Project Group - A Distributed Framework for Social Networks
Overview of the Seminar Topics

Dr.-Ing. Kalman Graffi
Universität Paderborn
30. March 2011
Email: graffi@mail.upb.de

Abstract

This paper presents the seminar topic for the project group (PG) "'A Distributed Framework for Social Networks"'. There are 12 topics, that will be assigned to the participants of the project group. In the following, you find a short motivation of each topic and a list of research questions that need to be addressed in your paper. Please keep in mind, that the PG aims at developing a p2p-based framework for social networks. Several questions how this goal is aimed at best are open and need to be clarified through the commitment of your seminar paper. As an input for your seminar, a few papers are listed. They mark the initial stepping stone for further investigations. All papers should be downloadable from the university network.

Organization

The seminar addresses essential questions for the creation of a distributed framework for social networks. Each seminar addresses questions in a given research field. The project task in the seminar is to elaborate on the question and to propose a sketch how the question could be solved in the PG. Typically the core question asks how a specific functionality has been solved in prominent papers. A subsequent question is, whether the approach is applicable in our scenario and if not, which modifications exist and are advisable. In order to solve these questions, an initial set of papers per seminar topic is handed out, additional suitable papers should be found by you.

Requirements on the Paper

The seminar paper should comply with following requirements:

- The paper should be written in Latex, using the template given on the website of the PG [1].
- 10+ pages, documentclass[10pt,letterpaper,onecolumn]{article} .
- The paper should motivate the question, elaborate on related work and existing solutions and sketch the answers/alternatives for the PG.
- It is advisable to use Google Scholar [2] to find further papers.
- Please use the correct and complete bibtex description from DBLP [3] to cite papers.

Seminar Dates

Following dates are planned for the seminar:

- Mon 28. March - Meeting: seminar topic announcement
- Mon 11. April - Meeting: seminar topic assignment

[1] http://www.cs.uni-paderborn.de/fachgebiete/fg-ti/lehre0/ss2011/pg-framework.html
[2] http://scholar.google.de/
[3] http://dblp.mpi-inf.mpg.de/dblp-mirror/index.php

- Sun 1. May - Deadline, 23:59: submission deadline of the seminar paper structure and literature list
- Sun 29. May- Deadline, 23:59: submission deadline of the seminar paper
- Thu 2. June - Deadline, 23:59: submission deadline of the seminar reviews
- Wed 8. June- Deadline, 23:59: submission deadline for the camera-ready seminar papers
- Tue/Wed 14.-15. June- Blockseminar: presentation of the seminar talks and discussion (pending)
- Sun 19. June- Deadline, 23:59: Deadline for the final version of the seminar papers
- Tue/Wed 21.-22. June - Kick-off Workshop at off-site location (pending)

Brief Motivation of the Seminar Topics

A general p2p framework offers a framework of combinable functional components that may be arranged and activated to meet the current needs of the p2p applications it is used for. A modular setup of the platform allows for rapid extension of the functionality of it. The platform consists of abstract layers, built over each other, such as for routing, storage and data access. Each single layer consists of one or more software modules or components. These components provide clear interfaces and may be used by other components. Each layer provides services for the immediate upper layer and uses services provided by the immediate lower layer. Through the layer idea, the functionality of a platform is structured and allows for reusable and replaceable software components. The components, located on the same layer, can freely use services of each other using the defined components interface. The number of components, a single layer consists of, can be dynamically changed. Components with new functionality can be included on each of the existing layers. In order to extend the platform and enable a richer functionality set, new architecture layers can be added on the top of the architecture. The new architecture layers should provide the same properties in terms of extendibility and interface as the existing layers, on top of which they were build on.

In the following, we state the components for a p2p platform for applications in general as depicted in Figure 1, as well as some mechanisms used to address these requirements:

- **Structured p2p overlay**: In order to interconnect all peers and to enable a peerID-based functionality, a p2p overlay is needed. Its main task is to interconnect the peers in a structured p2p overlay and provide reliable, consistent routing.
- **Reliable data storage**: In order to provide a reliable put/get operation for distributed data storage, a storage component is needed which can easily be integrated with the overlay. A solution is needed that implements replication strategies to keep data available in the p2p network in the presence of churn.
- **Security Module**: Though security is an omnipresent challenge, it needs to be addressed in every component in the architecture. However, a dedicated module might provide essential building blocks for this.
- **Storage Dispatcher**: In order to manage both local data storage as well as the access to the distributed data storage, a storage dispatcher is needed. Its main task is to identify where to store data or retrieve data based on the ID of the documents to store or retrieve, either locally or distributedly.
- **Information Cache**: In a distributed data storage environment, the retrieval of documents is delayed once accessing the network. In order to keep documents instantly available and to hide the distributed storage from the applications, an information cache is useful.
- **Message Dispatcher**: In order to enable the addressing of inner components or plugins in the p2p application, a pluginID-based message dispatcher might be useful, which main task is to forward messages to the destination plugins.
- **Quality monitoring component**: In order to support both the needs of the users, as well as the system providers, the system quality needs to be monitored.
- **Application functionality**: The actual functionality of the application should be extendible and thus implemented in a modular, plugin-based manner. In order to address this requirement, all involved modules in the software architecture might be based on OSGi declarative services. With this, applications in form of plugins may be loaded and unloaded during the runtime of the application easily.

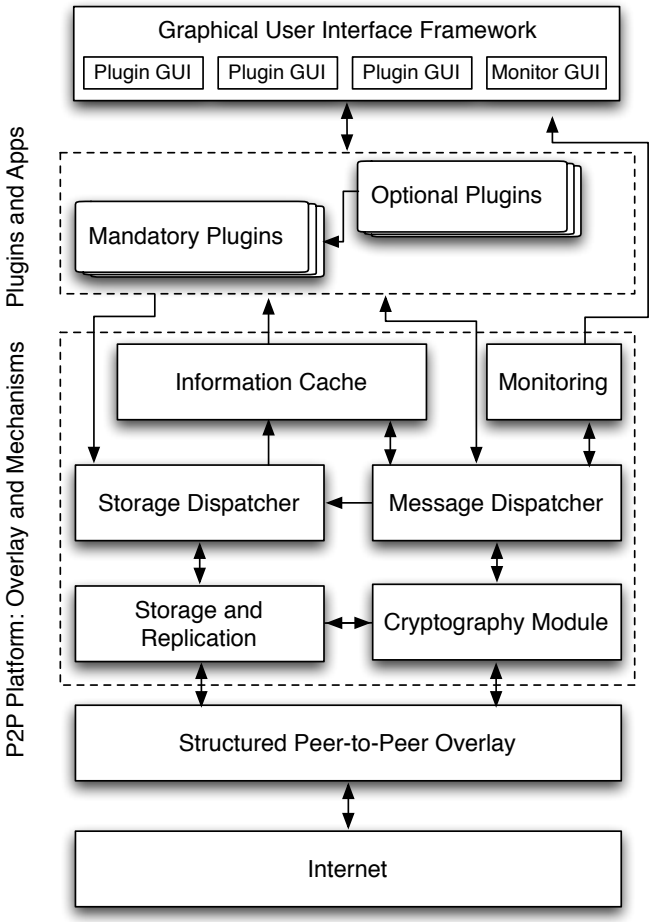


Figure 1. Draft Layout of an Architecture for the Distributed Framework

- **User Level**: The main purpose and motivation of the users need to be known in order to address their needs and to provide a service with satisfying quality.

The though investigation of the research challenges at each individual step in the architecture is addressed by the seminar topics.

1 User Models and Requirements

The main question in the seminar paper addresses the motivation of the users and providers. What kind of functionality and service quality do users expect? What is the exact functionality of Facebook, LinkedIn, Twitter and such online social network sites? What are the non-functional requirements? How fast, "'good"' should the functionality be? How quickly should the searches be resolved, how many results presented, what is the maximum tolerated load on each user device?

From the providers point of view, questions arise with regard to the expected workload. What are the workload statistics of the usage patterns. This comprises the number of users, access patterns, number of friends, activity patterns and as much as possible that helps to model the behaviour of the users. Further, the usage patterns should be analyzed with regard to the user groups. Does the usage vary in dependence of the age, country or social structure of the users?

With regard to the workload, document and infrastructure statistics should be also elaborated. How many documents are stored, of which size and access distribution. If possible, a financial analysis of the expected costs should be sketched. Here, a self operated server, a cloud environment and a p2p infrastructure should be compared.

- **Prio 1:** Danah M. Boyd, Nicole B. Ellison: Social Network Sites: Definition, History, and Scholarship
- **Prio 1:** Fabricio Benevenutoy, Tiago Rodriguesy, Meeyoung Cha, Virgilio Almeida: Characterizing User Behavior in Online Social Networks

- **Prio 1:** Kalman Graffi: Sichere Digitale Soziale Netzwerke Eine Chance fr E-Learning? [4]
- **Prio 2:** Ina O'Murchu, John G. Breslin, Stefan Decker: Online Social and Business Networking Communities
- **Prio 2:** Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, Xiangyang Lan: Group Formation in Large Social Networks: Membership, Growth, and Evolution
- **Prio 2:** Ravi Kumar, Jasmine Novak, Andrew Tomkins: Structure and Evolution of Online Social Networks
- **Prio 2:** Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, Ben Y. Zhao: User Interactions in Social Networks and their Implications

2 PAST - Replication and Data Storage in FreePastry

One important part of a distributed framework for social networks is the distributed storage of the user data. This data has to be stored in the network, distributed and available at all time. This functionality is typically provided by a distributed hash table (DHT) providing a put and get interface. *PAST* is a storage and replication layer on top of the p2p overlay FreePastry.

The seminar paper describes how it works and which benefits and drawbacks it provides with regard to OSNs. One drawback for example of PAST is, that it does not allow the modification of previously entered data, an issue which was addressed later on by the community. These approaches should also be presented and discussed. Other drawbacks of PAST should be described and possible solutions found in literature. As some examples to be named is the missing support for peer heterogeneity, missing deletability and the lack of security.

- **Prio 1:** Peter Druschel, Antony Rowstron: PAST: A large-scale, persistent peer-to-peer storage utility
- **Prio 1:** Antony Rowstron, Peter Druschel: Storage management and caching in PAST, a largescale, persistent peer-to-peer storage utility
- **Prio 1:** Dennis Geels: Data Replication in OceanStore
- **Prio 2:** Laszlo Toka, Matteo Dell Amico, Pietro Michiardi: On Scheduling and Redundancy for P2P Backup
- **Prio 2:** Reza Akbarinia, Esther Pacitti, Patrick Valduriez: Data Currency in Replicated DHTs

3 Distributed Data Structures for Online Social Networks

In order to implement a distributed social network, the user data needs to be stored distributed, too. Thus the user profiles, friend lists, group memberships and photo albums need to be mapped to a distributed data structure. A data structure defines hereby a predefined positioning and linking of the data, the structure allows quick answers on data lookups.

The main question that is answered by this seminar paper is which distributed data structures were proposed for distributed OSNs, or which exist in other application fields, that might be useful in an OSNs. How show typical OSN data structures, for example a friend list, be stored in order to make it easily editable, finable and still replicated in a right amount to fulfil the need for availability and load distribution. The paper also discusses which communication primitives are needed, which challenges to be addressed with regard to security and performance and which ideas are discussed in literature.

- **Prio 1:** Neil Daswani, Hector Garcia-Molina, Beverly Yang: Open Problems in Data-Sharing Peer-to-Peer Systems
- **Prio 1:** Kalman Graffi, Sergey Podrajanski, Patrick Mukherjee, Aleksandra Kovacevic, Ralf Steinmetz: A Distributed Platform for Multimedia Communities

4 Essential Security Aspects for Distributed Online Social Networks

Security is most important aspect in a distributed framework after functionality and performance. This seminar paper addresses this main issue in a broad way. The main question is how to address security issues that arise in a distributed framework for social networks. Typically, the root of trust for a distributed system is a public key infrastructure, which might be identity-based cryptography. The paper answers the question, how a distributed key infrastructure could be implemented, how communication can be secured, which approaches exist for distributed access control and how the trade-off between good privacy and censorship of illegal content could be solved. The paper discusses in broad the security issues that arise in a distributed framework for social networks and the solution offered by literature.

- **Prio 1:** Dan S.Wallach: A Survey of Peer-to-Peer Security Issues
- **Prio 1:** Guido Urdaneta, Guillaume Pierre, Maarten van Steen: A Survey of DHT Security Techniques
- **Prio 1:** Emil Sit, Robert Morris: Security Considerations for Peer-to-Peer Distributed Hash Tables
- **Prio 2:** Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, Ralf Steinmetz: Practical Security in P2P-based Social Networks
- **Prio 2:** Felix Gnther: Safebook Key Management
- **Prio 2:** Dominik Grolimund, Luzius Meisser, Stefan Schmid, Roger Wattenhofer: Cryptree: A Folder Tree Structure for Cryptographic File Systems

[4] http://www.e-learning.tu-darmstadt.de/openlearnware/lehrmaterial_in_olw/fachforumelearning/fachforumelearning2010.de.jsp

5 Services, Apps and OSGi

In order to create an extendible software architecture, which supports the idea of a app-market or plugin-repository, a suitable architecture approach is needed. Services and software components have a long tradition in software engineering, software extensions via p̈luginsäre common to software suites nowadays. Plugins or apps should be regarded as essential element of an architecture, which can be load and started on the fly.

In order to implement such a feature, the *OSGi* Open Services Gateway initiative was created. It allows to structure software in components (bundle, service) and to dynamic load and unload them on runtime. OSGi is for example well integrated in Eclipse.

This seminar paper gives and overview on OSGi, how it can be used and which ideas are presented in literature to create a p2p-based plugin-repository. A plugin-repository acts in the distributed framework for social networks like the android app-market or the apple appstore: new functionality in form of plugins can be downloaded and activated.

- **Prio 1:** Andre L. C. Tavares, Marco Tulio Valente: A Gentle Introduction to OSGi
- **Prio 1:** O. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, T. Watson: The Eclipse 3.0 platform: Adopting OSGi technology
- **Prio 1:** Holger Schmidt, Jon H. Yip, Franz J. Hauck, Rüdiger Kapitza: Decentralised Dynamic Code Management for OSGi
- **Prio 2:** Eunhoe Kim, Kihyun Yun, Jaeyoung Choi: RSP: A Remote OSGi Service Sharing Scheme

User Models and Requirements

Kavitha Jagannath
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: kavitha@mail.uni-paderborn.de

Abstract

This paper concentrates more on what users expect from an online social network (OSN) and what are the requirements that a provider need in order to satisfy them. Online social networks are increasingly attracting the attention of academic and industry researchers intrigued by their affordances and reach. OSN are used by the groups of people with common interests, or like-minds, associate together on social networking sites and build relationships through community depending on the topic, subject matter or atmosphere. Conversations are at the core of social networking and through them relationships are developed. The ability to send and retrieve information over the web using traditional and ubiquitous computing methods has changed the way we work and live. In the past five years, such sites have rocketed from a niche activity into a phenomenon that engages hundreds of millions of internet users. The paper provides with the main types of the social network sites and how they are being classified, and an evaluation will be performed in terms of features and functionality, expected workload with respect to the provider, statistics of the usage patterns and how they vary with different age groups of users.

1 Introduction

In recent years, the Internet and especially the Web has enabled a communication revolution: the ability to send and retrieve information everywhere has changed the way we work and live. Internet based access to information and internet communication means have become ubiquitous. Social networking and blogging account for nearly 10 percent of all time spent on the Internet [2] Since the introduction of Social Networking Sites (SNS) like Facebook, Twitter, Orkut, Yahoo 360, MySpace and many more such networks have attracted millions of people around the globe from a teenager to old person. All these users have made these SNSs as their daily routine in their life because of their wide range of features from sharing a photo to video, sending private messages, taking active participation in the communities that interests them and many more. Interestingly most of the sites not only provide some good features to the existing users but also to a newly added member. Now these networks have moved forward in such a way that they have even replaced e-mails. Social networking and blogging account for nearly 10 percent of all the time spent on the internet [2].

Through OSNs users connect with each other, exchange information with the users who have been added in their network. Different OSNs provide their own unique way for their users to communicate and share information. In a SNS user will store their personal information in their profile or home page. The basic thing that the user does here is join a networking site they like, connect to their friends, family or even strangers and start sharing their information based on each others interest. These networks also helps user to organize events, and then they share this with the people they want to invite to such events. While sharing such personal information the user trusts the other member with whom the content is being shared and most importantly even on the SNS they use. The tendency of people to come together and form groups is inherent in the structure of society and the ways in which such groups take shape and evolve over time is a theme that runs through large parts of social science research [1].

SNSs bring back the departed friends from school or any family member who they have never met may be because of abroad education or some other reason , hence such a networking sites helps people uniting to their old buddies and relatives. Internet plays very increasingly important roles amongst online communities as audiences seek out more specific information. SNSs

gather information from the users during registration period and store these data in user profiles. These user profiles reflect the way the user presents themselves to the other members of the same networking site. These profile information may vary from one networking site to another, their presentation also differs from other that is the look and feel of the personal home page. Some of the information contained in the user profile are whether they are in this network for social activity or professional activity or dating. Users relationship status can also be mentioned explicitly, their area of interest such their favourite books, movies, novels, things users do during leisure, some introduction about themselves. People are curious and voyeuristic; they tend to browse through these photos searching for people they find attractive. Orkut actively encourages this by its hot list or crush list section where members can also send a teaser to the member they find attractive [5]

Social networking sites make it easier to join and connect to new people or communities within a similar geographical area, and to share common interests. Members of sites are eager to sign up and increase their visibility within a network, and to get as many people to join their network making themselves look popular and important. The more connections a person has, the bigger their network is even if the connections are weak ties. Few SNSs presents its members with large networks as connectors, celebrities and stars depending on the number of profile views, average paths and fan counts as in Orkut each member has. However the presence or over exposure on these sites can also at times equate to a popularity contest based on status of how many friends or friends of friends one has. Users also try to find interesting people in these networks and make friendship with them. The activity of the user is of many types as they may be neutral users, where they just create their profile for the sake of their friend and they hardly look into their home profile. These users may be called as passive users. They perform zero activity in the OSNs. The other kind are active users who make SNS as their daily task, these users login many a times in a day and keep adding friends, update their profile, change their profile pictures, add videos, be active in forums or communities to which they have joined and so on. There are also some users who are neither active nor passive; they come between these two types of users. These users use SNS in a very limited way, in the sense this is not a part of their daily routine. Matrimonial networking sites like shaadi.com, findpartner.com are also available where users can find their life partners of the same interest, caste, religion, job and so on. These sites help people to get along with each other. Hence OSN plays a very important role in today's world where people start their day by logging into SNS.

In the next few section we will see how these online social networks motivate the users to use them as a part of their life and the functionalities provided by the network providers to attract these users more and more towards their SNS as there many competitors all around the globe. In section 2 we present the idea of users point of view and in section 3 about the providers point of view

2 Users Point of View

In this section we will see what the users of online social network expect from the providers of SNS.

2.1 Functionalities and service quality

Users register themselves to a social networking site by giving their personal details when filling the registration form. The personal details include their name, age, e-mail id, likes, dislikes, schooling, graduation, city, country, date of birth, sex, contact information, their own blogging website, and many more to list. Now coming to the reason why do they register to such networking site is to form a social network with friends and family. What makes social network sites unique is not that they allow individuals to meet strangers, but rather that they enable users to articulate and make visible their social networks. This can result in connections between individuals that would not other-wise be made, but that is often not the goal, and these meetings are frequently between latent ties who share some offline connection [3].Most SNSs like Orkut, Facebook require bi-directional confirmation for friendship, else they will not be added into their friend list nor they will allowed to view their profile or photos. A default setting will be already made by such networking sites in order to maintain the privacy of its users. Users credentials must not be revealed to anyone nor it must be saved in any cyber centres and users must be able to change their password when ever they want and any number of times.

Some functionalities and services expected by the users from such OSNs are

- **Storage of personal data** The social networking site must be able to store user data like their personal information which will be displayed in their home page, photos and videos. When ever the user log in to their account they must find data that was updated previously.
- **Security** The main functionality the users expect from an online social network is the security. Since the users give most of their personal details while registering to the networking site, they expect their personal data must be safe and secure because these sites are more vulnerable to many security attacks like data destruction, denial of service and hacking. The user wants their data or profile to be free from such threats and attacks. If such an attack occurs then the user must be able to make a complaint at the police station so that they take actions on the attackers and help the victim to retrieve their information back securely.
- **Editing** After registering and logging into the site users also expect their website to give them some rights to change their personal information. Once the SNS provides them with the editing feature, now the user can set their profile as they wish so as to make them public or private profile. The way they want their profile to appear, updating their education and professional background, relationship status, add or delete photo or video, change profile picture, delete messages, delete post as in case of Facebook.
- **Choice** The networking site must also provide an option where user can select the number of people in their list of friends to whom they want to share their data. Here the user must be able to create groups where they can add friends and share data only to the group members. Non group members cannot view this set of information.
- **Search** Users may also want to find a particular friend in their friends list, which consists of hundreds of friends in it. So if the user wants to find a friend in such a huge list, traversing the whole list will be cumbersome. In order to overcome this, it will be helpful to provide a search button which provides a quick search to obtain a friends profile to the user. This saves much time for the user.
- **Privacy** The service provided by the networking site will help the user to make their profile either public or private. In case of public setting, the details of the user data are available to everyone in the networking site. Here anyone can access the user's information at any point of time. In case of private setting the user data can be hidden or made private from others who try to access their data. Here in private option the user can decide whether they want their data to be viewed by others or just only by them. They can also set privacy option where only their friends can view their information because friend is the one whom they trust, whom they add into their network. One more feature of sharing photos in networking site is whether these are downloadable by others or not. Some of them want to just share their pictures but may not wish that other may download their photo. Such privacy option must also be provided by the online social network.
- **Applications** Some online games must be plugged into these websites, so users with similar interest in a particular game can play the game with each other. Facebook provides a game called Farmville where user create and develop

their own farm. Friends can also visit users farm and benefit by having a gift or increasing their point. Such games provide a good interaction between friends and help to keep in constant contact with each other as they share similar interest and idea.

- **Text Chat** Online chat is also expected by the user as they may not wish to switch from one window to another when they are sitting online in their networking site. User expects the OSN to be more flexible and reliable. Sending private text messages to other friend or non friend should not be visible to others on the network.
- **Audio/Video call** This would be an additional service which makes a user happy. With audio or video call users will be able to talk and see each other. This helps a lot to those friends who are living far from each other or in a different country.
- **File sharing** Users expect that their networking site allows them for file sharing with their friends.
- **Reliability** Users want their OSN to work even under certain hostile circumstances. Users will not accept if their network malfunctions or produces unexpected responses.

