1 Introduction

In the last two decades the Internet has developed not only into an indispensable tool and inexhaustible source of information, but has also given rise to a wealth of application services like, e.g, eMail, file-sharing, interactive geographic maps, persistent data storage, Web content search, and e-commerce services like online shopping, price search and comparison, e-auctions, and reputation management to give only a few examples. Of course with the growth of the Internet in terms of hosts and users, its applications are forced to cope with more and more clients in general.

In the pre-Peer-to-Peer (P2P) networks era, Internet services have been offered by servers only. Clients send network messages to servers to ask the latter to deliver a resource or perform a desired computation, for instance. Servers then respond to these requests with a copy of the desired resource or the result of the computation. This type of communication is typical of client-server systems ([Coul 05], p. 35). To offer a service, a provider must allocate and avail physical resources such as storage, memory, CPU cycles, and bandwidth in order to satisfy incoming client requests. This model has worked well in the past and continues to work well today.

However, if applications are of global scale and the user population is large (>1 million) and diverse in terms of access time, access pattern, hardware, and connectivity, then the client-server model reaches its limits. For example, every single server can serve up to its processing power and the bandwidth of its network connection. If the number of requests exceeds this upper bound, then it becomes overloaded and thus stops working in an efficient way. Moreover, every server is a single point of failure.

This situation can be alleviated by introducing variations of the classical one-server-multiple-clients model. For instance, multiple servers and caches can be used to achieve load-balance, better fault-tolerance, and performance to name a few. However, service providers are in charge of adding extra resources when the available ones have become unable to cope with the load. Moreover, they have to bear the expenses of installing, setting up, and maintaining additional resources. If clients do not pay for services they take advantage of, providers will be reluctant to spend more money on better service delivery in most cases. To summarize, the client-server model's key limitations are scalability, costs of operation and maintenance, and its unilateral way of granting or denying access to resources.

Peer-to-Peer technology [Oram 01] promises to be the next evolutionary step to distributed, robust, fault-tolerant, efficient, and democratic collaboration/cooperation among connected agents. The technology takes advantage of voluntarily offered resources such as stable storage, CPU cycles, bandwidth, and even human intelligence available at every peers' site. Peers have a significant or total autonomy from one another and share and combine their decentralized resources to build and make available services that can be accessed by every other peer. Thus they also share the costs of maintaining their aggregation and keeping application services on top of it operational. Such an aggregation of peers, a potentially large fraction of which having temporary network addresses and variable connectivity, that devote their resources to a public good is termed *Peer-to-Peer network*. However, a Peer-to-Peer network is notoriously unstable and in flux, which is caused by peers joining and leaving the Peer-to-Peer network at their own discretion. But users favor reliable services built atop a Peer-to-Peer network. So there must be protocols and algorithms in place that guarantee best possible service reliability in spite of peer population dynamics.

The traditional, research-driving application domain of Peer-to-Peer networking in the last 6 years was file-sharing. Peers when joining a Peer-to-Peer file-sharing network bring along their files for others to download. All peers can search independently for files and start downloading them once one or more peers storing the desired files have been found. The identification of a file is either by its associated name alone or by its name in combination with a small digest (a.k.a. hash) of parts of its contents.

The earliest and best-known Peer-to-Peer network that supports file-sharing is Napster [Brai 02]. This Peer-to-Peer network has a centralized peer solely responsible for answering search requests from all peers. Clearly, this approach suffers from not being scalable. Furthermore, the file-sharing application domain stipulates that resources such as audio and video files be immutable. Once an audio file is manipulated by accident or deliberately, it becomes a new file that is not identical to other files with the same name, for example. In effect, this means Peer-to-Peer file-sharing networks do not fit to applications that require updatable resources. On top of that, an inherent feature of these networks is that all files stored on a particular peer also vanish when it leaves. If no other peer still online downloaded these files, then all of them would become unavailable. When peers join or leave at a high rate, chances are that downloads once started never finish completely thus leaving unusable fragments of files on peers. An application that demands that resources, once inserted, be available with high probability would clearly not rely on services offered by Peer-to-Peer file-sharing networks.

Nowadays research interest focuses on other domains besides file-sharing. Among these we find groupware and personal information management, storage of reputation and trust information [Fahr 02], support of digital libraries [Riss 05], distributed variants of DNS [Cox 02], and multimedia data streaming [Kubi 00]. These domains demand that resources be distributed across peers and once inserted be available. Furthermore, access to resources must be efficient and scalable. To better understand the detailed requirements of these applications, we describe an application scenario from the domain reputation management. This domain serves to motivate research questions we tackle with our Peer-to-Peer network.

1.1 Application Domain: Reputation Management

In this section we want to give a brief introduction into reputation management and clarify what is meant with reputation, how it is handled and who benefits. First of all, we motivate why reputation is important in virtual (a.k.a. online) communities and then give an example from the real world to illustrate the concept of reputation.

