage Applications*. San Francisco, CA : Tutorial Slides, presented at *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html, 2005 (Zitiert auf Seite 18)

[44] Rowstron, Antony ; Druschel, Peter: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001, S. 329–350 (Zitiert auf Seiten 12 und 13)

[45] Schiüberg, Doris: *A Peer-to-peer Infrastructure for Social Networks*, Technische UniversitÃ¤t Berlin, Diplomarbeit, 2008 (Zitiert auf Seite 25)

[46] Soanes, Catherine (Hrsg.) ; Stevenson, Angus (Hrsg.): *"trust noun" The Oxford Dictionary of English (revised edition).* Oxford University Press http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t140.e82219 (Zitiert auf Seite 14)

[47] Stoica, Ion ; Morris, Robert ; Karger, David ; Kaashoek, M. F. ; Balakrishnan, Hari: Chord: A scalable peer-to-peer lookup service for internet applications. In: *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA : ACM, 2001. – ISBN 1–58113–411–8, S. 149–160 (Zitiert auf Seite 12)

[48] Sunhil H. Madhani, Alberto G.: *METHODS AND DEVICES FOR STANDALONE SOCIAL NETWORKING AND INTERNET.* 2008 (Zitiert auf Seite 15)

[49] Tanenbaum, Andrew: *Computer Networks.* Prentice Hall Professional Technical Reference, 2002. – ISBN 0130661023 (Zitiert auf Seite 10)

[50] Tanenbaum, Andrew S. ; Steen, Maarten V.: *Distributed Systems: Principles and Paradigms.* Upper Saddle River, NJ, USA : Prentice Hall PTR, 2002. – ISBN 0130888931 (Zitiert auf Seite 9)

[51] Weatherspoon, Hakim ; Kubiatowicz, John: *Erasure Coding Vs. Replication: A Quantitative Comparison.* http://dx.doi.org/10.1007/3-540-45748-8_31. Version: 2002 (Zitiert auf Seite 18)

[52] Zhao, Ben Y. ; Kubiatowicz, John D. ; Joseph, Anthony D.: Tapestry: a fault-tolerant wide-area application infrastructure. In: *SIGCOMM Comput. Commun. Rev.*

32 (2002), Nr. 1, S. 81–81. http://dx.doi.org/http://doi.acm.org/10.1145/510726.510755. – DOI http://doi.acm.org/10.1145/510726.510755. – ISSN 0146–4833 (Zitiert auf Seite 12)