If these features are enhanced in an OSN, user will be far more contended. A person's gender, race and ethnicity, and parental educational background are all associated with use, but in most cases only when the aggregate concept of social network sites is disaggregated by service. Additionally, people with more experience and autonomy of use are more likely to be users of such sites. Unequal participation based on user background suggests that differential adoption of such services may be contributing to digital inequality [4]

2.2 Functionality of Facebook, LinkedIn and Twitter

Social networks are popular platforms for interaction, communication and collaboration between friends. Students who live at home likely have much less exposure to their college peers than do students residing on campus. Ironically, by not using a service like Facebook, they are being exposed to their college buddies even less, because not only are they not interacting as much in person as those who live on campus, but they are also not following their peers' activities online.[4] Most of these social networking sites are free of charges except for some like matrimonial websites, which on payment provides complete details of the person whom the user want to contact like telephone number and e-mail id. Users join such groups by themselves or being invited by someone.

2.2.1 Facebook

The current numbers of Facebook users are more than 500 million active users. Facebook allows users to set up personal profiles that include basic information such as name, birthday, marital status, and personal interests. Users establish bidirectional social links by finding other users. Each user is limited to a maximum of 5,000 total friends.[6]. Facebook provide communication, storage and social applications for hundreds of millions of users. A social link exists in an interaction graph if and only if its connected users have interacted directly through communication or an application.

- **Universal Search** Allow users to search for other members profiles and groups. This search button appears on the top centre of the website, allowing users to engage in the search through the entire networking site.
- **Profile** A user network membership determines what information they can access and how their information is accessed by others. By default, a users profile, including birthday, address, contact information, Mini-Feed, Wall posts, photos and photo comments are viewable by anyone in a shared network.
- **Wall** Each profile includes a message board called the Wall that serves as the primary asynchronous messaging mechanism between friends. Users can upload photos, which must be grouped into albums, and can mark or tag their friends in them. Comments can also be left on photos. All Wall posts and photo comments are labelled with the name of the user who performed the action and the date/time of submission.
- **Mini-Feed** Another useful feature is the Mini-Feed, a detailed log of each user's actions on Facebook over time. It allows each user's friends to see at a glance what he or she has been doing on Facebook, including activity in applications and interactions with common friends. The Mini-Feed is ordered by date, and only displays the 100 most recent actions. Other events include new Wall posts, photo uploads and comments profile updates, and status changes.



Figure 1. Home Page of Facebook

- **Groups** Facebook is designed around the concept of networks that organizes users into membership-based groups. Each network can represent an educational institution (university or high school), a company or organization (called work networks), or a geographic (regional network) location.

- **Setting** Users can modify privacy settings to restrict access to only friends, friends-of-friends, lists of friends, no one, or all. Although membership in networks is not required, Facebook's default privacy settings encourage membership by making it very difficult for non-members to access information inside a network.

2.2.2 Linkedln

LinkedIn operates the world's largest professional network on the Internet with more than 100 million members in over 200 countries and territories. LinkedIn uniquely provided Jobs and Companies pages. LinkedIn shows a much lower degree of interaction among users using messages than Orkut. Only 4 percent requests in LinkedIn are related to messaging between users.[2]. The site is available in English, French, German, Italian, Portuguese and Spanish.

- **Search** Allows users to search for company profiles and people connected to it. This search button appears on the top right of the website. Here people look for members with whom they want to make a professional link. Employers can list jobs and search for potential candidates. Job seekers can review the profile of hiring managers and discover which of their existing contacts can introduce them. LinkedIn also allows users to research companies with which they may be interested in working. When typing the name of a given company in the search box, statistics about the company are provided. These may include the ratio of female to male employees, the percentage of the most common titles, positions held within the company, the location of the company's headquarters and offices, or a list of present and former employees.

- **Links** Because LinkedIn is a network used mainly for professional networking (e.g., finding jobs or employees), it is natural to expect that users primarily browse profiles and creates links with each other, rather than exchanging messages.

- **Connection** One purpose of the site is to allow registered users to maintain a list of contact details of people with whom they have some level of relationship. Users can invite anyone (whether a site user or not) to become a connection. However, if the recipient of an invitation selects *I don't know*, this counts against the person inviting them, and after five such I don't knows a member cannot invite another to connect without first supplying their recipient mail address. This list of connections can then be used in a number of ways, A contact network is built up consisting of their direct connections, the connections of each of their connections and also the connections of second-degree connections. This can be used to gain an introduction to someone a person wishes to know through a mutual contact. It can then be used to find jobs, people and business opportunities recommended by someone in one's contact network.

- **Share an Update** Share an update This gives updates of what members are doing in the site, whom are they following and whether they have changed their profile picture.

- **Follow** Users can follow different company profiles and can get notification about the new joining and offers available.



Figure 2. Home Page of LinkedIn

- **Bookmark** Users can save (i.e. bookmark) jobs which they would like to apply for.
- **Audio/Video Sharing** Audio/Video Sharing – Video and photo features are not supported in LinkedIn.
- **Suggestion** This gives a list of friends whom we may know but are not connected. This is done through mutual connection between two users.

2.2.3 Twitter

Twitter is a website which offers a social networking and micro blogging service, enabling its users to send and read messages called tweets. Tweets are publicly visible by default, however, senders can restrict message delivery to just their followers. It's a new and easy way to discover the latest news what's happening related to subjects you care about. Messages from users you choose to follow will show up on your homepage for you to read. Once you've followed a few people, you'll have a new page of information to read each time you log in. Click on the links in others Tweets to view images and videos they have posted, the profiles of users mentioned in their message

- **What's Happening** Each profile have their own message block where users can update their message which will be visible to the people who follow them. This can be made public or private depending on the users requirement.
- **Messages** Tweets are text-based posts of up to 140 characters displayed on the user's profile page
- **Timeline** Timeline is a long stream showing all Tweets from those you have chosen to follow on Twitter. Newest messages are at the top. You can interact with messages from within the timeline by hover your mouse over to reply or Retweets. Clicking anywhere on a Tweet in your timeline pulls open a sidebar, where you can see photos, videos, and profile information related to that message.

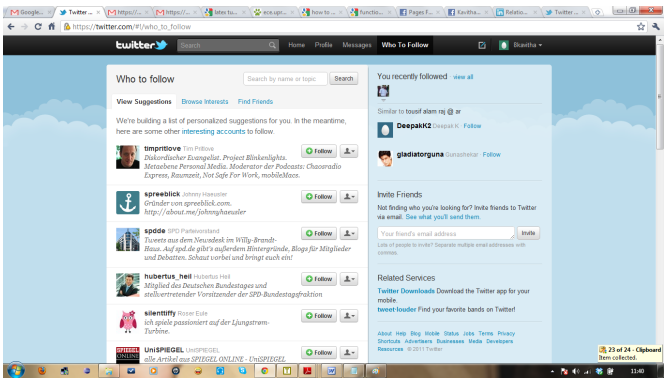


Figure 3. Home Page of Twitter

- **Follow** Users may subscribe to other users' tweets, this is known as following and subscribers are known as follower. When you choose to follow another Twitter user that user's tweets appear in reverse chronological order on your main Twitter page. You can follow people in any of the following ways via the web, SMS/text message or third-party application. Followers are people who receive your tweets. If someone follows you, they become your follower and will display in the 'newest follower' section, receive your tweets in their home page, phone, or any application that they use, If you follow someone, you are their follower.
- **Unfollow** Removing someone from your following list means that you'll no longer receive their updates on the web.
- **Browse suggestions** You can pick and choose which accounts you'd like to follow by clicking on the users page. Then, you can then select a different interest category to find more great accounts to follow.
- **Limits** Direct Messages that can be sent in a day is 250 and the maximum update is 1,000 per day. Here Retweets are also counted as updates.

In the entire above mentioned and other social networking site the common feature found is the user is trying to make a network of friends and family who can keep track of what message does the user wants to convey. The user updating their status on their homepage which will be visible by others members who have been added to their friend list. The other features were browsing other member's profile, viewing their photos, commenting on pictures. Security features are also been provided to make the users profile public or private to other members. Other than the messages that appear publicly on the user's homepage, an option of private message is also being provided. Photo album features is provided by most of the networking site except for the LinkedIn.

2.3 Non-functional requirements

Non-functional requirements or behavioural requirements describe how well a system performs its function. This is fundamentally different than the typical functional requirements which describe what that system does. A good non-functional requirement is one that makes it clear to everyone how the system performs.

- **Maintenance Cost** The maintenance cost for such online social network must also be considered. If developing such OSN requires more money then only way to gain it back is only through the advertisements or by selling user data.
- **Maintainability** How the lost data can be retained or restored like the way it appeared before with in a given period of time constitutes for the maintenance of the OSN and how easy it is correct the defects of OSN. Maintainability involves a system of continuous improvement from the past in order to improve the ability to maintain systems, or improve reliability of systems based on maintenance experience.
- **Good user interface** Any social networking site must provide an easy handling of its tools and other features of the system. If the features or services are difficult to understand then there are more chances of users leaving such social networking site and looking for better ones. Any user interface requirement will demand application navigation and good user assistance all through out the application. Overall the look and feel of the online social networking site must be good enough for the user to access it in an easier way.
- **Usability** Users often have different usability requirements and Users also have different data access requirements. Any system should be able to meet these requirements of its users. There are options like help pages where the user clicking the question mark on the toolbar or help menu option in the menu bar, user must be able find answers for their questions related to the networking site they use.
- **Recoverability** Recoverability is the ability to restore function and data in the event of a disaster, either natural or man-made. Recoverability ensures that if any kind of system failure occurs, regardless of the reason, the system will operate with minimal interruption. If the users system crashes, it should be possible for the user to access their data when they login via another system to such networking site. All their data stored in their homepage must remain the same without any deletion.
- **System Maintenance** There should always be regular system maintenance. You should think about how much time you need to do maintenance, and how and when you will notify users.
- **Efficiency** We also need to check how efficiently OSN provides the given output when the user request is made.. Efficiency is a measurable concept, quantitatively determined.

- **Performance** Performance- Performance details the way the system will perform for users where we need to deal with response time for the queries being asked by the user, the total amount of idle time before the user terminates their session, number of concurrent sessions opened by the user.
- **Safety** Safety is the state of being safe. The condition of being protected against consequences of failure, damage, any other event which could be considered non-desirable. Some kind of precautions must be kept in hand in order to face such failures like system crash, profile deleted, photo album being removed or hacking your credentials.
- **Documentation** OSN must aim at providing user guide to its users who find difficulty in performing their task related to a particular social networking site, online help, reference guides. Documentation is the process of providing evidence.

2.4 How quickly the search be reduced?

In any online social network the search for friends or communities must be as quick as possible producing relevant details of the required information. The search should be in such a way that user must be able to find all the similar kind of text pattern, once the user types a letter or two. For example if the user types the letter A in the search box, all the friends name or community name starting with this letter should be displayed so that the user can find the relevant friend in a very quick manner. The functionality must satisfy the user requirements. User expects his social network site to be as good as possible with all the functionalities being performed without any problems. The user expects his search result to be as quick as possible. The maximum load on each user depends on the data the user stores in their profile. This saves the time of the user where there is no need for him to traverse the whole bunch of list in the networking site. As every user might have hundreds of friends and it is impossible to traverse from the beginning until end of the friend list, this would be cumbersome for any user to traverse sequentially all through out the list. Even though the network is working well and good a quick search for peers or data will be an additional advantage to any online social network. The number of users in OSN may increase during the intial stage of the due to the enthusiasms and invite many other to join this networking site, next will be some dip in the users. Here user now know what features or application their networking site provides them, they loose interest and naturally dip out and finally there will be a steady users who login to this site and use them for their networking without loosing any interest [**?**].

3 Provider′s Point of View

The topic discussed here about workload statistics is based on the work of Fabrcio Benevenuto, Tiago Rodrigues, Meeyoung Cha and Virglio Almeida.

3.1 Workload statistics of usage pattern

Based on the survey of the authors [2] on detailed click-stream data, collected over a 12-day period, summarizing HTTP sessions of 37,024 users who accessed four popular social networks: Orkut, MySpace, Hi5, and LinkedIn gives a brief idea about the workload statistics. The analysis demonstrates the power of using clickstream data in identifying patterns in social network workloads and social interactions[2]. Clickstream data is the number of times the user clicks on webpage and this helps in the analysis of the user action being performed on an online social network. Now this clickstream data can be used to check all the activities that a user performs from the starting of their session by logging in until they logout from the network. This may comprise of how many times does the user check others profile, duration of their session between logging in and logging out, how many times a user visits their online social networking site per day, whether he deletes any photos or posts any message. All these activities are recorded via the clickstream. Analysis shows that browsing, which cannot be inferred from crawling publicly available data, accounts for 92 percent of all user activities [2].

Understanding the way user use social network site, gives a better view for the provider what need to be improved and on which aspect more effort should be put in order to improvise them so that the user have a good interface with the social network site. In order to observe the activities of the user, the workload is obtained by gathering the patterns of data such as messages, post, comment but these data contribute for the activities that can be seen. We need to think even about those activities which are not visible like visiting a friend's profile. Hence here a clickstream can be used which keep tracks of all the activities both visible and invisible.

Understanding how users behave when they connect to these sites is important for a number of reasons. Studies of user behaviours allow the performance of existing systems to be evaluated and lead to better site design and advertisement placement policies. Accurate models of user behaviour in OSNs are crucial in social studies as well as in viral marketing. For instance, viral marketers might want to exploit models of user interaction to spread their content or promotions quickly and widely and third, understanding how the workload of Social networks is re-shaping the Internet traffic is valuable in designing the next-generation Internet infrastructure and content distribution systems [2]

Workload with respect to provider is how the provider handles a large number of users in an online social network. Understanding workload characteristics is very important for optimizing and improving the performance of current network. Usage pattern varies with the total number of users accessing the social network site. The usage pattern also varies on the time the user accesses the network site. More user use these site during weekdays and on weekend the usage is less [2]. The duration in which user uses the SNS does not necessarily mean he is active all the time. The user may login and at the same time he may be engaged in some other work without checking this network site. Some of the patterns used are like in case of Facebook poke is used as a provocation pattern which allow one user to take action on another user. OSN allow user to create artefact for others like joining some music groups which will be displayed on user's home page. There are several applications like quiz, online games which allows user to share objects with other users collectively. Workload also contributes to the number of friends a user has and the activities carried out by the user such as navigating through the website, making changes in the settings of the home page, visiting friend album, viewing profile of friends of friends and so on.

3.2 Usage vary in accordance to age, country or social structure of the user

In any online social network there are millions of users of different age group. The usage varies with age, country and social structure of the user. All these users use this network site depending on their own interest. School students may look for friends, graduates may look for some jobs and also friends, some might have registered just for the sake of their friends forcing them to do so. Usage does vary a lot with age as there the school students may not be allowed to access internet at home for such online social network site, so they may use it only when they are outside. Bigger students use internet everyday in their life for their education and learning so these set of age group are not restricted with the usage of OSN and coming to professionals they use such network site only when they have time apart from work, when they are free they login to network site and start their network.

The data stored in user profile gives us a view or an impression on that person, this is like person portraying himself to the rest of the world. Other member of social network decide whether to make friendship links or not by viewing users profile. Usage also varies from one country to another. People of the same country have registered to social network site where most of their population have the network. for example like in India and Brazil the most used social network site is Orkut [2], like in Europe the very large social network is formed by Facebook [6]. In this way usage does vary across different country and culture.

4 Conclusion

Online social network has made a great impact among people all over the world by its feature to connect to family and friends. Here in this paper we have mentioned only about three such networks such as Facebook, LinkedIn and Twitter. There are many other social networking sites such as Amazon, Flickr, Friendster, hi5, Myspace, Orkut and many, in each of these website there are many users who are engaged in network in day to day life. All these network sites vary only in their structure and how user access them but overall functionality is to form a network of friends. Social networking have attracted the users in such a way that life without network is unimaginable. Users also create their own communities, groups and they may even join a large group which consists of very large set of users who may or may not be connected to each other. Groups such as a novel, artist, musician, actor, movie and many more to list. In this paper we presented mainly on two parts, one is with respect with the users point of view and the other is with respect to providers point of view. Here we have mentions about what users expect from the providers like security, fast access, and free registration and so on. The providers want to check the usage of the users in their social network site so that they can enhance their feature by examining what users use the most in such network site. This helps the providers to provide more advanced features to their site. In this paper I have mentioned about some features of Facebook, LinkedIn and twitter and how they perform. It has also been mentioned how people of different age group use such online social network

References

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[2] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proceedings of the SIGCOMM on Internet measurement conference*, 2009.

[3] D. Boyd and N. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 2008.

[4] E. Hargittai. Whose space? differences among users and non-users of social network sites. journal of computer-mediated communication, 2007.

[5] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. *Link Mining: Models, Algorithms, and Applications*, 2010.

[6] I. OMurchu, J. Breslin, and S. Decker. Online social and business networking communities. In *Proceedings of Workshop on Application of Semantic Web Technologies to Web Communities*, 2004.

[7] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao. User interactions in social networks and their implications. In *Proceedings of the European conference on Computer systems*, 2009.

PAST - Replication and Data Storage in FreePastry

Pascal Kallien
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: pkallien@mail.upb.de

Abstract

This paper focuses on the task of data replication in a distributed online social network. At first glance, the need for data replication is motivated and desirable goals of data replication in a distributed online social network are presented. This paper presents PAST – a large-scale peer-to-peer persistent storage utility – and discusses if this utility is suitable for the use in a context of a distributed online social networks. Drawbacks of PAST are discussed and related work is examined to present opportunities or workarounds to overcome these drawbacks.

1 Introduction

Online social networks like Facebook, Myspace or Twitter get more and more popular. Considering Facebook, there are more than 600,000,000 [3] registered members. Every member may contribute content like photos or text messages to a social network. Most of the online social networks implement the server-client model, so the user data is stored on the companies servers. Due to the huge amount of data, this has several drawbacks.

First of all, the costs to maintain the servers are high. To address this problem, the Facebook founders published within the open computing project [4], how they implemented their own server architecture, reasoning that conventional systems are not able to handle the huge amount of data efficiently.

Another problem of the central storage model is the privacy. All the user data is stored and accessible through the central storage servers. So the company has full control over the stored data. In addition, most of the social networks are commercially orientated or want at least to cover the server costs. A popular financing model is to sell the users data to marketing companies, which is often criticized by privacy groups [1].

An alternative approach to avoid these problems is to develop a distributed online social network. In a distributed online social network, every user contributes computing power, storage space and bandwidth to the social network. To implement this, a distributed framework based on a peer-to-peer network, is developed.

This paper concentrates on defining and implementing concepts for data replication that best suits in the context of a distributed online social network. Different to the server-client model, the data in a peer-to-peer network is not stored on single machines that are expected to have a high performance, availability and durability. Instead, the data is stored distributed on many different peers, which have different performances and an unpredictable online behavior. Data replication is the mechanism of storing the same data on several different peers. This is needed, to ensure that the distributed data is accessible, even if a responsible peer could fail anytime, e.g. it goes offline without any notifications to other peers in the network. Another benefit of data replication is load balancing, which allows efficiently handling of popular data. The desired goals and challenges for data replication in a distributed online social network are presented in Section 2. In Section 3, this paper introduces the basic concepts of PAST, a "large-scale, Internet-based, global storage utility that provides scalability, high availability, persistence and security" [7]. In addition, this section discusses some crucial drawbacks of PAST and examines, if this drawbacks are too bad and prevent us from using PAST as data replication system in our distributed online social network. Section 4 presents projects that also need to handle data replication in peer-to-peer networks. These projects may present solutions to overcome the discussed drawbacks of PAST. Finally, Section 5 concludes how the findings of Section 3 and 4 could be used to implement data replication in a peer-to-peer network for our distributed online social network, that agree with the definitions of Section 2.

2 Data replication

This section gives a brief overview of data replication and defines some demands on data replication that especially are desirable in a framework for distributed online social networks.

2.1 Definition

Since the data in a peer-to-peer network is stored on different peers, each peer that stores data has to be online to ensure access to every data object[1]. As this condition is not wanted and possibly not feasible, data replication is used to avoid it. Therefore, the original object is replicated on several other peers. There are four basic operations within data replication, namely *read, write, update* and *delete*, which all take as input the identifier of the relevant data object.

The write operation inserts or modifies a data object in the network, while the read operation accesses it. The update operation has the purpose to ensure consistency between the replicated objects. The peer on which the data object was modified has to find and notify all other peers with replicated objects and send them the modified data. This should be done automatically, or in other words, the user does not have to initiate this operation. Finally, the delete operation, deletes the respective object from the peer and additionally all replicated objects.

2.2 Challenges and Goals

There is a tradeoff between high availability and storage space. The more replicated objects exist, the higher the possibility to find an active peer holding a replica of the demanded object. But the more an object is replicated, the more storage is used for redundant copies. The challenge is to find an ideal amount of replicated objects to ensure high availability with a minimum amount of required storage space as possible. Additionally, it is hard to keep all the replicated objects up to date. As mentioned previously, all peers with replicas have to be found and notified, so that all replicas store the latest version. Since peers can concurrently modify the same data object on different replicas, it is eventually hard to decide, which is the latest version.

In our distributed online social network, a client should have the possibility to access the same kind of data that is intended for him, even if he switches the client machine. The following example illustrates this. A client/user adds a new friend to his friend list on his desktop machine and subsequently logs himself out of the network. Later he logs in with the same credentials on his mobile computer. The recently added friend should now also be in his friend list on his mobile computer.

The example illustrates three operations of data replication. The user *writes* the new added friend to his friend list, which is accessible through a data identifier. After this, the replication mechanism has to *update* all replicated files with the respective identifier, so that the user can *read* the modified friend list on his mobile computer. This example points out, that it is not enough to store this information only on the participants machines and additionally reasons the need for data replication. The example also points out, that the network must not be assumed to be homogeneous. A desktop computer may have more CPU, bandwidth and disk storage than a mobile computer. Indeed, more and more smartphones populate the Internet with the users focus on social networking. This smartphones may not contribute storage space for replication to the network, due to their low bandwidth and storage space, but they may want to intensively use the network.

Another indispensable requirement is the need for deleting or modifying the data in the network. The users of the distributed network should be able, to delete files like their pictures and modify or update their profile information and friend list.

Modifying and deleting data within data replication is not trivial, due to consistency problems. An acceptable solution must find and notice all replicating clients to delete or modify the relevant data. Replicating files on different peers has the consequence that probably private data is distributed to untrusted peers that are not allowed to examine the data. Therefore the replicated data has to be somehow encrypted, so that only authorized clients are able to decrypt the data.