Trusting the other party to behave as expected is more often than not an issue in the real world. If somebody trusts unequivocally, the other party might use this information to act against her. On the other hand, being distrusting all the time will preclude every interaction with other parties. At any rate, a sound balance is needed. This trust dilemma is even more acute in the aforementioned communities. In these, participants are lacking prospect-related information from their five senses because usually they do not have any direct face-to-face or haptic contact. Theoretical work [Raub 90] has shown that virtual communities benefit greatly from sharing and aggregating information about members past interactions. The exchange of this kind of reputational information helps a community develop better transparency, trust and cooperation. R. Lethin [Leth 01] defines reputation concisely as "...the memory and summary of behavior from past transactions". However, we must note that the application of this definition depends on the supposition that past behavior is indicative of future behavior.

In the e-Commerce domain, eBay [eBay 95] is one of the oldest and most prominent e-auction sites on the Web. Participants can rate the other's behavior after their trade has finished. Any rating comprises of a general assessment of the trade (positive, neutral, or negative) and a short comment. eBay also calls ratings *feedbacks*. It manages members' feedbacks in a centralized fashion with a system called "Feedback Forum". The running total of feedback points, i.e., the total number of positive, neutral, and negative assessments, is attached visibly to each participant's screen name, so everybody is able to readily check the transaction partner's accumulated *reputation*.

However efficient and accepted eBay's Feedback Forum is, it lacks the important feature of taking one's reputation elsewhere on the Web. For example, it would be beneficial for participants to bring their hard-earned reputation profile into action on a different platform. But this is not possible, for eBay lays claim to this data. Although it is possible to refer others to one's own reputation profile on eBay, one cannot add additional feedback coming from outside of eBay. Furthermore, you never know what will happen to your reputation data when you are offline or eBay's servers suffer a severe outage and large amounts of this kind of data are lost.

We feel, every participant who takes advantage of a reputation system such as eBay's Feedback Forum should be entitled to show, give and receive feedback whenever and to/from whomever she wants. Reputation built up by accumulating this feedback should then be the property of the one who was rated and it should be accessible at all times regardless of whether servers of a reputation service provider have failed or not. Feedback exchanged between participants is sensitive data, so it needs to be guarded against eavesdropping and tampering. Since feedback should not be stored centrally, we envision a distributed solution and think today's Peer-to-Peer technology seems to be well suited to provide an infrastructure reputation management can be built upon.

The basic idea behind reputation management is that past feedback histories guide future trust decisions participants make at the beginning of a transaction. These histories should encourage trustworthy behavior, and deter participation by those who are dishonest or unqualified. Moreover, participants need not only be human beings. It is equally possible that participants are autonomous software agents [Pado 01] rating one another. Such agents can act selfish or unfair thus leaving room for their behavior being rated either good or bad.

We envision the following e-Commerce scenario, where reputation management is an application atop a special Peer-to-Peer network that, unlike traditional Peer-to-Peer file-sharing networks, persists data items reliably. There are participants who want to trade with one another. They are registered with and currently authenticated for the reputation management application. After having looked up some potential trade partners, a participant wants to assess their trustworthiness. So she searches for her trade partners' reputations by looking them up in the Peer-to-Peer network. Feedback histories are stored securely and tamper-proof on peers in the Peer-to-Peer network. Soon afterwards she receives the desired histories and is able either on her own or with the aid of an evaluation software to assess the trustworthiness of the potential trade partners. Among the latter, she finds one who seems to have the best trustworthiness that matches her trust expectations. Subsequently, she advises the selected partner that she wants to trade with him. Now the partner checks her reputation. If he finds her trustworthiness to be sufficient, too, both start to trade. At the end of their interaction, both have made their experiences and compose a feedback about the other's behavior. This feedback is then stored in the Peer-to-Peer network by participants either calling the `insert` or `update` operation which creates or updates the other's reputation profile, respectively. Since feedback data will be of small size, lengthy data transfers to peers are avoided thus responsiveness of the application is maintained.

Of course, this scenario lacks many details. For example, she may not consent to the feedback given by her trade partner. So there must be some sort of mediation to remedy this situation. By describing an example scenario, our intent was to illustrate the way reputation management works. More details about our reputation management application and how it guarantees participant security can be found in [Fahr 02].

1.2 Contributions of the Thesis

The main contribution of this thesis is a solution to the issue of storing data items persistently and reliably in a Peer-to-Peer data store that is a special form of distributed system in which hosts (a.k.a. peers) come and go as they please and that has no centralized control. Our Peer-to-Peer data store called "HyCub" is able to guarantee data availability with high probability[1] even under increased peer population fluctuation.

In particular, contributions of this thesis to the field of Peer-to-Peer networking are to the best of our knowledge and can be summarized as follows:

- We are the first to provide a clear definition of the term "Peer-to-Peer data store".
- Our proposed Peer-to-Peer data store called "HyCub" is a hypercube-based overlay network that employs the concept of "replication groups". Peers in such groups ensure high availability of data items. This makes "HyCub" much more resilient against peer population fluctuation than Peer-to-Peer networks proposed in the last years.
- We devised a special intra-replication group communication scheme based on epidemic-style content dissemination that is robust against fluctuation and achieves a very high degree of dissemination, i.e., all peers in a group get new messages with high probability.
- The design of "HyCub"'s join process presents attackers less target for Denial-of-Service attacks.
- We propose novel maintenance operations in "HyCub" that ensure, on the one hand, optimal performance of the Peer-to-Peer data store in case the peer population grows and, on the other hand, data items remain available when the peer population shrinks. We analyzed both operations theoretically and validated our results with simulations.