In addition, authentication of the data is useful to be sure that the received data is not manipulated by a replicating peer.

3 Implementation

This section presents implementations of routing and data replication in a peer-to-peer network. At first it describes Pastry in Subsection 3.1, since it is the used content location and routing scheme in PAST. Subsection 3.2 discusses PAST, the data replication utility on top of Pastry, while Subsection 3.3 points out some drawbacks of PAST. An open-source implementation of Pastry — called FreePastry — can be found at the FreePastry Project website [2].

This section is based on the work of P. Druschel and A. Rowstron [9, 7, 10].

[1] depending on the granularity, this could be e.g. a file or a data block

3.1 Pastry

Pastry is a scalable, fault resilient, self-organizing and efficient substrate for peer-to-peer applications [10]. Pastry is able to route messages to peers within the network by only needing $\lceil log_{2^b} N \rceil$ hops, where N is the number of peers in the network and b being a configuration parameter[2]. The core idea behind Pastry is, that every peer — in the following called node — stores shortcuts to other nodes in the network. Every node can be addressed by its so called nodeId, which is a unique identifier. The nodeId is a value between 0 and $2^{128} - 1$ and is assigned randomly when the node enters the network. Since the nodes with nodeId 0 and $2^{128} - 1$ are defined to be adjacent to each other, the nodeId space can be seen as a closed circuit. The nodes are only adjacent within the Pastry overlay, so they may have a different location, since the nodeIds where assigned randomly. This has some benefits, which will be described later on in Section 3.2. Below is a description of Pastry's routing algorithm and the needed data structures.

Assuming a message m has a numerical key k, then m is routed towards the node t, which nodeId most closely matches to the prefix of k. It is not feasible, that every node stores routing information about all the other nodes in the peer-to-peer network and it is not efficient to delegate the message step by step through adjacent nodes until t is reached. Therefore, Pastry's routing algorithm takes advantage of two data structures, namely the *leaf set* and a *routing table* with shortcuts, which contents are different for every node.

In the leaf set, each node stores the nodeIds of the $\frac{|L|}{2}$ adjacent nodes *right* to it and the $\frac{|L|}{2}$ adjacent nodes *left* to it[3], with L as the size of the leaf set[4].

In the routing table, each node stores shortcuts to distant nodes in the nodeId space. Each routing table has $\lceil log_{2^b} N \rceil$ rows. Each row can be seen as a *level* and stores routing information to other nodes. The higher the level l is, the more the nodeIds of the nodes in this level match to the nodeId of the respective node, namely the l most significant digits of the nodeId are equal[5].

With these data structures, an efficient routing algorithm is possible. If the message m can directly be routed from node s to the target node t, k must be within the range of the nodeIds in the leaf set of s. Otherwise, m is routed towards a node within the routing table of s, which most closely matches with k and the operation is repeated recursively until m is delivered to t.

Figure 1 illustrates the routing algorithm of Pastry. In this example, the value of the configuration parameter b is 4. This figure represents only the boundary nodes 0 and $FFFF...F$ of the nodeId space and the nodes involved by the routing algorithm, so imagine there are several more nodes between the represented nodes. The node s_1 with nodeId $75A1FD$ wants to route the message with key $E46A1E$, but since it is not in range of his leave set, it is routed to the node s_2 with nodeId $E13DA4$, since this is the node in the routing table of s_1, which most closely matches the key. Assuming a high populated network and randomly generated nodeIds, the amount of nodes between s_1 and the target node t could nearly be halved with the help of this shortcut. This procedure repeats, until the message is delivered.

An additional data structure in Pastry is the neighborhood set M. In this set M, each node stores the nodeIds of geographically close nodes. Since this set is not used for routing, it is not of further interest for us.

A node sends repeatedly messages to the other nodes in its leaf set, to signal that it is still alive. Thus, when a node notices that a neighbored node has not send a live signal for a certain time, it is assumed to be dead. In this case, the affected nodes update their leaf sets. For detailed information of Pastry, it is recommended to take a closer look at the work of Druschel and Rowstron [9].

3.2 PAST

PAST is a "large-scale peer-to-peer persistent storage utility" [10]. The following presentation of PAST is also based on the work of Druschel and Rowstron [7, 10].

3.2.1 Basic procedures

Since PAST is a replication layer based on top of Pastry, each node in PAST is also identified by a 128 bit nodeId. In addition to this, each file inserted to the network can be addressed by a 160 bit identifier, called fileId. This identifier can be calculated by hashing the "file's name, the owner's public key, and a randomly chosen salt" [10].

An inserted file is stored to that node, whose nodeId most closely matches to the prefix of the inserted file's fileId. This node r can be found via Pastry's routing algorithm, with the fileId as the key of the routed message. Because PAST is

[2] b depends on the used notion, i.e. $b = 4$ in a hexadecimal system

[3] imagine the nodeId space as a closed circuit

[4] L is a parameter that is defined a priori

[5] assuming $l = 0$ in the first row

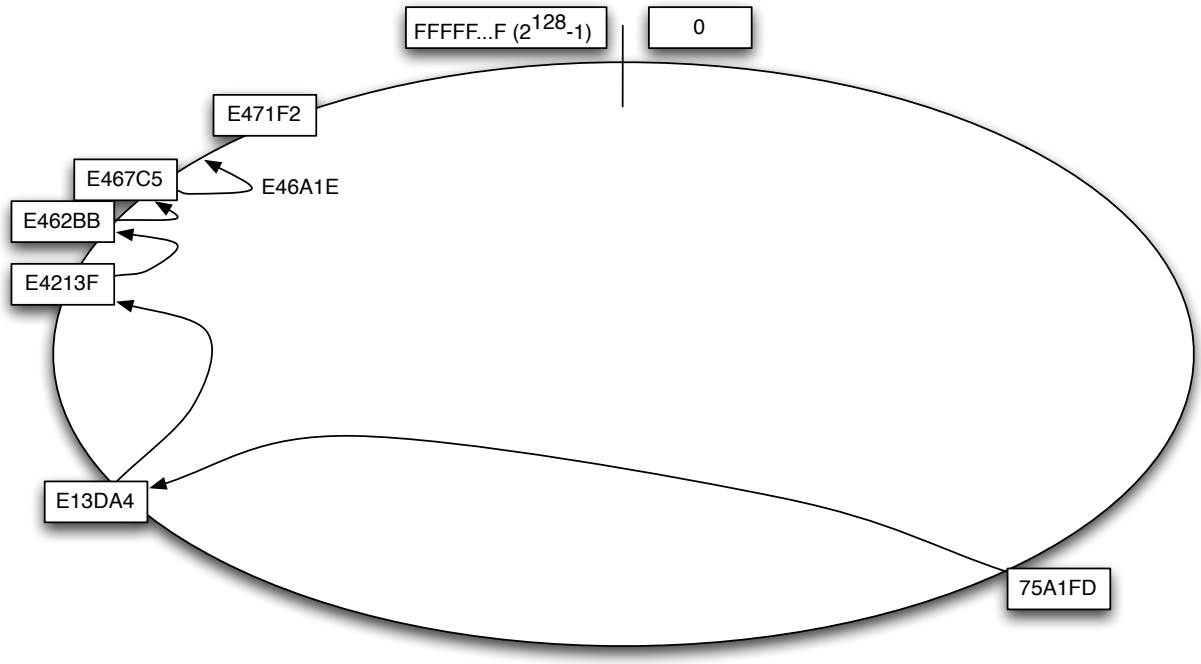


Figure 1. Routing a message with key E46A1E from node 75A1FD in Pastry.

a replication layer, this file is not only stored once at node r, but is replicated to $k-1$ additional nodes. The remaining nodes are the $k-1$ other closest nodes regarding to the fileId and can possibly be found in r's leaf set. Here asserts the earlier mentioned benefit of Pastry's random nodeId generation. Due to this, the replication nodes for the same file may be distributed to different locations, so the access latency can be decreased. Additionally these nodes are not correlated to local disasters, like earthquakes for example. During this insert operation, it is ensured that the file has not been corrupted or manipulated during transfer. This is achieved by creating an additional *file certificate*, which includes the "fileId, a SHA-1 of the file's content, the replication factor k, the used salt" [10] and a timestamp, and is send along with the file. With the hash of the file's content, the receiving node r can itself calculate the hash of the received file and check if it is identically to the hash in the file certificate, to ensure the data integrity. To even be sure that actually k replicas were created, each successful replicating node sends an acknowledgement message — called *store receipt* — back to the origin file insertion node. Besides this, it is also verified that the client is allowed to insert the file, e.g. if his remaining storage quota is bigger than k multiplied with the file size. In cases of any errors, the client is notified.

Figure 2 illustrates replicating in PAST, with replication amount parameter $k = 5$. Like in Figure 1, this figure also represents only the boundary nodes 0 and $FFFFF...F$ of the nodeId space and the nodes involved by PAST's replication, so imagine there are several more nodes between the represented nodes. The node i with nodeId $75A1FD$ wants to insert a file with fileId $E46A1E$. The responsible node r with nodeId $E467C5$ is found via Pastry's routing algorithm. r checks the integrity of the file and after successful validation, it distributes the file to the 4 remaining *closest* nodes.

Another operation in PAST is the lookup operation. The purpose of this operation is to access the stored file. Therefore, it takes the fileId of the respective file as input parameter. With this parameter, the responsible node r can easily be found via Pastry's routing algorithm. Additionally to the file, r also sends the file certificate back to the origin lookup initiation client, so that it can validate the data integrity.

A further operation in PAST is the reclaim operation. It takes as input parameter the fileId and the file's owner-credentials. As a result of this operation, the availability of the respective file is no longer ensured by PAST, but the file itself is not actually deleted. Before this operation is executed, it is ensured that the reclaim initiator is the file owner.

As empirical studies — later presented in Section 3.2.4 — show, these operations itself are not suitable to ensure successful file insertion. Therefore, PAST has two mechanisms to handle this, namely *replica diversion* and *file diversion*.

The first mechanism — *replica diversion* — has the aim to balance the load within a leaf set L. When a node r runs low

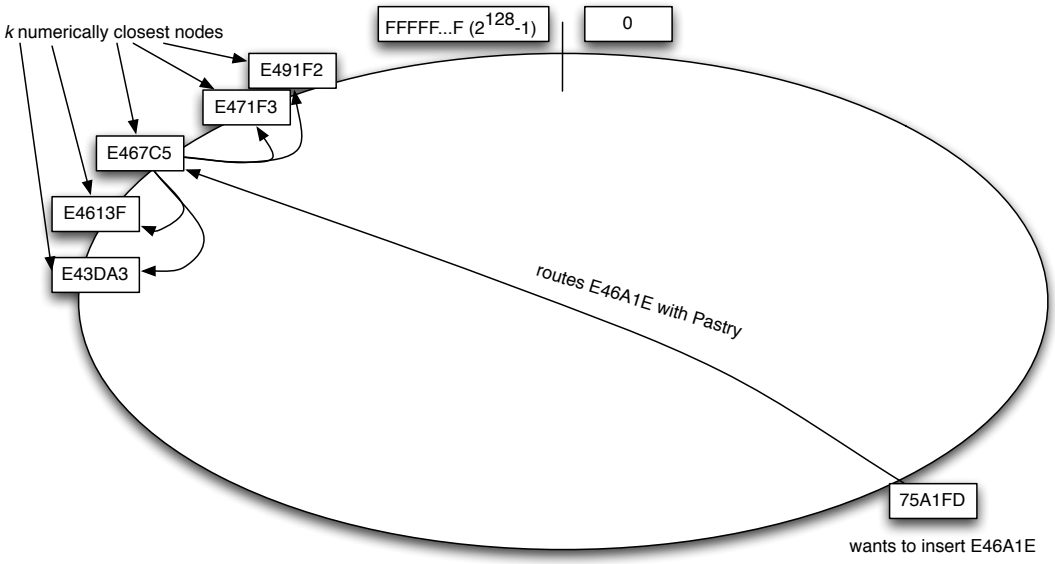


Figure 2. Inserting a data object with fileId E46A1E in PAST.

on storage, it may delegate an incoming replicate request to another node t within its leaf set, that has more storage available and was not previously selected for replicating the respective file. If t accepts the request, it stores the file and r stores a pointer to t. To ensure that the copy in t is still accessible, even if r fails, there is additionally added a pointer to t in the $k+1$-th closest node u. If r fails, u becomes responsible for the replica. Though it already contains a pointer to t and so the invariant of k replicated files is still fulfilled.

The second mechanism — *file diversion* — is initiated when a node's complete leaf set L is reaching its storage capacity. In this case, the file is distributed to another part of the nodeID space in PAST. This is achieved by choosing a different salt value, which generates a different fileId.

To increase the effect of this two mechanisms, Druschel and Rowstron present three policies, which define,

1. when a node accepts an incoming replication request
2. which node is selected for replica diversion
3. when to initiate file diversion.

A node r accepts an incoming replication request of the file f, if the size of f divided by the remaining storage space of r is smaller than a threshold $tres$. The threshold $tres$ is a parameter that is defined separately for the primary replica nodes ($tres_{pri}$) and the diverted replica nodes ($tres_{div}$), with $tres_{pri} > t_{div}$.

The selected node t for a diverted node fulfills the following condition. As previously mentioned, t must be within the leaf set of the initiating node r. From this nodes, t is the node with the most remaining free storage space and must not have been selected previously for replication of the respective file.

The policy for initiating file diversion implies, that file diversion is initiated after a node failed two times when trying to divert the replica to another node within its leaf set.

3.2.2 Caching

Another component of PAST is caching. Caching is used to decrease the latency of retrieving popular files within the network. This is implemented, by allocating a fraction of the unused space of the nodes free disk space as a cache. Each cache contains the files, that were previously forwarded from the respective node. A file is deleted from the cache, until the nodes used fraction of the storage is needed for replicas or new routed files are replaced in the cache. The corresponding caching policy is based on the GreedyDual-Size [6].

3.2.3 Security

PAST additionally includes a security model. Unfortunately the security model is based on the use of smartcards and brokers as third parties, that issue those smartcards, what makes it unsuitable for a non-commercial oriented, distributed online social network. Nevertheless some of the key aspects are described below.

The security model in PAST assumes, that most of the nodes in the network are behaving friendly. Further it is assumed that the content of the smartcards is not mutable through attacks from outside. Like common in cryptography, it is also assumed that cracking the used keys is vain, since its assumed that breaking the system takes more time, than the content of the encrypted data is worth.

In this model, every client in PAST has a private/public key pair, which is stored in its smartcard, that he obtains by the third party brokers. The nodeId is generated by a hash of the public key of the smartcard. Thus, each node can validate the authenticity of other nodeIds.

The keys in the smartcard are used for authentication and verification mechanism, so it is ensured that for example a reclaim operation is only permitted by the files owner. Another function of the smartcards is to manage the storage quota of each user. Within each insert operation the storage quota of the inserting client is debited with the size of the file multiplied with the amount of replications k. Details of the processes can be obtained in the work of Druschel and Rowstron [7]. A native support for data encryption is not implemented, but the users may decide themselves to use a cryptosystem. The usage of smartcards is not mandatory, but best practices without smartcards are not yet specified by the authors.

3.2.4 Performance

A brief overview of experimental results presented in [10] on simulating PAST is listed below.

The experiment setup contained two different workloads, the first was based on web logs of the NLANR[6] and the second was generated from 2,027,908 files, stored on the computers used by the authors, with a total file size of 166.6 GBytes.

The parameter for "the number of replicas k for each file was fixed at 5, b was fixed at 4 and the number of PAST nodes was fixed at 2250" [10]. The mean storage capacities were adjusted to the workloads to experiment with high storage-utilization.

During the tests it was verified that PAST is able to achieve the advertised behavior of maintaining data replication, as long as Pastry is able to maintain proper leaf sets. Nevertheless, the nodes in the network need sufficient storage space, to be able to store the files and their replicas.

Within a first experiment, the usage of file diversion and replica diversion was not allowed. The results show, that actually 51.1% of the insert operations failed, but the used storage was only at 60.8%.

In a second experiment, the usage of file diversion and replica diversion was allowed, using the thresholds $tres_{pri} = 0.1$ for the primary replica nodes and $tres_{div} = 0.05$ for the diverted replica nodes. Additionally, there are two different sizes of leaf sets with $|L| = l = 16$ and $|L| = l = 32$.

The results of this experiment show, how effective the use of file diversion and replica diversion really is. The storage utilization has risen to $> 94\%$ with $l = 16$ and $> 98\%$ with leaf set size $l = 32$.

In addition, the success rate of the insertion operations increased to $> 94\%$ with $l = 16$ and $> 97\%$ with $l = 32$.

Further experiments with different leaf set size l show, that there is no benefit in using larger leaf sets than 32. Additional experiments with different thresholds show, that the bigger the thresholds are, the bigger is the storage-utilization but the lower is the success rate of inserting big files.

The authors therefore recommend as outcomes of these result to use the thresholds $tres_{pri} = 0.1$ and $tres_{div} = 0.05$ [10].

3.3 PAST drawbacks

While the concept of PAST is simple and there already exists a free implementation of Pastry and PAST, there are drawbacks that prevent it to be used unmodified in a distributed online social network. These drawbacks are listed below.

First of all, PAST does not support the modification of the entered data. Furthermore it does not support the deleteability of existing data. As mentioned in Section 2, this is a desirable goal for data replication in distributed online social networks. When implementing a modification function, it must be noted to also implement an update mechanism, that notifies the replicating and caching nodes, since PAST does not offer such a mechanism. Otherwise, this could lead to data inconsistency, when a modified version exists, but instead an older file within a nodes cache is forwarded.

Another disadvantage is the assumption of a homogeneous network. It is assumed, that the storage capacities of the individual PAST nodes do not differ by more than two orders of magnitude.

[6] National Laboratory for Applied Network Research, ftp://ircache.nlanr.net/Traces

Since the authors do not present best practices without smartcards, most functions of the included security model cannot be used in our distributed online social network. The storage quota for example, is managed by the smartcard [7]. Thus, without a smartcard obtained by a trusted third party, each user possibly could select the storage quota themselves. Due to this, the users could possibly flood the network with a lot of files or select an abnormal high amount for the k replicas. Since a client can only verify the insertion and replication of a file by obtaining k *store receipts*, there is no guarantee that the file was actually saved. So, attackers could accept all incoming requests but then reject or delete the file immediately. Furthermore, if attackers could choose their nodeId and are able to generate k nodes with numerically close nodeIds and all of them are responsible for storing and replicating the same files, this could lead to irreversible data loss. Since the *file certificate* is signed by the smartcard, its non-existence does not allow this anymore. As a consequence of this, the authentication of the file certificate is no longer possible. Furthermore, the authors do not present mechanisms to prevent (distributed) denial of service attacks with lots of lookup request to a single peer. In Addition, there is no native implementation for data encryption.

For this reasons, the following section considers some other related work, where some alternative approaches for data replication are presented.

4 Related Work

This section presents two related works. These have in common, that they are dealing with data replication in distributed networks. The first project has a mechanism that allows mutable files, even if the data blocks itself are immutable. The second project presents an approach on how to dynamically change the amount of replicated objects and reasons why further research on scheduling data transfers between peers is not necessary. The examination of these two projects and their solutions may give some extra information about how to fix the drawbacks occurred in PAST or possibly give some alternative approaches.

4.1 OceanStore

This subsection describes the data replication subsystem behind the global-scale storage system OceanStore. Section 4.1.1 gives a brief overview of the purpose of OceanStore and 4.1.2 of the underlaying architecture. The concepts of data replication in OceanStore are described in Section 4.1.3. A prototype of OceanStore — called Pond — and its source code can be downloaded on the OceanStore Project website [5]. The following descriptions are based on the work of D. Geels [8].

4.1.1 Overview

"OceanStore is a persistent data store" [8] that can handle storage of more than 10^{18} bytes (1 exabyte) to billions of clients. The author assumes, that the participant clients pay a monthly fee and therefore except a high durability and availability of the data. The users' data is always encrypted by the clients and the involved servers only work with the encrypted data.

Unlike peer-to-peer networks, the OceanStore architecture consists of several servers that build the *inner ring*, which stores and manages the update operations on the data. Additionally unlike to peer-to-peer networks, where peers are expected to fail at any time, in OceanStore all the data is stored at these servers, where the availability and durability of them is expected to be high. Thus, the usage of data replication has other reasons than in our framework for a distributed online social network.

First of all, data replication in OceanStore is used to unload the utilization of the inner ring servers. Due to many request to these servers, they soon become a bottleneck.

An additional reason for data replication is to minimize the access latency. Clients that are far away from the inner ring servers will always have a higher latency.

Nevertheless, data replication is still helpful to ensure data access even if the inner ring servers fail.

4.1.2 Structure

Different to PAST, the computers in OceanStore use the overlay Tapestry for communication, instead of Pastry. Every computer is addressed by a 160-bit globally unique identifier (GUID). Storable objects are also identified through GUIDs[7]. A machine that stores an object, sends a message (*publishes*) to machines which GUIDs closely match the GUID of the stored object. The addressed machines remember the publishing machine and forward incoming access request to the data object to the relevant machine. So, the numerically closest machines store only a pointer to the regarding data object. This is different to PAST, where the k numerically closest machines store a copy of the data object. The OceanStore approach may lead to one more unnecessary hop, compared to PAST (but only if the according PAST node has not initiated file replication). A benefit of the OceanStore approach asserts, if more machines store the same data object. In this case, Tapestry automatically forwards the access request to machines that are near the requesting client.

[7] these are also called AGUIDs

As mentioned earlier, each client communicates with inner ring servers. The servers take advantage of a byzantine-fault-tolerant agreement protocol to verify that the servers can maintain client requests, even if servers fail. In practices, each client is assigned to four servers. With regard to the Byzantine-fault-tolerant agreement protocol, one server is allowed to fail. The mapping of clients to inner rings servers is implemented through each client's OceanStore Service Provider.

Another feature of OceanStore is the native support for versioning. This allows mutability of files, even if stored data blocks are immutable. This is implemented by mapping a single objects name to a chain of versions. Each version of a file consists of some metadata and the encrypted data of the client. The metadata includes a timestamp, the version's sequence number and the name of the previous version. The client's data is stored within a btree, with cross-references to the relevant data blocks. When inner ring servers generate a new version of an object, it only needs to store newly or modified data blocks.

"The set of versions comprising a single data object are" named *active GUID (AGUID)*, which is "equal to a secure hash" [8] of the data object's file name and the owner's public key. When needed, the inner ring servers generate the mapping of an objects most recent version. These mappings are called *heartbeat*.

4.1.3 Replication

The approach of solving the earlier mentioned problems, is to replicate the data on computers that are near clients or even clients. This replicas are called *secondary replicas*, while the replicas on the inner ring servers are called *primary replicas*. Due to the fact that some secondary replicas are nearer to a particular client, they provide a faster access. This approach is similar to caching in PAST.

The data on the clients is *soft-state*, which means that there is no guaranteed availability or durability of this data. If the storage on the clients is needed for something else, the client copies are deleted immediately.

To ensure high actuality, OceanStore uses a dissemination tree. Updates to the data is always performed at the inner ring servers. The updated data is then spread from the inner ring servers through the dissemination tree to the clients with secondary replicas.

4.2 Backup applications

Since cloud services get more and more popular, recent backup applications take advantage of this and outsource the backup data to online cloud storage [11]. Due to a monthly or annual fee for sufficient storage for online backups, on long term this gets more expensive for the customers, than buying once an additional storage drive for backups. Toka et al. [11] present concepts to delegate this tasks to peer-to-peer networks, which avoids high server maintaining costs.

But this migration to peer-to-peer networks produces further problems. Examples for the upcoming problems mentioned by Toka et al. are the unpredictable behavior of online peers, crashes and failures are highly excepted and low access latency is difficult to achieve. In the backup application scenario, Toka et al. pursue the goal to achieve short backup and restore times.

Due to this, they focus on the scheduling and redundancy problems. Scheduling includes the decisions that have to be made, on how to allocate data transfers between peers. The redundancy problem treats the question, how much of the data in the peer-to-peer system should be replicated, to guarantee complete and safe backup operations. A brief presentation of the core principles of the Toka et al. approach is given below.

4.2.1 Core principles

Toka et al. consider backup solutions, where the client choses a folder of his file system for backup. Beside on the clients machine, the backup data is also stored remotely.

A *backup object* is considered as "an opaque object [that] possibly represents an encrypted archive of changes to a set of files. [...] Each backup object with size o is split into k original fragments of a fixed size f, with $k = \frac{o}{f}$" [11]. Due to the existence of unreliable peers, there is additionally added an erasure coding, which implies the creation of *n encoded fragments* with size f. With this erasure coding, it is possible to recover the original data with only k of these n encoded fragments.

Toka et al. distinguished three different phases.

In the first phase, the backup phase, a client distributes his backup worthy data to a set of remote peers, that store the encoded data fragment. This phase accords with both, the scheduling and redundancy problem, since it must be planned on which and on how many peers the data should be stored.

The second phase is the maintenance phase. This phase includes the detection and correction of data loss, that happened due to failing peers.

The third phase mentioned, is called the restore phase. In this phase, a client contacts the remote peers to recover data that he has lost on his machine.

4.2.2 Scheduling problem

This subsection discusses the scheduling problem, which is the task of deciding which remote peers should be selected for data transfers. In this context, data transfers are used between the peers in the network. During the backup phase, the initiating client transfers his data to other remote peers for storage. In the restore phase, the data is transferred from the remote peers back to the client. The formal definition of the scheduling problem can be found in [11], since this is only a brief recap with focus on the research's outcome. The authors compare a schedule, that is based on a full information set with a priori knowledge about the online behavior of the clients, with a schedule that randomly selects online peers for transfer. Unlike expected, under certain circumstances the performance of both schedules is nearly the same. This is the case for the backup phase, if the number of nodes in the system is larger than the number of fragments to store. This comes due to the fact that a strict, optimal schedule is not needed, when enough alternate remote peers for storage transfers are available. The same performance for both schedules holds for the restoration phase, if the redundancy rate is not close to one.

4.2.3 Replication problem

Toka et al. present an approach, that dynamically changes the replication rate. The value of the replication rate is adjusted in realtime and depends on a data loss probability and the expected time on doing a restore of the data — called $eTTR$ (estimated Time To Restore). These values are calculated within the backup phase. The $eTTR$ is bounded by the download rate of the restoring peer p and the average online availability and upload rate of the remote peers, that store a backup fragment of p's data. Remember, that for a restore operation only k of the n fragments are needed, thus the data loss probability is based on the probability that more than $n-k$ peers have disk crashes at the time t. t is the time between p loses its data and initiates the restore operation plus the time to complete the restore operation. The details on calculating the data loss probability and the $eTTR$ can be found in [11]. This approach increases successfully the performance of the backup application system.

But since this approach is presented in a context of backup applications, the authors reason that prompt data availability is not a primary goal, because the peers only want to retrieve their data upon crash events, that are assumed to be rare. This does not necessarily correlate within a context of distributed online social networks, so this approach is not further discussed.

5 Conclusion

This section sketches how the challenges of data replication in our distributed online social network can be solved by taking advantage of the knowledge obtained by Section 3 and Section 4. The following briefly summarizes the challenges mentioned in Section 2.

Data replication in a distributed online social network should achieve high availability of the data, delete-ability and modifiability of the data in addition with providing security of private data. The implemented framework must consider heterogenous peer capacities.

The service provided by PAST suffices as a basis for the data replication in our distributed online social networks. The presented results in Section 3.2.4 show that a reliable data availability can be provided. Additional features like mutability of files could be achieved by modifying some of the PAST operations.

Since OceanStore is not purely peer-to-peer based and takes advantage of highly available and durable servers, this approach is no serious alternative for us. The opportunity of versioning introduced by OceanStore is not necessarily needed in a distributed online social network environment either. The results of applying replication in OceanStore, are already achieved by the caching mechanism in PAST as well.

Nevertheless, the applied method in OceanStore to access the same file object through the different versions could be used in our PAST modification, to obtain mutability of the data. The reason, why data mutability is not able with original PAST, is the creation of certificates for each file. Every file certificate contains a SHA-1 hash that takes as input the content of the file. A result of this is, if a file object is changed, also the content is changed which results in a new hash and therefore the certificates of a modified file may be rejected.

A further benefit of modifying data is, that it needs less storage space. Creating a new file instead of modifying the origin file would cause an additional file inserted besides the original file. Using the same fileId with a still valid certificate allows to simply overwrite the existing file with the modified data.

This would additionally introduce the ability of deleting files. Private data, like photos, that should not be stored anymore in the social network, could simply be *overwritten* with a block of zero bits. While the origin data – the photo - cannot be retrieved after applying this deletion, the storage space is unluckily still used and additionally the data is still replicated.

Unfortunately, this approach would introduce some more drawbacks. First of all, attackers can simply overwrite existing files by flooding the system with fake deletion commands, due to the fact that the generation of a fileId only needs the file's name and the file's owner's name, which is no private data. This problem could be solved, by concatenating the file's owner's private key instead of the public key to the input of the hash function.

Nevertheless, two drawbacks still remain. Removing the file's content as part of the certificate, removes also the ability of the receiving peers, to authenticate that the data has not been corrupted or modified during transport.

Furthermore, accepting a file with small size by a low powered peer could lead to unexpected behavior, when the file size increases rapidly during modification. A possible solution is, that a peer is allowed to reinitiate replica diversion or file diversion. But in the case of file diversion, the $k - 1$ other numerically closest nodes should be notified, so they can delete their replicas.

Since PAST uses a fixed amount k of replicated files, Toka et al. present in their research, that a dynamically change of the replication rate could add an beneficially effect to the systems performance. This should be kept in mind, if an additionally boost of the systems performance is needed. Since the assumptions made to reason this approach in a backup application do not necessarily correlate within the context of a distributed online social network, the implementation of this feature is not planned, since the beneficially effect is not guaranteed in our context.

As the results from Toka et al. show, further investigation of a concrete scheduling algorithm is not needed, since the performance of randomly scheduled peers for data transfers can compete with the results of an ideal schedule, assuming that the underlying peer-to-peer network has sufficient peers.

Another mentioned problem in PAST is the assumption of homogeneous peers. A possible negative effect on PAST's performance, brought by using a heterogenous peer-to-peer system, should be examined in practical use. A possible solution could be the individual adjusting of the file replica and file diversion thresholds. Poor powered peers may set the thresholds so, that they reject more request than powerful peers. But this could be exploited by malicious nodes, that pretend to be low performance peers.

It remain the problems of security in PAST. As mentioned by the authors, data privacy could be achieved by using an asymmetric cryptosystem to encrypt the client's data. This and other presented security problems depend on the existence of a pair of public/private keys. Since the security of the whole distributed online social network should not only rely on the data replication, this topic should be discussed in more detail within the security paper.

References

[1] FinanceTechNews: How Facebook sells your personal info — and gets away with it. http://www.financetechnews.com/how-facebook-sells-your-personal-info-and-gets-away-with-it/.

[2] FreePastry. http://www.freepastry.org/FreePastry/.

[3] msnbc: Goldman to clients: Facebook has 600 million users. http://www.msnbc.msn.com/id/40929239/ns/technology_and_science-tech_and_gadgets/.

[4] Open Compute Project. http://www.opencompute.org.

[5] The OceanStore Project. http://oceanstore.sourceforge.net/.

[6] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS)*, 1997.

[7] P. Druschel and A. I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HotOS: Workshop on Hot Topics in Operating Systems (HotOS)*. IEEE Computer Society, 2001.

[8] D. Geels. Data Replication in OceanStore. Technical report, EECS Department, University of California, Berkeley, 2002.

[9] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Systems Platforms (DSP)*. Springer, 2001.

[10] A. I. T. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *SOSP*, 2001.

[11] L. Toka, M. Dell'Amico, and P. Michiardi. On Scheduling and Redundancy for P2P Backup. *CoRR*, abs/1009.1344, 2010.

Distributed Data Structures for Online Social Networks

Daniel Breitlauch
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: dodegy@upb.de

Abstract

At the moment two trends are becoming apparent. First, online social networks are getting ubiquitous and enjoy a steady growing user base; secondly, distributed peer-to-peer (also p2p) systems becoming widely accepted and more and more used besides file sharing. We would like to combine these two trends and build a distributed social network directly in a peer-to-peer system. So primarily this paper centres around requirements needed in p2p overlays in order to develop a distributed social network on top. We discuss possible solutions to extend existing peer-to-peer systems so that they provide a persistent distributed data structure which incorporates linked lists and trees just like local data structures. A key component feature not present down to the present day is the deletion of content. This will be discussed and some solutions presented. We also address the need for non-probabilistic searches that yield in certainty. All these methods are discussed as extensions to existing distributed hash tables which commonly only support exact-match searches. Lastly we discuss security and access control schemes which extend to the whole range of topics. Thereby we present some insides on attack scenarios and problems when dealing with this kind of data structures.

1 Introduction

Although distributed data structures are not new, realising an online social network with them, puts a few new challenges on top of these established data structures. Also the apparent and growing need for feature rich and user friendly online social networks drives this development. Known social networks like Facebook or StudiVZ are built and realised with large server farms in the cloud. This provides them with tight control, but carries the burden to finance this architecture. After all, a single company provided with this vast control and massive power over all the personal information left in this network is greatly discouraged by more and more users. Consequentially more and more ideas arise to overcome this architecture.

Fortunately there are already other architectures which support heavy peer-to-peer systems that scale well and do not need large server farms. In addition these systems provide highly decentralised control, hence nobody can misuse or sell your personal and private data. But these mostly file sharing systems are not designed to support such things as profiles, friend lists, photo albums or even video massaging. Even persistent storage and non-probabilistic retrievability of stored data are big problems and highly discussed at the present state of development.

Out of this situation and the promising features of such distributed peer-to-peer systems we are highly encouraged to add these social functions to existing systems. This paper presents state of the art methods to alter given peer-to-peer systems which enables them to store linked data such as trees and lists persistently, therefore providing the basis for a distributed social network with distributed control mechanisms and nearly all features known from existing centralised solutions.

The rest of this paper is organised as follows. First there is a brief introduction on existing p2p overlays in Section 2, followed by a compilation of requirements a social network application would pose for p2p overlays in Section 3. Subsequently in Section 4 the main data structures get defined and search methods elucidated which provide the user with a wide range of socially desired possibility to connect to other participants. Lastly Section 5 is about security aspects and finally the conclusion of this work.

2 Distributed Hash Table and P2P Overlay

This section provides some insights in the possibilities to decentralise a social network. Such an application has to handle large amounts of user generated data. Commonly it stores this data at large servers in the cloud. But it is also possible to store this data in one distributed hash table (DHT). This data structure uses a hash function just like a local hash table. The big difference consists in storing the key-value pairs not in local bucket, but using the bucket identifier as an address. Finally these addresses represent all nodes in a network. So all the key-value pairs of this hash table get stored decentral in this network of nodes. The big advantage in this will be the immense space and collaborative bandwidth all nodes provide. But hence there is no direct connection to the underlaying Internet, a routing scheme is needed. This routing-overlay over the aforementioned network of nodes is called peer-to-peer overlay.

2.1 Data Responsibility

Due to constant joining and leaving in the network (named churn), there is not always a node online with the exact nodeID. Namely also the large object space and other things come into play with this. So to which node should a key or objectID be moved? To answer this question a p2p overlay choses the node with smallest distance between objectID and nodeID. This distance measure is singly constructed from both IDs. Therefore no geographical information is used and further nodeID and objectID have to be in the same space. Due to churn the stored data is also not very persistent, but there are schemes discussed in Section 3.2 to make the data persistent.
Also worth to mention is the data distribution. In order to be a fair network nobody should be overloaded. It is equally important to minimise daytime and location dependencies to avoid off-time of all neighbours at the same time. These problems are addressed by the hash function commonly the SHA1 function which outputs an uniform distribution. To further minimise unequal treatment, in Section 3.3.1 methods get discussed to balance request loads which is seen over long time non-uniform.

2.2 Routing Schemes and Topology

As mentioned in the section above, this kind of network needs its own routing system. Further this routing is based on the hash key generated for an object. So there are a few different routing schemes to get messages from one node to another.

Chord Chord uses the absolute value as a distance measure whereby the highest nodeID back circles to the lowest. To route some message through the network a node uses a successor link in the ring space and more important $O(log(n))$ shortcuts where n is the number of nodes in the network. Each shortcut in one node doubles the distance of the predecessor shortcut and the first shortcut is the successor link. This way any message only needs $O(log(n))$ hops in the overlay network to reach any node. There are also only log(n) routing entries needed in every node.

Pastry Pastry in contrast uses more table entries in the nodes to construct a better routing table. But first things first, Pastry's distance measure consists of prefix similarity of both IDs. The routing table consists of links to nodes with the same prefix and to nodes with a shorter equal prefix. However, the important improvement in Pastry comes with the leaf nodes and neighbours. The leaf node set comprises a number of nodes which are nearest in the objectID space. And the neighbours set comprises nodes which are geographical or by means of IP the nearest. With this sets routing gets improved through the more advanced choices to route via low latency neighbours. The leaf set will be of great help to implement a replication scheme.

2.3 Interface for Higher Levels

All other functionality of a peer-to-peer social network application would be based on the p2p overlay and specifically the DHT. Hence all the needed data structures will be using this interface to relay messages, stare, retrieve or search data objects. Therefore and to support replaceability of the p2p overlay part, we have to provide a general way to handle the various kinds of interaction.
The next section will contain a list of requirements the p2p overlay should implement. These operations are a bare minimum to support all application relevant tasks. Section 4.2 will propose some additional structures direct above the p2p routing to support a rich search mechanism. Also the DHT could provide an updated list of the aforementioned leaf nodes to simplify replication mechanisms drastically but full particulars in Section 3.2.

3 Requirements for an Online Social Network

In this section requirements on p2p overlay networks will be presented they have to fulfil in order to be useful in an online social network implementation. These requirements are posed by data structures which are necessary for a distributed social

network and will be considered in detail in Section 4. So all these preconditions should be complied and implemented directly on top of the distributed hash table.

3.1 Functional Requirements

These requirements rise naturally from the needs of overlay data structures. These needs are broken down to their most fundamental level such that a simple and elegant interface could be constructed. This way the DHT and perhaps more components could be exchanged without much hassle. So all of these operations work directly on the DHT.

Write and Read Data In order to write or read data there is not much to do. The standard DHT interface supports this kind of action already. First an objectID gets generated from the data. This happens normally by means of hashing some attributes or the whole data. With help of this objectID the data gets stored in the distributed hash table under the key-value pair (objectID, data). To retrieve this data the DHT needs only the objectID, just like a normal local hash table. The DHT performs a lookup operation in the p2p overlay and a responsible node sends the data. Only shortcoming is the construction of the objectID. Either you know the objectID from other data or you need a mechanism to generate it from the data or attributes. In order to keep data available even in the case of churn, the following two mechanisms are helpful. Storage replication is the active duplication of objects on various peers which are unlikely to fail all at the same time. Caching is the passive temporary spreading of popular content in order to have more peers that can answer to popular queries.

Update Data Updating Data is the first challenging operation. Not so much due to the DHT which can update key-value pairs (objectID, data) quite easy, but because of other factors like storage replication and caching. If a caching scheme is in place other nodes not responsible for the objectID might have the corresponding data. In that case it is advisable to construct a cache message through appending a date of expiry if updating of data is needed. In case of an online social network there is no easy way around this while caching enabled. The other problem mentioned, namely replication, is easier to handle. Here the update operation has to work on all copies or replicas of the data. In most DHTs this is realised through a responsible node nearest at the objectID position which manages all replica updates.

Append Data Appending data should be canonical after updating is implemented. But there is the need for updating only half of the data and the other half should be denied to alter. This problem arises from a needed data structure in Section 4. There we propose a linked list schema to store information. Imagine an inbox data structure containing a list of objectIDs to unseen messages. Another node should be able to add its created message objectID to the list, but should be denied to alter the existing data. Some details will be addressed in Section 5 and the best way would be a rigorous encryption and signing system, but this would go beyond the scope of this work.

Delete Data Considering an online social network, the deletion of the users own data should be allowed, supported and common sense despite some existing contrary examples like Facebook. But also in most DHT p2p overlays there is no mechanism to delete data. Therefore we have to implement some deletion method, possibly by updating the data with an empty document. This can signal the receiving node to simply delete the key-value pair. This message should be propagated to all replica nodes.
One additional use case emerges from the fact that we are dealing with a distributed online data structure possibly with encrypted user data. So there need to be privacy for all users by all means, but also a scheme to delete illegal content. In this scenario it is hard to delete illegal content. This particular question needs to be answered along with the already identified access control problems.

Message Communications There is a distinct use case to communicate directly with other participating nodes besides the use of the distributed hash table. This use case originates from plugins which provide some functionality like chat or video conferencing. Because this communication does not have to be persistent, but the receiver is identified by its nodeID. Thus, the DHT must route the messages to the nodeID but must not store the data, only delivery is important in this case. So most DHT implementations have callback functions to incorporate foreign or external behaviour. In fact they outsource already the storage routines, for this reason there should not be a problem to implement an interface for raw communication over the distributed hash table.

3.2 Redundancy through Replication

Yet former sections mentioned some redundancy and replication ideas, but in this section the topic will be presented in a broader theme. So there are some replication schemes, all with similarities and differences just like distributed hash tables. In the following list are three broader categories of replication solutions for DHTs. For more details regarding some of the shorter presented categories below consult [6].

Multi Publication Key Replication In this scheme the distributed hash table uses many arbitrary hash functions, such that with every hash function another target node gets used. These target hashes are reproducible and thereby the data is also retrievable if any of these nodes is online. Every of these nodes also has to know the status of all other-hashed nodeIDs for every key-value pair it holds, that it could replicate the data to a neighbour of the just gone node. This data overhead could be reduced, but it is more reasonable to use already known nodes to replicate.

Path Replication Path replication tries to use redundancy in a kind like caching. Though caching operates after a lookup occurred and the data gets cached all the way on all nodes from the responsible node to the recipient. Path replication happens while sending the data and results in data stored along the way to the responsible peer. Downside to this approach is the bad reproducibility where the copies are and following a poor to impossible situation to delete or update copies.

Neighbour Replication Remember, a key-value pair (objectID, data) gets stored on that node with the smallest difference between objectID and nodeID. Now, not only this node is responsible for storing but also a whole range around it is in charge. If in case of churn any stored data exists, the lookup will find the responsible node automatically. With similar objectID comes cooperative responsibility. This is easy to accomplish, hence Pastry already stores and updates a set of leafs or neighbours in objectID space. In contrast Chord nodes save only one successor node and therefore perform a kind of successor chain replication. Dependant on the replication count these chain of consecutive nodes is responsible for replicating all of the respective first nodes data, but churn would be more problematic.

3.2.1 Neighbour Replication and PAST in Detail

So the most promising replication scheme seems to be neighbour replication. Here we use neighbour information which in case of PAST we already have through the DHT management, thus no additional work to update this information. But some factors have to be addressed. These update mechanism through the DHT leaf set is only possible if the replication count is smaller than the leaf set. Now, before a list of relevant situations gets presented, it is important to discuss this one most relevant parameter in replication. The replication count determines how many copies have to be fabricated. This is highly dependant on churn and data or chunk sizes. To go in more detail large amount of churn means many of the below mentioned events and following many synchronisations and copies. These copies need time. If the amount of time needed to synchronise a leaving nodes data to a new one exceeds the average online time of nodes, that could be fatal. Therefore if data and chunk sizes are large, replication count must also be high, following the promised relevant situations.

Leave/Fault Behaviour First of all there is no big difference in handling both of these events since we have to always prepare for faulting. This is required in order to achieve robustness. So leaving is simply a nicer faulting with a 'good bye'. Fortunately our cooperative responsibility spans only in the range of leafs the distributed hash table in Pastry already monitors. Pastry nodes do this by pinging their leaf nodes regularly and register their online status. This way a node can observe replication partner nodes leaving and faulting. In which case the node starts to replicate its data to the nearest node not replicating its data prior to that. Only problem left for a node is to detect what is its data. This means the data it is responsible for and not the replicated data from its neighbours. This is also solved by recognising another node in the leaf set which might be nearer by some objectIDs. And since the nearest node is responsible every node knows by monitoring leafs when it gets responsible for data.

Join Behaviour Joins are handled relatively easy. The previously offline node synchronises all its data to the state of its nearest neighbours. This includes finding objectIDs from neighbours it should own and deleting/updating old data not found on any neighbouring nodes. Also through nodes joining, other nodes have to lose their responsibility on some objectIDs. So here comes the same mechanism as above into play.

Worth to Mention Scenario This special case lays more attention to the shift of responsibility which is not covered solely by the sum of all events above. In this scenario five nodes are responsible to store a key-value pair. These nodes are normally quite distant, but this time no other node lies between them. Therefore they are responsible and all things are legitimate. Now many nodes join the network and are placed between them. Some of the replica storing nodes are drifting away and if another node sends at this point a delete message then the responsible node would delete the data in all nodes in replication range. But it could not reach out to the now far away emigrated nodes. Therefore while nodes join, not only they have to erase some data and get up to date but also ousted nodes have to delete replicas just because they are poorly reachable and finally no longer in replication range.