With our contributions we attempt to give answers to the following questions:

[1] When we use the term "with high probability", we essentially mean with probability at least $1 - N^{-c}$ for a $c \geq 1$ and a sufficient large N.

1. To what extent is our proposed Peer-to-Peer system capable of persisting resources in a reliable way so that requirements of the example application domains are met?

2. How well does self-organization in our proposed Peer-to-Peer system maintain resource availability and consistency despite peer population dynamics?

3. How does our proposed Peer-to-Peer system compare to other Peer-to-Peer networks in terms of performance and fluctuation resilience?

1.3 Thesis Organization

The organization of the remainder of this thesis is as follows: Chapter 2 introduces the basic definitions found in the Peer-to-Peer networking domain and lists requirements every Peer-to-Peer network must meet (Section 2.1.2). We give a classification of modern Peer-to-Peer networks in Section 2.1.1.

Since we concern ourselves with peer population fluctuation, we need to give a definition of the notion "fluctuation" in the context of Peer-to-Peer networks (Section 2.1.3). Looking up resources works differently in Peer-to-Peer networks than in traditional client-server systems. Section 2.2 describes both approaches and illustrates the differences.

When a Peer-to-Peer network inventor proposes her novel Peer-to-Peer network, she presents it in her own way. However, presentations from different inventors are not compatible with one another, which makes comparing their performance much more difficult. We present a conceptual model with which Peer-to-Peer networks can be described in a way that makes them comparable (Section 2.3). To show that this in fact is feasible, we present well-known Peer-to-Peer networks like Chord, Kademlia, Kelips, etc. invented during the last 6 years.

The next section 2.5 deals with the important problem of data availability in Peer-to-Peer networks, which is much more severe than in client-server systems. We list and describe approaches to ensure data availability. The examples are from the distributed database systems and Peer-to-Peer domain. Whenever replication is used to make sure resources remain available, data consistency issues arise.

Section 2.6, first, gives a definition of data consistency. Then it presents various levels of data consistency and how they differ from one another. Furthermore, it shows how different systems also from outside the Peer-to-Peer domain implement those consistency levels.

Chapter 3 is one of the central chapters of this thesis. It is about our Peer-to-Peer data store called “HyCub” which excels Peer-to-Peer networks in data availability in general. Since the concept of a Peer-to-Peer data store is new, we, first, describe necessary environmental conditions for its existence and then give a definition of what it is about.

Section 3.2 starts with a list of additional requirements for Peer-to-Peer data stores and uses the conceptual model presented in Section 2.3 to describe “HyCub” and its features. Among the latter are “HyCub”’s topology, how data availability and data consistency are guaranteed and ensured, how peer failures are detected, and how the Peer-to-Peer data store is maintained. We also describe basic operations such as lookup, insert, update, and delete applications atop of “HyCub” can call to find and manipulate resources.

Designing Peer-to-Peer networks in general is a complex task. So every new proposal needs to be tested before it can be deployed in the real world. To this end, simulation proved to be a valuable tool to test a new design and evaluate its characteristics and performance. Chapter 4 introduces into simulation of distributed systems in general and lists requirements and necessary assumptions for simulating our Peer-to-Peer data store. We studied several integrated simulation environments and list the results of this evaluation in Section 4.4. Unfortunately, none of the simulation environments was able to fulfill all of our requirements so we turned to designing and implementing our own simulator, which we describe in Section 4.5. We are also able to generate visualizations of simulation runs. How this is accomplished and by which means is the subject of Section 4.5.3.

Chapter 5 is our second central chapter of this thesis. In this chapter we describe and expound the simulation experiments and results we did with “HyCub” and the other Peer-to-Peer networks.

First, we provide a methodology that enables us to arrive at our results. Then we analyze “HyCub”’s lookup algorithm theoretically and verify our theoretical results by simulation experiments (Section 5.2).

The next Section 5.3 details both theoretical and experimental results of the study of the message forwarding algorithm. Message forwarding is necessary for the lookup algorithm to work at all. We found out our original lookup algorithm leaves room for improvement. So we amended it and present details of this amendment and its impact on lookup performance in Section 5.4.

Since we are especially interested in how “HyCub” is able to cope with fluctuation, we devote the next two sections 5.5 and 5.6 to studying “HyCub”’s maintenance operations that counter the effects of fluctuation.

Finally, in Chapter 6 we summarize our contributions, discuss our achievements, and conclude with recommendations for future research in the field Peer-to-Peer data stores.

Appendix A lists publication that were written in the context of this dissertation and Appendix B provides a list of the symbols we use in this thesis.