3.3 Non-functional Requirements

Besides these important functional and technical requirements there are also non-functional preconditions which need to be addressed in order to present an acceptable user experience. Most of these problems, if not tackled, tend to result in unpleased users which consider not to use the social network. There are many of such non-functional requirements, but the most important ones get described in the following list.

- **Performance** has to be addressed. Given the application gets its data over the Internet, high connection latency leads to GUI latency which results in waiting and resigning users. Moreover the latency depends greatly on the hop count that enlarge the level of indirection to travel.
- **Costs** are good indicators of useable p2p overlays. Whereby cost means how much traffic is generated. This is important because the user has the burden to pay for the bandwidth, hence stopping to use expensive applications.
- **Retrievability** should be a given fact for a social network. So every information placed in the network has to be retrievable for certain. This means probabilistic searches are not feasible. More details how to implement searches are given in Section 4.2.
- **Fairness and Heterogeneity** are by far the most relevant. If some participating nodes have to act like high end servers as in a server-client architecture, the whole concept of peer-to-peer architectures gets useless. So load balancing is needed where it is appropriate and load unbalancing is needed for weaker participants.

Adding those solutions at a later timeframe could result in an unfitting amalgamated design or even unforeseen, hard to find bugs. Minimum countermeasures would consist in a layered, plugin based design with clearly defined interfaces. But almost easier is to account them early and implement them right away. In literature most of the problems get discussed and effective solutions are presented. Hence following a short description of the most suitable of these solutions.

3.3.1 Fairness through Load Balancing

First of all it is appropriate to differentiate between routing load and request load. There are different algorithms in place to balance each of this, but many of these algorithms profit on a synergetic effect between them. Thus, they are independent but work best together. Most practical in p2p networks is a local decision algorithm which lets the desired property globally emerge. To fulfil these balancing acts the respective node needs to gather some information upon which it can decide. So basically all algorithms incorporate a form of monitoring resource requests to identify heavy loaded nodes.

Virtual Servers This method is proposed in [9] and utilises the fact that load depends on the nodes position in objectID space. For this method a node is called virtual server and many of these virtual servers are combined on one machine. These combination of servers is then called a node. So one node has no responsibilities at all, only its virtual servers manage a range in the objectID space. Load in contrast is bound on the node, actually the sum of all loads on the virtual servers. Now the idea consists in moving this virtual servers around to other nodes such that all nodes have equal load. [9] describes three different rulesets to move these virtual servers between nodes and an overall balancing within 80 % of optimum balance is achieved. However this method has also some drawbacks, namely it is not suitable for Zipf-like or exponential distributed requests [4]. It is only feasible for distributions like Gaussian, because exponential distributions can not be balanced through shifting of virtual servers. Only if dividing virtual servers is performed exponential distributions could be handled, but this introduces the problem of objectID space fragmentation. Increasing virtual server count generally eases balancing, but the increase in virtual servers would also complicate routing through the p2p overlay. So virtual server granularity and overlay routing ease are conflictive and considered a trade-off this method has to face.

Routing Load Balancing Another approach arises from the idea to alter the routing table. This is fully orthogonal and independent to the aforementioned virtual server method, so both could theoretically be implemented. Routing table updating (RTU) is introduced in [4] and copes primarily with the impracticality of virtual servers at Zipf-like load distributions. First step in this method would be the identification of heavy loaded nodes. This happens through monitoring all the resource requests one node receives. Then the node where the RTU algorithm is executed goes through its routing table and checks if these nodes load is larger than the average local load. If there is a node identified which gets more and more requests then the original node asks this heavy loaded node to provide a light used node with the same prefix as its. And since RTU is based on the Pastry overlay network, thus having the leaf node set and locality information about neighbour nodes at hand, the heavy used node can provide a light used node from its leaf node set. Now the original node removes the heavy loaded node from its routing table and routes through the new light used node instead. This simple procedure is executed on all nodes in fixed intervals. Also worth to mention, this method works great together with hot resources caching [4] explained in the following paragraph.

Request Load Balancing There are few methods to counteract non-uniform request distributions. One of the most used methods consists of caching heavily requested documents on other nodes. Like common sense dictates these documents should be cached on nodes which lay on their ways from the responsible node to the recipients. Happily resource lookups converge early in Pastry. So there frequently exists a node where two lookups meet or converge to get to a common requested resource. An early convergence means this node is often closer to both of the recipients as to the actually responsible node for the requested objectID. Thus there are many nodes which through caching can help reduce load of the original node or incorporating the use of replication also the original k-replicas.
One method also described in [4] will be discussed more precisely. In detail hot resources caching (HRC) consists of a ruleset executed locally without global knowledge of any kind. So now and then every node checks whether they own a resource that is heavy requested. To minimise this load, the node identifies strong demand carriers and sends the heavy requested resource to one of them. Considered this algorithm was designed with Pastry in mind the chosen node would be the one with lowest latency. As a result the requested document gets cached closer and closer to the requesting nodes thus combined with RTU all the nodes get equal lookups and can provide the data thanks to HRC. Details on how to identify heavy requested resources in changing demand patterns is left open in this paper, while several possible methods to tackle this problem are found and further elucidated in literature. In [4] they use a simple approach where they count all requests and associated recipients in different time intervals therefore recognising recent overly used documents. Figure 1 illustrates an experiment, how well RTU and HRC balance Zipf distributed requests. Details of the experiment are found in [4].
With this algorithm in place it is crucial to mention unlike replicated documents, cached resources have to be deleted after some time. This timing should greatly depend on user expectations, which duration they tolerate for updates to happen. Hence the aforementioned cache message should include a date of expiry.

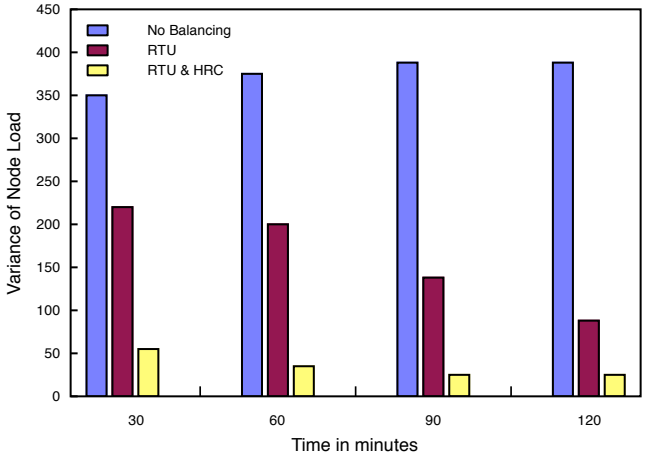


Figure 1. Node Load Variance

3.3.2 Heterogeneity

There are also certain situations where a particular load imbalance is appropriate. These needed imbalances are introduced by the whole range of possible bandwidth of attending nodes. A mobile version of the online social network application would rarely participate in the data exchange and therefore it should not be responsible for a large range of objectIDs. On the other hand there could be high bandwidth servers possibly attending for altruistic reasons. So these nodes should be enabled to lift heavy weights and be responsible for more objectIDs or their replica.
First of all mobile or low bandwidth devices could send a specific token when joining, so that they do not get involved in routing or lookup requests while being considered for join and leave messages. So they can maintain their neighbours, leaf and routing tables whilst not being bothered with the rest. Secondly, the aforesaid method (Section 3.3.1) introduced to balance intense used nodes could also account for heterogeneity. Every node, while joining, could send a weight or target load which gets considered in the load calculation to find heavy loaded nodes. This way every node can contribute adequately. Mind that these weights should only be allowed to increase the load compared to normal load. So either you contribute nothing, normal load or a multiple of that. With some regulations on these mobile tokens this should account for some misuse or rip off. Those mobile tokens could be bound to actual work providing nodes such that everybody contribute sometimes.

4 Structuring the Data

This section is dedicated to the main idea how to connect distributed data to structures. Most of the technics are based on local data structures which then are widened to distributed scenarios. In the following Section 4.1 is a first but most important example, how to overcome locality. In Section 4.2 will be explored how to offer powerful search mechanisms on top of a DHT.

4.1 Providing Linkage

The general idea is to use the unique objectID as pointer [2], this eases the handling of the persistent data and provides a clean abstraction of the p2p overlay. This way we can use most simple local data structures like linked lists and trees also distributed across a whole p2p network. This object oriented distributed scheme could be used to store and retrieve whole graphs connected with this objectID pointers. Retrieval of objects on the fly would not be possible hence the identifying objectID is not known. But due to the graph structure it is possible to retrieve some objects with the help of known data like own friend list. Now the application could brachiate from graph node (objectID) to graph node. All the data has to be arranged in this directed graph and entry points are given by reconstructing some of the objectIDs because they incorporate a given structure.
Also, there is the possibility to cluster some objects like videos in smaller chunks and give every chunk an objectID. Now we need only one object which is a list of all the chunk objectIDs and use this list. This has the advantage to distribute popular items better in the network, but has one big drawback, if one of the chunks is missing or offline the whole file could not be assembled and used.

4.2 Providing Search

Until now finding data consists of constructing the objectID through hashing of the data we know and would like to find. Thereby we needed to know how to compose the objectID. And since we use the userID (public key) to construct the objectID, we can not search easy for names, we have to know the userID first. So either we have a pointer to some data or we have an userID or objectID and can traverse this data until we come to the needed data. But we also want a third option to get an objectID or userID with only some minor information about the data. Imagine to use the social network application and having to know exact names of friends or groups to get in contact.
There are these following listed use cases a social network application that has to be address in order to find data. The expressiveness of such searches related to the implicated implementation costs are well discussed in [1].

- **Keyword** searches needed for tagging or other mechanisms.
- **Substrings** needed to find user names and groups.
- **Integer range** searches are needed in order to find profiles with the help of age specifications.
- **Multiple attributes** searches are handy to find a profile which meets more than one criteria.
- **Non-probabilistic search methods** because such methods like random walks, flooding or rendezvous point are not feasible because user expect that data they insert into the system is either there or lost.

Keyword search Keyword searches on top of DHT are simple. There are two methods to get keyword search implemented. The first and easy mechanism was often mentioned and is realised by a known scheme to incorporate the keyword within the objectID such that the objectID could be reconstructed. The more involving method consists in making a list of objectIDs of items that belong to the keyword and storing the list under a reconstructable objectID.

Substring search Substring searches are implementable through range searches, but this method is easier and also leading to the desired result. This method is based on textual similarity searches from computer linguistics and uses also the aforementioned keyword search. To start and make objectIDs recognisable under certain substrings all searchable words get divided in 2- and 3-grams and can be used as keywords. To search for results with only a substring we have to divide this substring in 2 and 3-grams, too. Then the algorithm performs a keyword search for this grams and counts all results for the linked objectIDs. Then immediately after sorting we get the most promising objectIDs.
By the way, all objectIDs satisfying Equation 1 should be dropped, due to non existence of this substring in the word.

$$hits_{objectID} < \frac{|substring|}{3} \tag{1}$$

where: $|a|$ denotes the character count of the string a.

Range search Range searches are widely studied in literature. In most cases ([7],[5]) an additional data structure over the DHT is needed. These data structures work parallel to the p2p routing overlay. Such additional data structures are skip graphs which work just like there local pendant skip lists, prefix hash trees (PHT) and range search trees. Range search trees are discussed in [7]. This method is far to invasive for the DHT, thereby not applicable to all DHTs and thus not useful as a search method in the near future.
On the other side are prefix hash trees presented in [8] which are constructed upon the p2p overlay lookup-function solely. This DHT-extension works for one dimensional range searches and "updates are doubly logarithmic in the size of the domain being indexed and resilient (the failure of any given node in the Prefix Hash Tree does not affect the availability of data stored at other nodes)"[8]. Hence the updates are not dependent on the actual data size only on the domain size, some really fast range searches are possible for small domains like age information. Another big advantage is the strict overlay architecture which leaves the DHT alone.
The exact workings of the PHT would blow this paper out of proportion, so it is compacted to a moderate description. A PHT builds a binary trie upon the data set. Each node of this trie has a label. This label is recursively defined and could be visualised by walking from the root to the node every direction decision adds a 0 or 1 respectively. The PHT has a set of leaf nodes (which should not be confused with Pastry's leaf set) which contain all the values that have the same prefix as the corresponding node. This way a range search first searches both border nodes of the interval through walking down from root and choosing the direction in the binary trie based on the search value. Second the search algorithm traverses only the leaf nodes and collects the values and paired objectIDs.

4.3 Further Considerations and Improvements

This is a short section about some minor considerations and improvements.

- Minimising the workload while joining is possible with just a minor constraint. It is expected that users will often disconnect and reconnect thus generating massive churn. If one system has always the same nodeID or in case many people are using the same application has the same nodeIDs, then the amount of updates to handle replication and new data would be minimal.

- There are a few cases where this distributed approach has its problems. For example when it should not be possible to do two things parallel because this interaction would cause irresolvable trouble. So it should not be possible for two users to have the same userID. While registering for an account this method has to add an user atomically. Also this is the case for all other creation routines like group or keyword creation, especially adding a message to an inbox linked list.

5 Security

As nice and essential the above mentioned data structures are, they also open the door to potential threats. In this section a list of threats gets presented which are solely possible through this data structures. The following list is organised by security principle which gets infracted through the attacker.

- **Integrity and Availability** The basic assumption in a distributed environment is aptly summed by ”Do not trust anyone”. In case we have not established a rigorous key and signing scheme, this holds also for the nodes in the p2p system. Especially do not put trust or presumptions in the implementation of these nodes. The software do not have to be the application in mind. In this case the portrait data structure which enabled linked list like behaviour is prone to some attack. If an attacker node gets asked for some data which contains objectIDs to other data, it could replace the ID and would be answering with a malicious document. This scheme could be used to generate many requests on one node and results in a denial of service.
  Another problem arises from a design aspect that the nodeID is the userID. In that case the joining node could decide the userID and would in any case get the desired responsible range in Pastry. Luckily there has to be a countermeasure in any case, because posing under another userID should not be possible.

- **Privacy** With the distributed hash table mapping objects always to the same ID and userIDs always linked to one node, this imposes a privacy threat, hence the user is bijective identified with a node. All requests could be monitored and archived, the attacker node only has to be in any routing table on the requests way.

- **Fault Tolerance** Though the p2p overlay routing architecture is the important factor to achieve robustness, it is most advisable to use an overlay which has many routing ways and reacts to fault of nodes. For example Chord in its simple implementation do not use very good countermeasures to keep the routing stable. On the other hand stands Pastry which uses its additional routing information often mentioned to route around dead or unresponsive nodes.
  A clearly distinct threat to the system arises through a kind load balancing algorithm. It is an open question and debatable whether a node is allowed to not participate in routing, storing or replication. This is highly exploitable by malicious nodes which mark as overloaded and gets as few load as they want. This way the whole network could collapse.

- **Access Control** This is probably the most important part of security. Every user wants to define access rights for his buddies, workmates or fans such that he can decide fine-granular whom he allows to visit his photo albums, friend lists or evening activities.

Solution Despite the mentioned obstacles to secure such a distributed application, do not be disheartened. The solution for most of these problems are a rigorous encryption and signing scheme proposed in [3][2]. There also has to be a key management system. Firstly, to have a working signing system you need to have the possibility to retrieve the public key of an user. Thereby it is best practice to use the public key as userID this way there is no man in the middle attack while retrieving it. Secondly, to have a working encryption system the user needs a private key at hand. Naturally the private key can not be stored in the network, so the application has to generate the private key from an email or user name and password. Build upon this it is possible to have an access control system with signed keys for other people to decrypt objects they are allowed to see. This way a malicious node can not hand out false data because checking sign of the owner would report an error.
This security measures imposes an overhead in every message and calculation overhead on every data preparation, but it is necessary and unavoidable in order to provide these functions which are expected by most users.

6 Conclusion

This work has shown more non-functional requirements than functional. This might be awkward but is intended because user experience is far more important in a social network application than in a distributed filesystem or p2p file sharing application. Thus, so many ”soft” requirements.
In process of this work it became apparent that it is best to use a preexisting p2p overlay. Most required functionalities are provided or easy extendable. Most implementation work would be to provide caching and search algorithms, but against this background it is inevitable because such a social network should work continuously and balancing is a key component to provide this. In order to implement caching the p2p overlay should provide a method to copy data to nodes with an expiration data, but this is a minimal change and is doable in Pastry.

References

[1] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *Database Theory ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*. Springer, 2002.

[2] K. Graffi, S. Podrajanski, P. Mukherjee, A. Kovacevic, and R. Steinmetz. A distributed platform for multimedia communities. *Multimedia, International Symposium on*, 0, 2008.

[3] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Cryptree: A folder tree structure for cryptographic file systems. *Reliable Distributed Systems, IEEE Symposium on*, 0, 2006.

[4] Y. Gu, L. Chen, and K.-M. Tang. A load balancing method under zipf-like requests distribution in dht-based p2p network systems. In *Web Information Systems and Mining, International Conference on*, 2009.

[5] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*. Springer, 2002.

[6] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance evaluation of replication strategies in dhts under churn. In *Proceedings of the international conference on Mobile and ubiquitous multimedia*. ACM, 2007.

[7] H. Nazerzadeh and M. Ghodsi. Raq: A range-queriable distributed data structure. In *SOFSEM 2005: Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*. Springer, 2005.

[8] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: prefix hash tree. In *PODC*, 2004.

[9] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*. Springer, 2003.

Essential Security Aspects for Distributed Online Social Networks

Jonathan Schlüßler
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: sjonny@mail.upb.de

Abstract

Online social networks are an integral part of the Internet. Existing online social networks have a major problem with handling privacy and security concerns, which derive from their centralized structure. This paper examines a distributed approach for online social networks and the affiliated privacy and security issues. For this several privacy and security requirements are identified and approaches to realize these requirements in a distributed online social network are pointed out. In addition, several attacks, including the Sybil and Eclipse attacks, are highlighted and existing countermeasures are identified. The introduced approaches and countermeasures are also evaluated for the practicability to be used in a real distributed online social network.

1 Introduction

Online social networks (OSNs), such as Facebook, Baidu or LinkedIn, became very popular in the last years and are among the most visited sites in the Internet [1]. In addition to this, individual OSNs are valued in the range of tens of billions dollars [3, 4]. Their purpose is to allow users to publish personal information, link it with friends, share different kind of data and communicate with other users. One of the biggest concerns with using these networks is the way private data is treated. All common OSNs are centralized services, thus the operating company has all information of the users. This is a big security issue in several ways: 1) the company has some limited rights to use the data in ways not all users agree with, 2) the company may illegally misuse the data, and 3) attackers might get access to the data by exploiting bugs in the used system. In fact there are many people who refuse to use a social network because of these security and privacy concerns. As a solution to this some distributed approaches appeared, e.g. Diaspora and LifeSocial. Especially Diaspora gained a lot of publicity by fundraising more than $200,000 as start-up money [2].

This paper examines the security and privacy issues that arise with a distributed OSN. Some issues which are not possible to be solved with a centralized approach can be solved with a distributed approach. However, many other issues are more difficult to solve in a distributed approach, which does not have any centralized authorization entity. For adversaries it might also be easier to attack a distributed system than a centralized. There is a number of severe attacks which are only applicable to distributed systems. These attacks are also addressed in this paper as well as existing countermeasures.

The structure of this paper is as follows: In Section 2 some necessary background knowledge on OSNs and peer-to-peer overlays is given. This section also reveals the security and privacy requirements users have when using an OSN. Section 3 examines how interactions with the system and access control on the existing data can be performed in a secure way. The trade-off between privacy and censorship is also addressed. The next section (Section 4) lists existing attacks on distributed systems and offers several approaches to reduce the impact of the attacks. The applicability of the examined approaches to achieve a secure and private distributed OSN and the defenses against existing attacks is discussed in Section 5.

2 Fundamentals

This section explains the basics of structured peer-to-peer overlays which are needed to understand the examination of security aspects in distributed OSNs. In addition to this, the security and privacy requirements of users who use these networks are discussed and the motivation for attacks is pointed out.

2.1 Peer-to-Peer Overlays

A distributed online social network uses a peer-to-peer overlay to organize the distributed data. This peer-to-peer system allows to store and look-up data using a key. One popular method to organize a peer-to-peer system is to use a distributed hash table (DHT). Although the security threads addressed in Section 3 and 4 are not based on a specific peer-to-peer system implementation, we assume a DHT-based system is used. Popular peer-to-peer overlays that use the DHT-based approach are Pastry [17] and Chord [20].

A DHT uses a predefined-defined ID space for all nodes, typical sizes for this ID space are 2^{128} or 2^{160}. All data items have an assigned node-ID from the ID space at which they are stored. The mapping from data to an ID is done using a hash function. Participating nodes also have an assigned IDs from the same ID space. The ID space can be seen as a circular structure in which the successor of the highest ID is the lowest ID, i.e. 0. This results in a ring with active and passive nodes, where the active nodes are participating peers and the passive nodes represent data items. An active node is responsible for a passive node if it is the *closest* active node to this passive node. Depending on the used DHT implementation the *closest* node can be defined differently. Pastry, for example, defines the closest node as the highest node-ID of all active nodes that is still smaller than the node-ID of the passive node. Every active node has a routing table with routing entries to other active nodes, these pointers to other nodes are also called fingers. The routing table can be constructed such that it is always possible to find a node which is closer to another passive node. Eventually, this routing process will approach a responsible node for the requested passive node. This allows the DHT routing protocol to find a route to any node-ID, independent of the node-ID of the requesting node.

Figure 1 shows a simplified DHT which uses an ID space with 32 (2^5) IDs. Active nodes are marked in gray, all other nodes are passive nodes or might not even have any data mapped to it. Node 29 has a total of 3 routing table entries. The look-up request from node 29 for node 9 is depicted in the figure. The request is first sent to the closest node to node 9 in the routing table of node 29, which is node 17. It is then forwarded to node 11 and finally to node 8. Node 8 is responsible for nodes 9 and 10, thus it is able to reply the request.

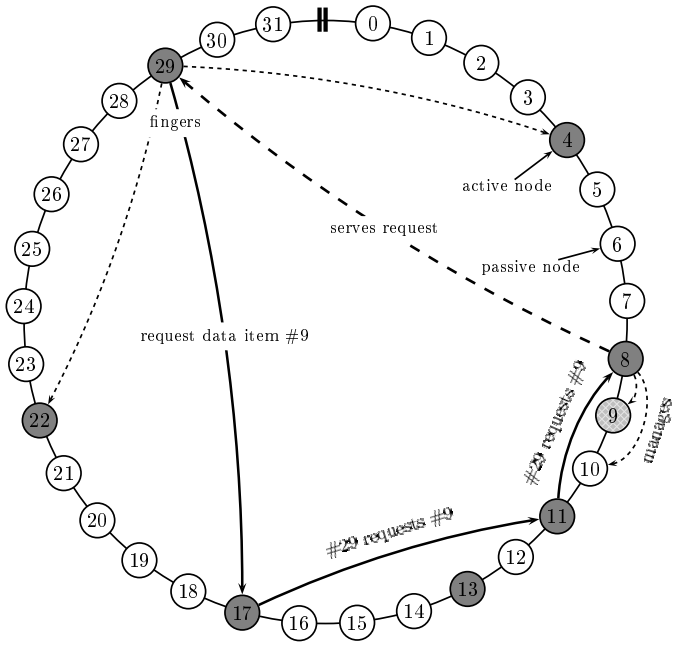


Figure 1. Basic structure and functionality of a DHT.

2.2 Online Social Networks

Online Social Networks (OSNs) are used by its users to present themselves, either to everyone in the Internet or only to a selected set of authorized users, and to form relationships to other users. An OSN offers its users a wide set of functionality

which is usually implemented in the form of plugins—some are mandatory, others are optional. Some typical functionality of popular plugins are to manage the posts on the user wall, to collect uploaded photographs in albums, or to offer the possibility of playing games. The users of an OSN are interested in having a broad scope of functionality available with a good performance.

Distributed OSN aim for the same functionality using a distributed approach. One of the advantage of a distributed OSN is that it offers better options for security and privacy. The security and privacy requirements are as follows:

Login Before a user interconnects with an OSN he has to login using his login name and a password—the credentials. The login should be able to be performed independently of the location of the user.

Access control Users want to specify who is allowed to see which content of them. It is desirable to offer a fine-grained access control. The user can specify for every item which individual users and which groups of users are allowed to access the item. Non-privileged users should also not be able to gather information about connections between data items and users. In a centralized OSN the operator theoretically has access to all user data. The advantage of a distributed approach is that there are no privileged nodes, which have more access than others.

Secure communication The exchange of information between users of an OSN has to happen in an encrypted manner. All users who are not participating in the communication should not be able to overhear the communication. The participating users need to be able to check the origin of the message and detect forged information. If there is an end-to-end encryption in a distributed system no one else but the participating users will ever see the data unencrypted. This is not guaranteed in a centralized OSN.

Data integrity Stored data must not be editable or deletable by non-privileged users. If an adversary node falsely processes a request there has to be a way to retrieve the data from another node using a replicated version of the data.

Capability of resistance The system should withstand the presence of reasonable amounts of adversary nodes. This is achieved if the system is still operable, possibly by slightly lowering the performance.

2.3 Goals of Attacks

Distributed OSNs are much more open for attacks than centralized OSNs because every active node in the peer-to-peer system has the same rights and there is no central authority which could ban adversary nodes from the system. There are different goals which are pursued by the attackers.

Disrupt service The attackers might just generally degrade the correctness and performance of the OSN. The motivation for this kind of attacks might be vicious competitors, moral reasons, or simply boredom. In the worst case these attacks might even end in a system take-over. This happens when most nodes in the distributed OSN are adversary nodes. At some total share of adversary nodes the system will not work properly anymore.

Hide content This is a particularly frequent attack found in peer-to-peer file-sharing networks to censor specific data. This is because data cannot be removed efficiently and therefore the holder of rights might want to join the peer-to-peer system, trying to become responsible for concerned data and deny its existence. This is also applicable to OSNs, applications are manifold.

Spread wrong content This can also be used as another option to hide unwanted data. Furthermore, adversaries might want to modify arbitrary data of other users without having the permission rights.

The attacks resulting from these motivations and the privacy and security requirements are discussed in detail in the remaining part of this paper.

3 Security and Privacy Aspects

This section discusses how the security and privacy requirements identified in the last section can be achieved in a distributed OSN. The basic idea is to use an encryption scheme for almost all interaction with the network. This encryption scheme can then be used to fulfill all of the security and privacy requirements. At the end of this section the trade-off between privacy and censorship is discussed. This is particularly of interest for the deletion of illegal content.

3.1 Encryption

In an overlay network all data is accessible to everyone. To suite this to the needs of an OSN, which requires fine-grained access control, an encryption scheme is needed. This scheme requires that every data item in the network has an owner which can specify access rights for the data item. These access rights are expressed by listing the users who are allowed to interact with the data item. Often access permissions are not only given to individual users but to groups of users, e.g. friends from school. It is important that all users who do not have access rights cannot access the information in the data item.

There are different encryption schemes which can be used to achieve this. In the following the straight-forward solution of using a public key infrastructure [14] is explained as well as the more complex Gentry-Waters Broadcast Encryption [13].

Both approaches make use of symmetric and asymmetric keys. A symmetric key is a single key which is used for decryption and encryption of data. Every user with the key is able to perform these operations. An asymmetric key is a key pair of two keys, a private key (*priv*) and a public key (*pub*). The private key is only known by the user who created the key pair, whereas the public key is available to everyone. An asymmetric key pair can be used for two operations. The first operation allows for a prove of origin by signing data with the private key and then verifying the signed data with the public key. The second operation allows to send encrypted data to the user by encrypting data with the public key, the receiver can then decrypt the data with the public key. Figure 2 exemplifies how the verification and encryption is performed using an asymmetric key.

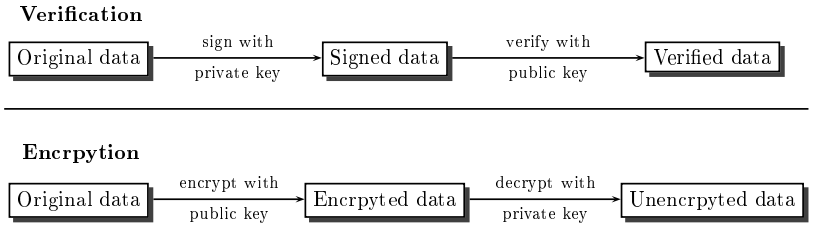


Figure 2. Verification and encryption with an asymmetric key.

3.1.1 Public Key Infrastructure

For this approach [14] every user has to have an asymmetric key pair. This key pair can be created by every user on his own. The private key has to be hidden from every one else, preferably not storing it in the peer-to-peer system at all. The public key is announced to everyone else, e.g by storing it unencrypted in the user's profile.

Every data item, which should only be allowed to be read by authorized users, is encrypted with a symmetric key. Every user who should have access to the data item needs the symmetric key, so that the user can decrypt the data item and interact with it. There are basically two options [15] for storing the encrpyted symmetric keys:

1. The symmetric key is sent to each privileged user. To hinder that the key is intercepted by unauthorized users the transfer of the key is encrypted with the public key of the receiver. If there are n authorized users this results in n encryption steps at the publishing user, n key transfers to the users and n decryption steps at the receiving users. Additionally every user has to store the key for future use. One disadvantage of this option is that there has to be a way to publish keys to offline users.

2. The symmetric key is stored with the encrypted data. This option is called Access Control List (ACL). Unauthorized users should not be able to attain the key, so it has to be encrypted with the public keys of the authorized users. All encrypted keys are put into a list. If an authorized user wants to get symmetric key, he first has to get the correct encrypted key of the list and decrpyt the symmetric key with his private key. This also results in n encryption steps at the publishing user, but the transfer of the key does not have to be performed. Instead of this, the access of the data item results in an additional transfer of the key.

If a new user is granted permission the symmetric key has to be encrypted with the public key of the new user and the result has to be either sent to the user or added to the list of encrypted keys. The case that a user gets his permission withdrawn involves much more work. A new symmetric key has to be generated because the old user could store the old key and still use it although his permissions were withdrawn. This new key has to be encrypted again with all public keys of all authorized

users. If Option 1 is used the encrypted symmetric keys have to be distributed to all authorized users, if Option 2 is used the encrpyted symmetric keys have to be stored in a list attacked to the encrypted data again.

The disadvantage of this approach is the number of operations which have to be performed whenever the user permissions change. This problem is solved with the Gentry-Waters Broadcast Encryption.

3.1.2 Gentry-Waters Broadcast Encryption

It is also possible to use a broadcast encryption approach for managing access control, such as the Gentry-Waters broadcast encryption [13]. The main advantage is that the symmetric encryption key K only depends on the set of authorized users. If a new user is added or an existing user is removed from the set of authorized users only K has to be recreated and the data has to be re-encrypted with the new K. To achieve this functionality, this approach uses four different algorithms. The first algorithm `Setup` is used to initialize an asymmetric key ($priv_A, pub_A$). The next algorithm, `KeyGen`, creates n unique keys which depend on $priv_A$. In the same way as with the previous approach these keys can be stored attached to the data, each encrypted with the public key of the authorized user. These two algorithms only have to be performed once. The next step is the `Encode` algorithm, which creates the symmetric key K using the set of all authorized users and pub_A. With the last algorithm (`Decode`) the key K can be recreated with help of the set of authorized users, pub_A, some header data which was created with the `Encode` algorithm, the requesting user and its key. The number n is the maximal number of users who can get individual access rights to the data. If the number of authorized users would exceed n the `Setup` and `KeyGen` steps would have to be performed again with an increased value for n. Figure 3 illustrates how this approach works.

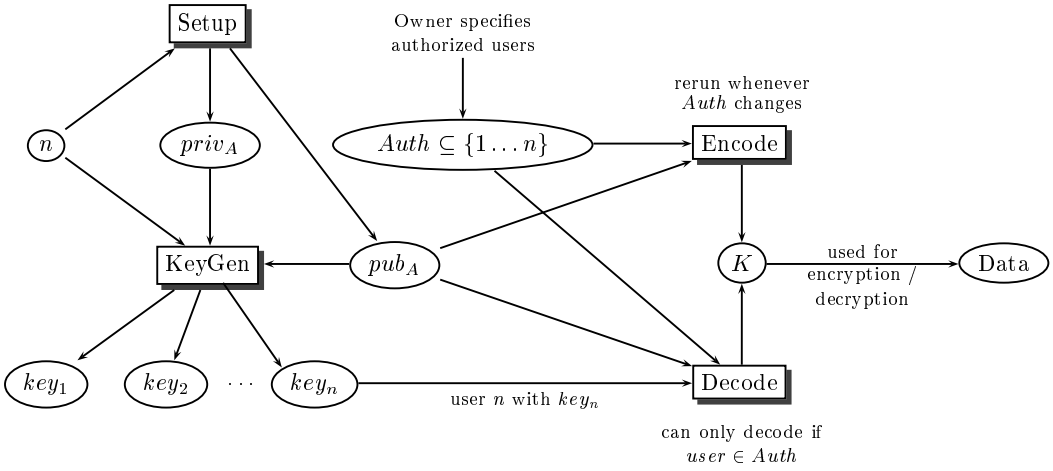


Figure 3. Creation and retrieval of the symmetric encryption key K with the Gentry-Waters Broadcast Encryption.

If a user is added or removed from the set of authorized users the only step that has to be done again is the `Encode` step. This creates a new key K which is then also used to encrypt the affiliated data anew. This only involves the recreation of one key, instead of recreating a new key for every user. The `Decode` algorithm then also works with the new set of authorized users and thus only reveals the code to the authorized users.

Gentry and Waters showed how this can be implemented in a distributed way. Although this approach sounds very promising, it has not yet been implemented for an OSN [15].

3.2 Registration and Login

The first step a user has to do before interacting with the OSN is logging into the network. The user needs to know a bootstrap node to which it can send his login request. Any online node of the overlay system can be used as a bootstrap node. The user has to provide some identification so that the bootstrap node knows which user wants to login. This identification should remain the same independent of the location, thus it has to be based on the login credentials. PeerSoN [6], for example, uses a hash of the e-mail address for this. LifeSocial [14] uses a slightly different approach: The user creates an asymmetric key and the identification in one step by using the username and the password. As we mentioned earlier, every user needs an asymmetric key to be able to perform necessary encryption steps. The public part of the key then becomes the identification

of the user. The identification is also used as the node-ID, hence directly indicating where the user-node, which stores all information about the user, is located. After the new user has been validated, it will receive additional information from the bootstrap node which will allow the new node to integrate into the overlay network. This information is encrypted with the public key of the joining node. Thus the joining node needs to know his own private key to continue with the integration into the overlay network. This prevents foreign nodes to join the network which only have the public key but not the private one.

If the user-node does not exist there are two possibilities:

1. The user might have entered a wrong combination of username and password. This results in a different public key which references a wrong and possibly non-existing node-ID in the system. Therefore the login is rejected.

2. The user might want to join the network. If this is the case, the user creates a new user-node with minimal information and adds it to the overlay network via the bootstrap node. The login process is then continued as described above.

3.3 Modifying Data

It is essential that modifying data is only allowed by the owner. The private key of the user ensures that no other user can replace existing data with new valid data. The responsible node for a data item can also check the integrity of a user who wants to modify the data by checking if the user has the corresponding private key for the data item. If the responsible node is adversary it could overwrite the existing data with arbitrary invalid data. The next requesting user would realize that the requested data is not correctly validated. As a consequence the requesting user would then send the request to a node which has this data replicated. Therefore the only user who can successfully modify data is the owner himself.

In some situations it is necessary that multiple users are allowed to modify the same data item. For this all users need to know the private key of the data item. The private key could also be shared with an ACL. For this the ACL approach explained above has to be extended by a permission flag. Depending on if the requested user has read or write permission the symmetric key or the private part of the asymmetric key is revealed to the user. The previously identified disadvantage of ACLs that changing permissions involves many operations is even more present. If a users write permission is withdrawn a new private key and thus a whole new asymmetric key has to be created and distributed to all remaining authorized users. If the public key is used as a node-ID even the node has to be moved in the overlay network to a new position. Therefore the advantage of using the Gentry-Waters Broadcast Encryption is even bigger. The Gentry-Waters Broadcast Encryption could also easily be adopted to support different permissions, e.g. by using two different sets, one for users with read access and one for users with write access.

The overlay network should also support the deletion of data items. Simply deleting the data item might not always be appropriate. Therefore LifeSocial uses tombstone objects which overwrite the existing data item. This tombstone objects are signed by the previous owner of the data and thus future requester can prove that this data item was really deleted and not only removed by some adversary node.

3.4 Direct Communication

A common functionality in social networks is the direct communication between two nodes, e.g. chatting or file transfer. Communicating directly is only possible if the node is online. For this the node is looked up using the overlay network and its online status is checked. If the node A wants to establish a direct connection to the online node B, A creates a symmetric key for the data exchange, encrypts it with the public key of B and signs it with his own private key. B can then extract the symmetric key with the private key of B and check the integrity with the public key of A. After this setup procedure the two nodes can safely communicate to each other using the symmetric key. The same approach can be used for multiple simultaneous connections where every communication peer uses the same symmetric key. This is, for example, needed in group chats.

3.5 Indirect Communication

Indirect communication includes private offline messages and wallposts. PeerSoN [6] suggests to store these asynchronous messages in the DHT marked with a specific flag. The next time the user logs in, it processes all pending asynchronous messages and integrates them into its data structures. On the one hand, this method might be applicable for data which is only seen by the user itself like private messages; on the other hand, it is not applicable for wallposts because they should also be visible to visitors of the user's profile before the user logs in the next time.

A more appropriate approach would be to use data items for collaboration on which several user have write access, as explained in Section 3.3.

3.6 Censorship of Illegal Data

One of the strength of a distributed OSN is that the user has full control over his own data. In some cases, this strength can also be a disadvantage. Some kind of data should under no circumstances be in the social network, e.g. child pornography. However, by the distributed design and the confidentiality of the system it is not possible to effectively solve this problem.

In the case that a user stores some illegal data in the system and does not allow any users to retrieve this data, no one, but the user himself, will be able to read the data. This also applies to a group of people who agree on the illegal data, as the data will be hidden from everyone outside of this group.

There are some solutions for content pollution [9] in file-sharing networks. These reputation systems assign reputations to either users or single data items. When querying for available data the search results are filtered according to the given reputation. However, data with bad reputation is not deleted but only hidden from the search results.

For distributed OSN there has not yet been any research for distributed censoring. In our scenario we would also like to be able to remove illegal content from the OSN. So there would have to be a way to delete badly rated data from the system. One idea to achieve this would be that all nodes responsible for some data item regularly check the reputation of the data item. If the reputation is beyond some threshold the data item is overwritten by an invalid tombstone. One problem with this is, that this could quickly end up in a reverse attack where adversary nodes mark everything as abusive. Therefore, the exact implementation of a reputation system would have to be deliberate.

Another completely different approach is to take a note of the IP-address of the user who offered illegal data. This is possible as the used peer-to-peer system is not anonymous. With this information legal measures can be taken.

4 Attacks and Defenses

This section shows different attacks performed on the overlay system. The two most prominent attacks in this category are the Sybil and the Eclipse attack. These provide the attacker with a way to obtain the ownership of a large number of nodes. Starting from this, the real attacks on storage and routing can be effectively performed.

4.1 Sybil Attack

If there are too many adversary nodes in a peer-to-peer system it will not be operable anymore. The probability of having an adversary node on the path from one node to the node responsible for a specific item increases with the number of adversary nodes. Therefore the accumulation of many adversary nodes has to be avoided.

Usually an attacker would not be able to obtain enough adversary nodes to really harm a peer-to-peer system. However, the Sybil attack [12] offers a way to obtain a huge number of nodes in a peer-to-peer system. The trick is to flood the peer-to-peer system with many nodes from the same physical device. The peer-to-peer system will have no knowledge about the connecting physical device and thus accepts many identities of a single physical device.

There are different types of defense methods [21] against the Sybil attack of which each has its disadvantages. In the following a centralized and a distributed approach is pointed out. In addition to these, the approach of using social relationships between the nodes to verify valid nodes is briefly explained, because this is particularly interesting in our case of deploying a distributed OSN. Other approaches that are not further examined include physical network characteristics, computational puzzles and game theory.

4.1.1 Centralized Certification

Douceur showed in the initial paper [12] addressing the Sybil attack that it is only possible to completely defend this attack by using a trusted centralized identification authority. The centralized authority needs to be able to detect Sybil attackers, e.g. by using certificates which are bound to the IP address [7]. However, only relying on the IP address is not necessarily save enough because some attackers might have several IP addresses, either by using a botnet or by assigning different IP addresses to their computer. One solution to this would be to charge money for each certificate, although this has the disadvantage that normal users might not want to participate if they have to pay money, even if it is only a small amount. A centralized identification authority also has the disadvantage that there is a single point of failure in the system. Although being the only real solution to this problem it is not applicable if a completely decentralized OSN has to be created.

4.1.2 Distributed Approach

Although it is impossible to create a completely Sybil-resistant distributed approach, it is possible to come up with a distributed approach which prevents Sybil attack to a high percentage. The self-registration approach by Dinger and Hartenstein [11] allows every IP-address to join only a limited amount of time. This is done by sending a join request to r different nodes

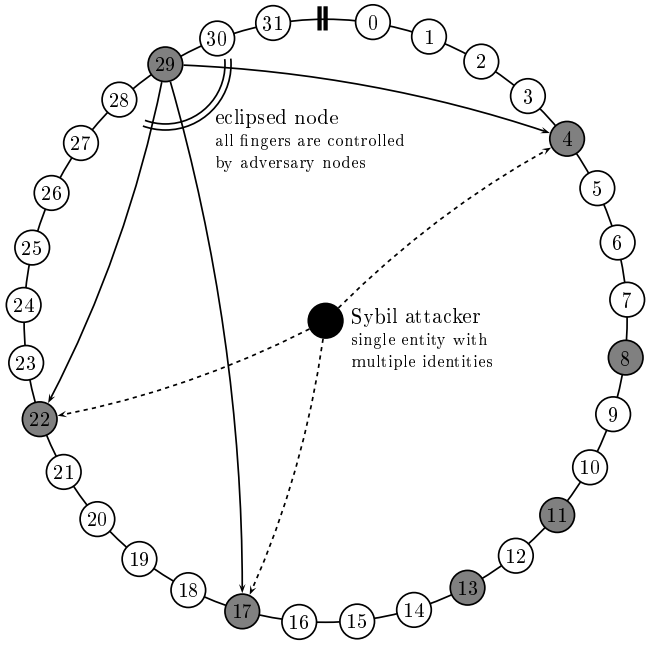


Figure 4. An example of a Sybil attack and an Eclipse attack on a DHT system.

which are calculated using the IP-address. A node accepts a join request if the specified IP-address has not yet joined more often than the specified maximum. Only if $\geq \lceil \frac{r}{2} \rceil$ of the r nodes accept the join request the node is allowed to join. Tests implementing this approach with the peer-to-peer overlay Chord achieved very good results.

4.1.3 Social Network

This approach uses relationships between all participating nodes to validate honest nodes. The relationships of all nodes form a graph, where each relationship is represented by an edge connecting the two nodes with each other. The idea is that even if there are many malicious nodes because of a Sybil attack they will only be connected to the network with a few edges (the attack edges). The amount of attack edges does not increase if the attacker adds more malicious nodes. This is because additional nodes of the attacker will only have relationships to existing malicious nodes. This results into a social graph that is split into two parts, one with honest nodes and one with malicious nodes. These two parts are only sparsely connected by the attack edges.

The basic technique which is used by the state-of-the-art methods SybilLimit [23] and SybilInfer [10] is to create a random walk through the network of a specific length for each node. If a node A wants to validate a node B, it has to compare the nodes in its own random walk with the nodes in the random walk of B. If a sufficient number of nodes in these two walks are the same, B is probably in the same subgraph as A and thus can be validated. The main difference between SybilLimit and SybilInfer is that SybilLimit uses a deterministic approach and SybilInfer uses a statistical approach.

This approach is based on relationships between all participating nodes and thus it is particularly interesting for the scenario of a distributed OSN.

4.2 Eclipse Attack

The Eclipse attack [19] is a routing table poisoning attack. The routing table entries of a specific node are announced via routing updates. If several adversary nodes manage to announce incorrect routing updates to a node, its routing table can get filled with wrong data. In the consequence a node can get isolated from all benign nodes. Then, whenever a routing request arrives at the node, it will forward the request to one of the adversary nodes. In a typical attack the attacker would try to manipulate all routing tables which point to a specific item in the overlay network. Doing this the attacker gains control over the item.

The desired solution to this problem is that the fraction of adversary nodes in every routing table is similar to the fraction of adversary nodes in the overall system. Literature have presented a number of solutions to this attack over the last 8 years. The existing solutions to this problem usually result in a performance trade-off.

4.2.1 Constrain Routing Table Entries

As every node of an overlay network has a unique node identifier, the routing table entries could be constrained to be from a specific node identifier range. Then an adversary node can only announce routing updates to nodes in its neighborhood which limits the impact of a single adversary node. This basic defense method is used e.g. in Chord.

In the case of getting control over a specific item the attacker would have to control several nodes in the neighborhood of that item. To prevail this kind of attack the node identifiers have to be random and stable [21]. If this is not the case it is possible to get to any identifier range by leaving and rejoining the network as long as the assigned node identifier is not in the desired range. Random and stable node identifiers can be either achieved by using a central authority or by assigning new node identifiers to all or some nodes in the overlay network whenever a new node joins [5].

4.2.2 Trusted Routing Table Entries

These approaches [7, 8] use two different routing tables, an optimized and a trusted one. The trusted routing table only contains entries which can be in some way trusted, e.g. constrained by their node identifiers. The optimized routing table can have any routing table entries and is normally used. Only if the looked up entries in the optimized table are suspicious the trusted routing table is used. It was shown that after some time the optimized routing table will eventually contain only adversary entries, always leading to a fault detection when the routing is done with the optimized table, followed by an additional routing with the trusted routing table. Therefore it is useful to overwrite the optimized routing table after some time with the trusted routing table [8].

4.2.3 Limit Degree for Each Node

This approach [18] is based on the assumption that adversary nodes have more ingoing edges than benign nodes because they propagate themselves as the next routing destination to as many nodes as possible. Therefore it would be possible to limit the amount of ingoing edges to some average value. It is also necessary to limit the amount of outgoing edges because otherwise a single adversary node could increase the number of ingoing edges for every other node in the overlay network by one.

This is realized by using a backpointer set for every node. The backpointer set stores which nodes have a routing table link to the node. The number of outgoing and ingoing edges is controlled with an auditing process. In this process a node a requests a node b to send its backpointer set. If the set is too big or does not contain a, the node a drops the connection to node b. The auditing is always done through a single intermediate node to anonymize the origin of the audit. Only if the auditing fails several times using different intermediate nodes the connection is dropped.

4.3 Routing and Storage Attacks

Both, the Sybil and the Eclipse attack make it possible for an attacker to efficiently control more nodes respectively more routing table entries than usual. This is a good starting point for an attacker to start the real attack. The real attack [21, 22] might be routing attacks, which announce wrong next hops or refuse to answer at all, or storage attacks, which claim ownership for requested data and answer requests with wrong data.

Storage attacks can be detected with the described encryption techniques in Section 3.1. For the case that a request is not correctly answered, i.e. not answered at all or answered with invalid data, we need an alternative way to retrieve the data. For this it is necessary to replicate the data on different nodes. The data can either be replicated at nodes close to the original data node or at random positions. Storing the data at nodes close to the original data is the more prominent method which is also used by Pastry and Chord.

Routing attacks can be defended by using different paths to look up a key. In general there are two different techniques [16] for this, multiple paths and wide paths (c.f. Figure 5).

With multiple paths the requester sends the same request to different nodes, these nodes have different routing tables and thus likely also choose different second hops. It is more likely to have one out of multiple paths without an adversary node, thus this technique increases the probability of getting a correct answer. The result could be further improved by using multiple distinct paths, so that all paths have to consist out of different nodes. In this scenario every adversary node can only manipulate at most one path. The problem with multiple paths is that one adversary node in a path is sufficient to invalidate



(a) multiple path (b) wide path

Figure 5. The difference between multiple and wide paths.

the whole path. If f is the fraction of adversary nodes and p is the path length the probability of having a path without an adversary node on it is: $(1-f)^p$ [21]. The probability to have 1 out of m multiple distinct paths without an adversary node on it is: $1-(1-(1-f)^p)^m$. So if we would have $f = 0.25$, $p = 5$, $m = 5$ we would get a probability of $\sim 75\%$ to have at least one good path. This would not be acceptable in a working environment.

A better solution is to use wide paths. A wide path is a single path with a width > 1. The path is split into different layers from 0 (at the requester) to l at the final destination. All nodes of one layer always collaboratively find out the nodes of the next layer and forward the request to each node of the next layer. With this technique a path is successful if at any layer there is at least one benign node. Let f be the fraction of adversary nodes, w the width of the path and p the length. Then the probability of at least one benign node in one layer is $1-f^w$. This has to hold in all p layers leading to a probability of: $(1-f^w)^p$. With the same values as for the multiple path we get a probability of $\sim 99.5\%$ to have a succeeding path. This approach can be done in an iterative or recursive manner.

To reduce the network load on the overlay network the first request could be done with a single path look-up. Only if the request is not answered or answered with invalid data the look-up would be repeated in failsafe mode which would either use the multiple path or the wide path technique.

5 Evaluation for Project Group

We have shown how to securely perform the login request. The approach of LifeSocial, which creates an asymmetric key with the user credentials and uses the public part of the key as the node-ID, seems to be a very elegant approach. Using asymmetric keys to sign and verify data items is a good way to prevent storage attacks because it offers a way to prove the integrity of data. Furthermore, the asymmetric keys can be used to share symmetric keys which are needed for secure communication.

This paper examined two different techniques to achieve access control. The usage of Access Control Lists is simple but requires more computational load when the authorized users change. The computational load is in the order of approved authorizations. The novel approach of Gentry-Waters Broadcast Encryption reduces the computational load to a constant. Although this approach is much more complex it should be favored over using Access Control Lists. However, the used approach should support different access permissions, i.e. read and write access to the data item, which we showed is possible with both approaches.

The preferable approach to combat routing attacks should be wide paths as it is more powerful than using multiple paths. The routing could first use a single path and only if the response is suspicious the wide path method should be used as a fallback. This is possible because it is easy to detect suspicious routing answers.

It is also necessary to prevent Sybil and Eclipse attacks as they can seriously harm the correctness and performance of the overlay network. The social network approach for combating Sybil attacks sounds very promising because social relationships already exist in the network. SybilLimit is not applicable because it is a deterministic approach which cannot cope with frequent addition of new active nodes, thus SybilInfer is to be preferred. The centralized certification, although being the only safe method, has the big disadvantage of introducing a central authority. For preventing Eclipse attacks the third introduced approach of limiting the in and out degree for each node sounds most promising.

Eclipse attacks might be especially dangerous if using a static node-ID allocation as with the public key approach of LifeSocial. If the attacker knows the used hash function he can precompute several username/password combinations which all map the public key to the node-ID range he wants to attack. So the attacker could easily control almost all active nodes in the neighborhood of a node he wants to eclipse. It is not even necessary to send many routing table updates to other nodes as the nodes of the attacker will control almost all responsible nodes of the attacked node by the design of the DHT. Therefore it is particularly important that the used hash function, which maps all data to the id space of the overlay network, is not revealed to the public.

6 Conclusion

Online social networks are an integral part of many people's everyday life. Centralized approaches have some permanent security flaws which are based on the centralized architecture. Therefore using a distributed approach is a possibility to overcome these security flaws. In this paper, we discussed how to fulfill several security and privacy requirements for a distributed framework. Necessary encryption schemes were discussed as well as a secure login process, fine grained access control, and secure synchronous and asynchronous communication between users. It was also looked at how an existing system can prevent and detect existing attacks. The attacks we looked at are the Sybil attack, Eclipse attack and different kind of storage and routing attacks. The applicability of the presented approaches for our project group was also discussed. The result is that it is possible to build a secure distributed online social network and reduce the impact of attacks enormously.

References

[1] Alexa Top 500 Global Sites, 2011 (accessed May 29, 2011). `http://www.alexa.com/topsites`.

[2] Decentralize the web with Diaspora, 2011 (accessed May 29, 2011). `http://www.kickstarter.com/projects/196017994/diaspora-the-personally-controlled-do-it-all-distr`.

[3] Facebook set to be worth '$100 billion' next year, 2011 (accessed May 29, 2011). `http://www.telegraph.co.uk/technology/facebook/8490376/Facebook-set-to-be-worth-100-billion-next-year.html`.

[4] LinkedIn share price more than doubles in NYSE debut — Reuters, 2011 (accessed May 29, 2011). `http://www.reuters.com/article/2011/05/19/us-linkedin-ipo-risks-idUSTRE74H0TL20110519`.

[5] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. *Theory of Computing Systems*, 45(2), 2009.

[6] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. PeerSoN: P2P social networking: early experiences and insights. In *Proceedings of the EuroSys Workshop on Social Network Systems (SNS)*. ACM, 2009.

[7] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 2002.

[8] T. Condie, V. Kacholia, S. Sank, J. M. Hellerstein, and P. Maniatis. Induced Churn as Shelter from Routing-Table Poisoning. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2006.

[9] C. P. Costa and J. M. Almeida. Reputation Systems for Fighting Pollution in Peer-to-Peer File Sharing Systems. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2007.

[10] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil Nodes using Social Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2009.

[11] J. Dinger and H. Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2006.

[12] J. R. Douceur. The Sybil Attack. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*. USENIX, 2002.

[13] C. Gentry and B. Waters. Adaptive Security in Broadcast Encryption Systems (with Short Ciphertexts). In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. IACR, 2009.

[14] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz. Practical Security in P2P-based Social Networks. In *Proceedings of the International Conference on Local Computer Networks (LCN)*. IEEE, 2009.

[15] F. Günther. Safebook Key Management. In *Seminar P2P im Sommersemester*. TU Darmstadt, 2010.

[16] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks. In *Proceedings of the International Conference on Distributed Computing (DISC)*. EATCS, 2003.

[17] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*. IFIP / ACM, 2001.

[18] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*. IEEE, 2006.

[19] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*. USENIX, 2002.

[20] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1), 2003.

[21] G. Urdaneta, G. Pierre, and M. van Steen. A survey of DHT security techniques. *ACM Computer Surveys*, 43(2), 2011.

[22] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proceedings of the International Symposium on Software Security (ISSS)*. Springer, 2002.

[23] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks. In *Proceedings of the Symposium on Security and Privacy (SP)*. IEEE, 2008.

Services, Apps and OSGi

Andreas Volk
University of Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: avolk@mail.upb.de

Abstract

To overcome some limitations of the Java programming language, the OSGi Service Platform provides a dynamic module system for Java. It defines a bundle as its modular unit that is deployed using the JAR file format. A bundle can be installed, updated and removed at runtime, without restarting the whole system. Unfortunately, the distribution of bundles is not addressed by the specification of the OSGi Service Platform. A common solution is, to provide a central repository that allows to discover and deploy bundles. Because this approach suffers from typical server-based problems, a p2p approach may be a better solution. This paper gives first an overview of the OSGi Service Platform, presents some popular implementations and sketches an approach how to use it for a distributed p2p framework. In its second part, it presents some ideas that are found in literature to create a p2p-based distributed bundle repository and reasons on an approach how to adopt these solutions.

1 Introduction

Java is one of the most important programming languages. Although its popularity has steadily increased since its peak at around the turn of the millennium, in 2011 it is still on top of the TIOBE Programming Index[1]. This may result from the fact that Java was originally designed to be both, simple to use and easy to learn. Despite that, it uses a Virtual Machine that makes its code portable, allowing it to run on many platforms. Also, it is strongly backed up in industry by its inventor, Sun Microsystems.

Java is a prominent example of the object-oriented programming (OOP) paradigm. Unfortunately, neither Java nor one of its competitors does fulfill the promises the hype about OOP has given in the 1980s. For example, the reuse of previously created components from a repository such that you only have to plug them together to have a working application, has never achieved a level as it was predicted at that time [7].

Nevertheless, software is getting more and more complex. To efficiently handle the amount of code and functionality, it is common to divide the software into independent parts. At best, this so called *modules* [6] need to have a low coupling to the outside and a high cohesion in the inside. They should only depend less on other modules, but strongly focus on its functionality inside the module. That also corresponds to the OOP principles of encapsulation and information hiding.

The Java module system [13] consists of classes and packages that group classes. Unfortunately, the information hiding principle is only applied to classes. It is not possible to restrict access to a public class that is defined in a package. Furthermore, the Java module system is rather static. It is not possible to dynamically remove or update modules. To achieve, this the whole system has to be restarted.

The typical unit for deployment in Java is a JAR[2] file. It contains Java classes and other resources needed to provide some functionality. A typical application is often composed of multiple JAR files that contain different components of the application. Although a JAR file is meant to be stand-alone, the JAR files forming an application normally have dependencies among each other [1]. But these relationships can not be expressed inside the JAR file. They are only available implicit from inside the classes, raising a `ClassNotFoundException` if it is executed, but a dependency is not satisfied. Also, a JAR file does not have a version defined. Missing this information makes it really hard to maintain different versions of a JAR file. Even more, providing different versions of a JAR file for different components within a single application is not possible. At

[1] TIOBE Programming Index, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[2] Java Archive Format, http://download.oracle.com/javase/6/docs/technotes/guides/jar/jar.html

last, there is also no mechanism for information hiding. As mentioned above, in Java, classes in a package can be marked as public or package internal. But if a class from an other package, that resides inside the same JAR file wants to have access to this class, the class has to be public. This implicates that every other component residing in the application also has access to this class, even though it is meant to be internal only.

To overcome these limitations of the Java module system, the OSGi Alliance has created the OSGi Service Platform. It defines a framework that provides a dynamic module system for Java. The OSGi Service Platform helps to create applications from independent, exchangeable components that work reliable together.

The structure of the paper is organized as follows. Section 2 introduces the OSGi Service Platform. It describes the OSGi Framework, gives an overview of available implementations and sketches an adaption of the OSGi principles to a p2p framework. Section 3 introduces the idea of a distributed repository that stores modules of an application in a p2p network. It presents some ideas found in literature and reasons on an approach how to adapt these solutions to a p2p framework. Section 4 concludes the paper.

2 OSGi Service Platform

> "The OSGi Service Platform specification delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion." – OSGi Service Platform Core Specification [9]

The *OSGi Service Platform* was created in the late 1990s by the OSGi Alliance [7]. As part of their process to standardize home automation, they had to create a software system that was made out of variable components from different vendors. These components should be independent, but work reliable together. Although the OOP paradigm defines a basic approach, it did not fulfill all of their requirements. They needed a model where applications could reuse resources and functions, but without knowing a priori what is available. They wanted a dynamic system that can update its components without shutting down the whole application. And they needed a system that is open and runs on different target platforms. At that time, their requirements could not be realized with existing solutions. They had to develop one on their own.

The *OSGi Alliance* itself was formed by a group of companies with members including IBM, Ericsson, Nortel, Sun Microsystems, Motorola and Philips. As a matter of fact, the OSGi Service Platform was and is still heavily backed up by industry and as such is used in many different areas and has proven to be a mature solution [1].

Though, the OSGi Service Platform was not planed as a concrete implementation. It is rather a specification that every vendor may implement on its own. As long as every implementation obeys the interfaces defined in the specification, they will work together. The only restriction the specification mentions is that Java is used as the programming language. Of course this is due to the fact that Sun Microsystems is an OSGi member.

The OSGi Service Platform specification is steadily evolving, adapting to the needs of its implementations and usage scenarios. It started with Release 1 in 2000 and is at the time of writing, in 2011, at Release 4.3 [9]. Next to their own specification, the OSGi Alliance is also trying to include some of their proposed ideas into the Java language itself. They are working on two Java Specification Requests (JSR)[3], that define their dynamic module system for all Java platforms.

2.1 OSGi Framework

The main part of the OSGi Service Platform specification is the *OSGi Framework* [9]. It provides the basic concepts that are needed to build a modular application on the principles of OSGi . Next to the OSGi Framework, the specification defines a set of Standard Services. These services are not required for an implementation to provide, but are nice features for the users of the implementation. Examples for Standard Services are the Log Service or the HTTP Service. The following sections introduce the basic concepts of the OSGi Framework. They describe some internals that are needed for developers to start using an implementation and helps to better understand the behavior of the framework.

2.1.1 Service-Orientation

The OSGi Framework is based on a service-oriented approach [5]. With the buzzworded Service-Oriented Architecture (SOA), this approach has gained a lot of attention in the last years. In contrast to SOA, the OSGi approach does not use Web Services or any additional communication layers. Nevertheless, it follows the same general concept of service-orientation: *late non-explicit binding*.

[3] JSR 291 Dynamic Component Support for Java SE, http://www.osgi.org/JSR291/,
JSR 232 Mobile Operational Management, http://www.osgi.org/JSR232

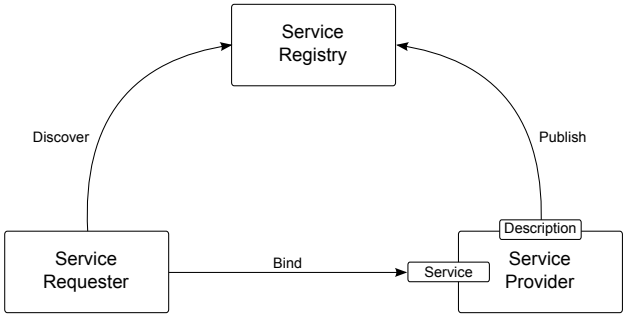


Figure 1. Service-Orientation [5]

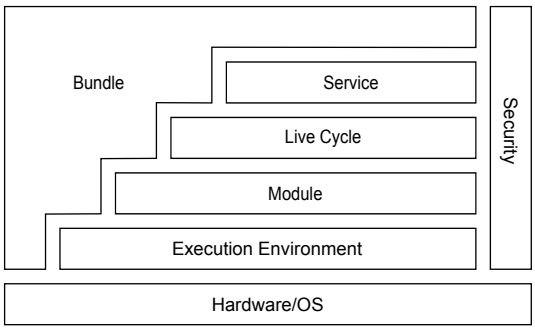


Figure 2. Architecture of the OSGi Framework [9]

A service-oriented approach is based on the key concepts of a service, service provider, service requester, service description, and service registry. Figure 1 provides an overview of this approach. A *service* is some sort of functionality a *service provider* is offering. Its functionality as well as how it is used is described in a *service description*. This description is published to the *service registry* by a service provider. The service registry keeps a list of available services, i.e. service descriptions, and in addition, which provider offers this service. A *service requester*, who wants to use some service, is now able to discover a specific service in the service registry. He binds to the appropriate service provider and uses the service according to his needs. In this concept, the binding between a service requester and the service is established at runtime, i.e. late. Because there may be more than a single service provider offering a specific service, it is also non-explicit. As long as the service conforms to the description, its implementation details do not matter.

2.1.2 Services and Bundles

A service in the OSGi Framework is represented by a plain old Java object (POJO) [7]. The service is described by its service interface and implemented as a service object. Although there is a semantic difference between the interface and its implementation, in practice they are often mixed in a single class. In this case the service interface is defined by the public methods of the class. However, more common is to separate interface and implementation. This enables different services to provide implementations for a specific interface.

The OSGi Framework groups services in modules, so called *bundles* [9]. A service is owned by and run inside of a bundle. This bundles act as both, a service provider as well as a service requester. They publish their own services to the service registry and discover and use services from other bundles. In terms of OSGi , these bundles are the components an application is composed of. It consists of a set of Java packages that provide some functionality. Bundles are deployed in a traditional JAR file that contains the classes of its corresponding Java packages as well as additional resources. Although it is a JAR file, a bundle has its own MIME type: `application/vnd.osgi.bundle`.

Figure 2 provides an overview of the complete architecture of the OSGi Framework. It is a layered architecture [9], with

the hardware and the operating system at the bottom. On top of that is a layer that defines the execution environment, i.e. the Java Virtual Machine (JVM). The layer sitting above defines the dynamic modularization model for Java. It manages the bundles that are deployed in the framework. These bundles have a certain life cycle that is defined in the next layer. It defines how they can be installed, started and stopped. The service layer provides the dynamic service model mentioned above. Bundles may register services at runtime, provide concrete implementations and bind to services. A bundle uses all of this layers to provide its functionality. In addition, there is also an optional vertical security layer that is based on the Java 2 Security Architecture. Amongst others, it provides code authentication through signatures or access control for the bundles.

2.1.3 The Manifest

A bundle contains a special file, the so called *manifest*. The manifest describes the JAR file and provides additional information about the bundle itself as well as its dependencies. An example manifest from [13] is shown in Listing 1.

```
 1: Manifest-Version: 1.0
 2: Bundle-Name: dictionary_handler
 3: Bundle-Activator: org.dictionary.handler.activator.HandlerActivator
 4: Bundle-SymbolicName: dictionary_handler
 5: Bundle-Version: 1.0.0
 6: Bundle-RequiredExecutionEnvironment: CDC-1.0/Foundation-1.0
 7: Export-Package: org.dictionary.db.search,
 8:  org.dictionary.handler.translate
 9: Import-Package: org.dictionary.db.search,
10:  org.osgi.framework;specification-version="1.0.0"
```

Listing 1. An Example Manifest [13]

The manifest is a simple text file and contains a list of attributes that have of a name and a value. The attribute `Manifest-Version` in line 1 is a common attribute and describes the manifest file version, according to its specification[4]. The attributes in the following lines are OSGi specific. Most of them are optional and can be omitted. `Bundle-Name` provides a short and human readable name for the bundle. The `Bundle-Activator` defines a special class in the bundle that is instantiated when the bundle is activated. The `Bundle-SymbolicName` is the only required attribute and defines a unique name for the bundle. It should be based on the reverse domain name notation and is used to identify the bundle, together with the optional `Bundle-Version`. `Bundle-RequiredExecutionEnvironment` defines an execution environment that is needed for this bundle. In the given example it is the J2ME Foundation Profile[5].

The `Export-Package` attribute defines some services the bundle is offering. It lists a set of Java packages whose classes are automatically registered in the service registry and may be used by other bundles. The bundle exports this packages. The `Import-Package` attribute is the counterpart to `Export-Package`. It defines dependencies to other services. It lists Java packages that are used in this bundle, but are not provided by its own. In order to use this bundle, these services have to be present in the service registry. This implies that there are bundles that provide implementations for this Java packages and exported it through `Export-Package`. The bundle imports this packages.

The advantage of `Import-Package` is, that it does not indicate that a special bundle is present in the system. It only assumes that there is some bundle providing an implementation for this Java package. This allows to have different implementations for the same package. Related to the `Import-Package` attribute, there is also the `Require-Bundle` attribute. It defines a list of bundles, specified by their `Bundle-SymbolicName`, that have to be present in the system. By doing this, the bundle imports all of the exported Java packages from the bundles in the list. Although this eases dependency management, it creates a tight couping between the bundles. To avoid this coupling, it is recommend to use the `Import-Package` attribute.

Next to the mentioned attributes, there are plenty of others defined in the OSGi Framework [9]. For example to categorize bundles, store copyright notices or define icons for the bundle.

[4] JAR Manifest Specification, http://download.oracle.com/javase/6/docs/technotes/guides/jar/jar.html#Manifest-Overview
[5] J2ME Foundation Profile, http://java.sun.com/products/foundation/

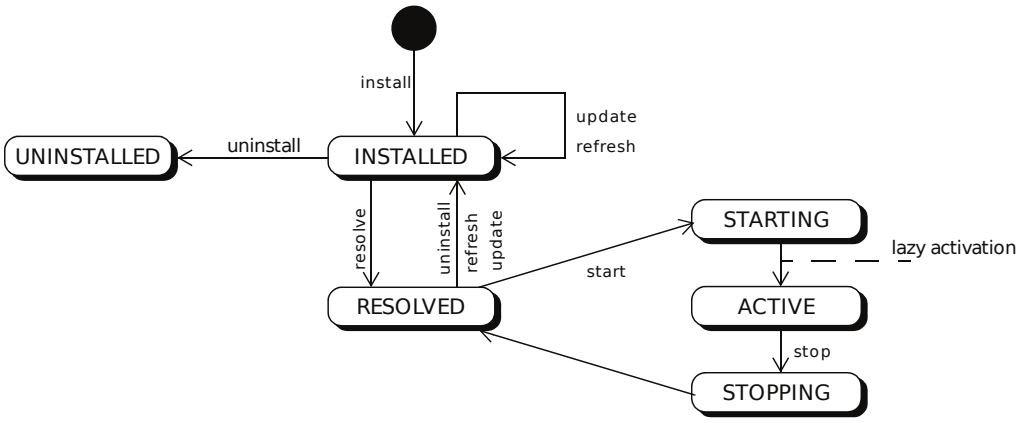


Figure 3. Life Cycle of a OSGi Bundle [9]

2.1.4 Bundle Management

In an OSGi application, every component is defined to be a bundle. With respect to this, also the framework by itself is represented as a special bundle, the so called *system bundle* [9]. Through this bundle, the framework is offering its own services to other bundles. It defines the service registry, provides access to a persistent data storage, or implements a logging service.

Next to the system bundle, there is another bundle that has a special role: the *management agent* [4]. It is designed as a regular bundle but has additional administrative privileges. That means it has access to the core of the framework and is allowed to install bundles. While the actual installation of a bundle is done by the framework, the management agent selects the bundles the user is going to install. The problem in its task is, to keep track of the dependencies of the bundles. A bundle can be easily installed, but it can only be started if every dependency, it has defined in its manifest through the `Import-Package` and `Required-Bundle` attributes, is satisfied. If a bundle has unresolved dependencies, the management agent has to provide a bundle that provides this dependencies. How exactly this is done, is not defined in the specification. It is left up to the implementations to handle the dependency management properly.

Next to the dependency management, the management agent is also responsible for managing the active configuration. The management agent is able to install, start, stop, and uninstall bundles at any time. He actively adapts the running bundles to the current needs of the user. This process is independent of the current state of the framework, because it restores the active configuration automatically after a restart. To manage the active configuration, the management agent has to control the bundles life cycle.

2.1.5 The Life Cycle

Figure 3 depicts the different states of a bundles life cycle. The life cycle of a bundle starts by installing it. This is usually done by specifying a location string, normally an URL pointing to the bundles jar file. The framework retrieves the content of the bundle from the given location and tries to validate it. It checks its manifest file for syntactic errors and ensures that it contains all required attributes, for example the `Bundle-SymbolicName`. If the bundle is valid, the framework stores it in its own persistent storage. The whole installation process is both, atomic as well as persistent. A bundle is either installed complete or the framework is reset to a state that is the same as it was before it tried to install the bundle. Furthermore, a bundle remains installed until it is explicitly uninstalled by the framework.

Once a bundle is installed, it is not yet ready to be used. It first needs to get resolved. That means, all dependencies described in its manifest needs to be available from bundles, including its own and the system bundle. As mentioned above, common dependencies are imported packages, required bundles, the execution environment, and native code. The execution environment is typically provided by JVM through the system bundle. Native code are platform dependent libraries that

needs to be loaded by the JVM.

After a bundle has been resolved, it is ready to get started and activated. This is the first that time some functionality provided by the bundle is triggered. If the bundle has defined a `Bundle-Activator`, the Activator is created and notified that the bundle is starting. With the Activator, the bundle is able to initialize some functionality, start threads, or register services. The Activator is optional and for example not need in a bundle that only provides some libraries. Services defined in the manifest as exported packages are registered automatically. Typically, the activation of a bundle is immediately done after starting the bundle. But it is also possible to use a lazy activation policy. That means, the activation process is deferred until a first class from this bundle is loaded. This reduces memory usage and improves the start up time, if a lot of bundles are loaded, but not immediately needed. However, the loading of other resources than classes does not trigger the activation of the bundle.

An active bundle can be stopped at any time. If it defines an Activator, it gets notified and should clean up all resources, allocated in the activation step. He also has to stop all of its running threads and may unregister its services. Though, from the frameworks point of view, this is not necessary. Because the framework has to ensure consistency and keeps a list of all services registered, it removes all services from that bundle automatically.

To uninstall a bundle, the framework removes its contents from its persistent storage and tries to establish a state as if the bundle has never been installed. Only if the bundle has exported some packages that are used by another bundle, these packages remain available until the framework gets stopped or restarted. This ensures that no other bundle breaks immediately by uninstalling this bundle.

With the knowledge of bundles and services provided above, one should now be familiar with the basic concepts of the OSGi Framework and be able to understand and use an implementation of the OSGi Service Platform.

2.2 Implementations

As mentioned above, the OSGi Service Platform was designed in a way that every vendor could implement a framework on its own. The OSGi Alliance provides a detailed specification but not a reference implementation. The following presents some popular open source implementations of the OSGi Service Platform.

- The probably most used, but, at least by its name, less known OSGi Service Platform implementation is *Equinox*[6]. It is the core runtime of the famous Eclipse project. As of version 3.0, the developers of Eclipse decided to move away from their current proprietary plugin-based implementation [4]. They explored different solutions, but selected OSGi because of its dynamic environment and the existing momentum of OSGi technology in industry. Since the adaption of OSGi in Eclipse, a lot of extensions that were developed in Equinox were included in later revisions of the OSGi specification. For example, the now essential attribute `Bundle-SymbolicName` was first introduced in Equinox as well as the possibility to require bundles to be present in the system via `Require-Bundle`. It is not surprising that a lot of work that is done in Equinox is sponsored by IBM, who is also a member of the OSGi Alliance. Still, Equinox defines additional extensions that are not part of the OSGi Service Platform. For example their extensions mechanism that lets bundles easily contribute to other bundles through extension points.

- Another open source framework that implements the OSGi Service Platform is *Apache Felix*[7]. It is based on the Oscar framework, that was one of the earliest implementations. Since 2007 [1], it is a software project at the Apache Software Foundation. Apache Felix was originally designed to be compact and easy to embed. As of today, it is still the smallest framework available. A prominent example that uses Apache Felix is the NetBeans IDE, a direct competitor to Eclipse.

- A third popular OSGi Service Platform implementation is *Knopflerfish*[8]. Although it is an open source implementation, it is mostly maintained and sponsored by the Swedish company Makewave[9]. The company also offers a commercially supported version that is certified by the OSGi Alliance to be fully OSGi compliant. As such, Knopflerfish is quite used in industry, e.g. by companies such as Siemens or Volvo[10].

Though there are different implementations of the OSGi Service Platform, the bundles created for one implementation in general are interchangeable [1, 7]. This makes it easy to change the underlying framework of an OSGi application. But of course, this is only possible if the bundles do not relay on other features the implementation provides in addition.

[6] Equinox, http://www.eclipse.org/equinox/
[7] Apache Felix, http://felix.apache.org/
[8] Knopflerfish, http://www.knopflerfish.org/
[9] Makewave, http://www.makewave.com/
[10] See Makewave Showroom, http://www.makewave.com/site.en/showroom/

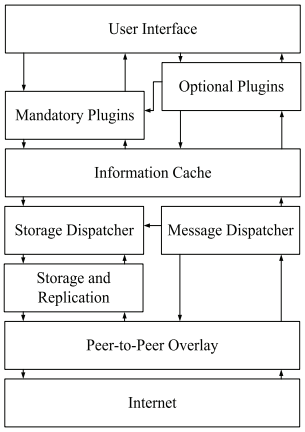


Figure 4. Architecture for a P2P Framework [3]

2.3 Adapting OSGi

Adapting OSGi in general is quite easy [1]. A developer does not need to know a lot about the theories and architectures behind the OSGi Framework. He only has to decompose his application into components and create the according bundles. Because of the great tool support that is available, for example in Eclipse, this is also not a very complex task.

2.3.1 Architecture

A popular example for an application that is build on top of a p2p framework is a distributed online social network. Focusing on this concept, an architecture for a p2p framework may look like the one proposed by Graffi et al. in [3]. Figure 4 gives a overview about the architecture.

The architecture is decomposed into components that are layered on top of each other. Every layer offers a well known service to the layer above and uses the services from the layer below. The Internet is the basis for the whole architecture. It provides the physical connectivity for the Peer-to-Peer Overlay. The Storage and Replication layer uses the p2p network to create a reliable storage for data objects. The Storage Dispatcher keeps track of the actual application data being stored, while the Message Dispatcher provides a message exchange between individual peers. The Information Cache stores previous requested data to make them instantly available upon a further request. The Plugin Layer contains the actual functionality of the application. It is split into plugins to allow an easy extension of the application. Every plugin is expected to also have a graphical user interface that is decoupled from the actual functionality of the plugin.

The components of this architecture could directly be mapped to bundles in an OSGi approach. Creating an interface for each component abstracts from the real implementation and makes them easily exchangeable.

These loose coupling of the components would make it easy to create different derivations of the application. For example, to provide storage, one could link a server to the p2p network that does not run a graphical user interface. There could be different packages to download, each containing only a subset of the available bundles, needed to provide some functionality. Following this approach, also a dedicated monitoring infrastructure could be installed, that is hosted on its own servers and does not participate to the actual application build on the p2p framework.

Using the open system of the OSGi Service Platform, would also allow third-party developers to contribute their own bundles. A user can decide on its own, which bundles he has installed on its machine. So, he can easily customize its installation by integrating bundles that provide games or replacements for default bundles.

2.3.2 Framework

As an implementation of the OSGi Service Platform for a p2p framework, Equinox would be a good choice. It is a mature project that also provides some further features that might be useful for the p2p framework. For example, the extension system defines a powerful mechanism for creating plugins. Building on the whole Eclipse infrastructure would also help to rapidly build a graphical user interface.

3 A Distributed Bundle Repository

Although the specification of the OSGi Service Platform is very comprehensive, it does not address the distribution of bundles. To install a bundle, the user, respectively the management agent, has to provide an URL to a location, where the framework finds the bundle to install. But for a real usage scenario this is most probably not sufficient. The OSGi Framework is designed highly dynamical and lets bundles be installed at runtime. But with the only definition of a simple location string, it assumes first, that the user knows which bundles are available, second, that the user knows where the bundle is located and third, that all dependencies the bundle has are satisfied a priori.

There are two principal points that needs to be addressed in the context of distribution: discovery and deployment. Discovery is the first step. The user needs to know which bundles are available. At best, he has a list of bundles, maybe even categorized, from which he chooses the bundles he wants to install. In addition, all bundles are selected, the selected bundles have dependencies to. To be more precise, the transitive closure of its dependencies needs to be selected. That means the dependencies of the selected bundles, as well as the dependencies of their dependencies, and so on, recursively defined [10].

Deployment is the next step. It defines, how a bundle is transfered to the target machine such that the framework has access to its content. This can be achieved using a network protocol or simply by plugging an USB device into the computer. If the bundle is on the machine, the framework has access to it and can install the bundle into its local persistent storage.

A wide spread solution for the distribution of bundles is the OSGi Bundle Repository (OBR2) [8]. It is based on the Oscar Bundle Repository (OBR) that was originally developed for an early version of Apache Felix. OBR2 describes a central repository that consists of a web site hosting an XML document. This document lists all available bundles as well as links to other repositories, forming a federated repository. A user may search this list by keywords or categories and select the bundles he needs. In addition to this, ORB2 also eases the selection of the transitive closure of the dependencies of the selected bundles by providing a list of required bundles.

All of the OSGi framework implementations listed above implement a mechanism that is based on OBR2. Although this is a widely used solution, it is not the best solution. The OBR2 defines a central repository, that also suffers from typical issues from server-based solutions [2, 11]. It is a single-point-of-failure and only has limited scalability in cpu, bandwidth and memory. This problems may be addressed by a p2p-based approach for a distributed bundle repository. The goal is to provide both, discovery as well as deployment with the help of a p2p network.

3.1 Current Solutions in Literature

Although both, p2p as well as OSGi has been under research for quite a long time, there are only less publications that combine this two topics to provide a solution for a distributed repository for bundles. The following section presents two approaches that are found in literature.

3.1.1 POSGi

One solution for the distribution of bundles within a p2p network is POSGi [2]. POSGi uses the FreePastry[11] p2p network to provide a distributed hash table (DHT) that stores the bundles. A hash table is a data structure that maps a key, the hash, to a value, its content. For every hash, a peer node from the p2p network is responsible to store the actual content. The relation between a hash and a peer node is automatically computed by the p2p network and as such known to every peer. In addition, POSGi defines a standardized file name format for the bundles. It consists of the name of the bundle followed by its current version. This allows version management without exposing the contents of the bundle, because the version is normally only stored in the manifest file, as well as automatic updates.

In POSGi, every peer can share a bundle. At first, it creates a hash of the bundles name. Under this hash stores the DHT the current version string of the bundle. Then the hash for the bundle name plus its version is created. Under this hash the DHT now stores the real binary content of the bundle. Listing 2 provides this two steps in pseudo code.

```
1: publish(hash(bundleName), bundleVersion);
2: publish(hash(bundleName_bundleVersion), bundleBinary);
```

Listing 2. Publishing a Bundle

To deploy a bundle, the name of the bundle is required. It is hashed and the current version string of the bundle retrieved that is stored in the DHT. After that, the bundles name plus its version is hashed and the real content of the bundle is retrieved

[11] FreePastry, http://www.freepastry.org/

from the DHT. Listing 3 provides the pseudo code for this two steps. The separation of the version of the bundle and its content ensures that always the last version of the bundle is deployed. To address a bundle in the p2p network, POSGi defines a protocol handler in the format `p2p://bundleName`. This handler encapsulates the whole deployment process and makes it accessible to the framework.

```
1: bundleVersion = retrieve(hash(bundleName));
2: bundleBinary = retrieve(hash(bundleName_bundleVersion));
```

Listing 3. Retrieving a Bundle

POSGi has a simple, but powerful solution for the deployment of bundles in a p2p network. It addresses some of the problems mentioned above, by using the features the p2p network provides, like the replication in the DHT. Unfortunately POSGi does not define a solution for the discovery problem. It is still using a central OBR2 repository for a searchable index, but provides p2p URLs.

3.1.2 D2CM and OSGi4C

Another solution that uses a p2p network to provide discovery and deployment of OSGi bundles is presented in D^{2}CM [12] and OSGi4C [11]. They are both based on the same concepts and technologies, while OSGi4C can be seen as the successor of D^{2}CM.

The solution presented there is based on the generic p2p network JXTA[12]. The nodes in the network are called peers and are organized in peer groups. Resources in the network are identified by advertisements, a XML structure that defines meta data for resources. Each advertisement contains a global unique identifier, by which it can be referenced from other advertisements.

In a peer group, resources are published as well as discovered by three advertisements. The *interface description advertisement* (IDA) describes the existence of an interface. It contains the name and the version of the interface and can be searched by the full qualified interface name or based on keywords. The actual interface definition is not part of the IDA. It is described in a referenced resource advertisement. A *resource advertisement* (RA) defines arbitrary resources. In general, it contains some data structure. As described above, it may contain an interface description and is referenced by an IDA. The third advertisement is the *code description advertisement* (CDA). It describes the existence of an implementation of an interface and is also referenced by an IDA. The CDA by itself references again a RA that contains the data necessary to load the resource, such as size, filename, and checksum. The JXTA network provides a standard service to discover this advertisements.

To find a bundle in the p2p network, a specific advertisement is created. The type of advertisement depends on the goal that needs to be reached. If a specific bundle is needed, a CDA is created that contains the name of the bundle. If an interface dependency for a bundle is needed to be resolved, an IDA is created that contains the name of the interface. This advertisement is send by the JXTA network to all of the peers in the current peer group. If a peer hosts a bundle that matches the incoming advertisement, it creates a CDA that describes the bundle and sends it back to the host that started the query.

The deployment part in this solution is quite simple. After the CDA for a certain bundle has arrived at the peer, the bundle needs be transfered to the local machine. As written above, the CDA references an RA that contains the concrete loading address for the bundle. The presented solution implements specific transfer handlers that are dynamically selected, according to the concrete loading address. The bundle by itself can be either transfered via HTTP, i.e. the hosting peer opens a HTTP server, or directly via the underlying JXTA network.

The solution presented in D^{2}CM and OSGi4C uses the benefits of the p2p network to distribute bundles in the p2p network. Bundles are deployed among the peers as well as the discovery is performed inside the network. Unfortunately, the concept is highly dependent on the advertisements of the underlying p2p network. In addition, there is no guarantee for the search to return all bundles that are available. The peer can only wait for a certain time for the CDAs to arrive from a previous search. If it does not retrieve all CDAs in time, the results are incomplete.

3.2 Adapting a Solution

Adapting a solution for a distributed bundle repository is not an easy task. There are only few solutions in literature, as presented in section 3.1, and they are hard to adapt, if the preconditions are not exactly the same.

[12] JXTA, http://jxta.kenai.com/

As the p2p framework is likely to be based on FreePastry, the approach described in POSGi [2], in section 3.1.1, would fit quite well. The DHT is a good base for the distribution of the bundles and an URL handler is a nice integration into an existing OSGi infrastructure. Also, the URL handler is defined in the OSGi Service Platform as a standard service [9] and as such straightforward to implement.

Adapting a discovery strategy is rather tricky. FreePastry has no direct correspondence to the advertisements of JXTA as used in DC^{2}M [12] and OSGi4C [11], described in section 3.1.2. But with SCRIBE[13], there exists an add-on for FreePastry that allows group communication in the p2p network. A simple approach would be, to create a group for sharing bundles. Every peer node that shares a bundle is part of that group. If another peer node wants to discover some bundle, it sends a request to the group. Every peer that hosts a matching bundle answers by its own to the peer, providing a description of the bundle and how it can be accessed. It can then be deployed using the strategy mentioned above.

If this approach is really applicable, needs to be further evaluated. The deployment part is proven to be working, but the discovery is just a rough sketch. However, it uses a similar approach as described by DC^{2}M [12] and OSGi4C [11] that actually works quite well.

4 Conclusion

The OSGi Service Platform provides a dynamic module system for the Java programming language. It defines a bundle as its modular unit that contains a set of Java packages. A bundle can export some of its own packages and import packages from other bundles. This leads to a decomposition of the application into smaller components that are loosely coupled. The bundles can be installed and removed at runtime, without restarting the whole application. The OSGi Service Platform defines a very flexible approach for creating component based applications. It is easy to adapt, available in different implementations, and has good tooling support. As such, the OSGi Service Platform is a good starting point for every Java application that is slightly more complex.

Unfortunately, the distribution of bundles is not addressed by the specification of the OSGi Service Platform. A widely used solution is a central repository that keeps a list of all available bundles. Because this approach suffers from common problems of a server-based solution, there are some approaches that distribute bundles within a p2p network. A combination of two solutions found in literature could lead to a solution that works well in a p2p framework. Bundles are discovered by a simple group communication mechanism and deployed with the help of a distributed hash-table. A p2p framework could implement a basic prototype to test this approach.

References

[1] N. Bartlett. OSGi in Practice. http://njbartlett.name/osgibook.html, 2009. DRAFT PREVIEW, Last accessed in May 2011. 1, 2, 6, 7

[2] S. Frénot and Y. Royon. Component Deployment Using a Peer-to-Peer Overlay. In *Proceedings of the International Working Conference on Component Deployment (CD)*. Springer Berlin / Heidelberg, 2005. 8, 10

[3] K. Graffi, S. Podrajanski, P. Mukherjee, A. Kovacevic, and R. Steinmetz. A Distributed Platform for Multimedia Communities. In *International Symposium on Multimedia (ISM)*. IEEE Computer Society Press, 2008. 7

[4] O. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, and T. Watson. The Eclipse 3.0 platform: Adopting OSGi technology. *IBM Systems Journal*, 44, 2005. 5, 6

[5] R. S. Hall and H. Cervantes. Challenges in building service-oriented applications for OSGi. *Communications Magazine, IEEE*, 42, 2004. 2, 3

[6] P. Jorgensen, D. Fernandez, M. Greco, B. Hussey, S. Kuchta, H. Li, S. Overkamp, D. Rodenberger, and R. VanderWal. Has the Object-Oriented Paradigm Kept Its Promise? Technical Report, Department of Computer Science and Information Systems, Grand Valley State University, 2002. 1

[7] P. Kriens. How OSGi Changed My Life. *ACM Queue*, 6, 2008. 1, 2, 3, 6

[8] OSGi Alliance. RFC-0112 Bundle Repository. http://www.osgi.org/download/rfc-0112_BundleRepository.pdf, 2005. Last accessed in May 2011. 8

[9] OSGi Alliance. OSGi Service Platform Core Specification. http://www.osgi.org/Specifications, 2011. Release 4, Version 4.3, Last accessed in May 2011. 2, 3, 4, 5, 10

[10] K. Pauls and R. S. Hall. Eureka-a resource discovery service for component deployment. In *Proceedings of the International Working Conference on Component Deployment (CD)*. Springer Berlin / Heidelberg, 2004. 8

[11] H. Schmidt, J.-P. Elsholz, V. Nikolov, F. J. Hauck, and R. Kapitza. OSGi4C: enabling OSGi for the cloud. In *Proceedings of the International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE)*. ACM, 2009. 8, 9, 10

[12] H. Schmidt, J. H. Yip, F. J. Hauck, and R. Kapitza. Decentralised dynamic code management for OSGi. In *Proceedings of the workshop on Middleware for network eccentric and mobile applications (MiNEMA)*. ACM, 2008. 9, 10

[13] A. L. C. Tavares and M. T. Valente. A gentle introduction to OSGi. *ACM SIGSOFT Software Engineering Notes*, 33, 2008. 1, 4

[13] SCRIBE, http://www.freepastry.org/SCRIBE/default.htm