Hybrid Online Social Networks

Telecooperation Lab

Masterarbeit von Carsten Porth

Tag der Einreichung: 25. März 2019

1. Gutachter: Prof. Dr. Max Mühlhäuser
2. Gutachter: Dr. Jörg Daubert, Aidmar Wainakh

Darmstadt, den 25. März 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Telekooperation
Prof. Dr. Max Mühlhäuser

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Carsten Porth, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Masterarbeit stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Masterarbeit des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, den 25. März 2019

Unterschrift

Abstract

Despite numerous scandals about data protection in established social networks (Facebook, Google+), the platforms still enjoy great popularity. Alternative social networks, which have focused on protecting the data of their users, lack attractiveness, so they regularly fail. If users remain loyal to social networks notwithstanding poor data protection, ways must be found to protect their data. A hybrid approach enables the natural use of the platform and uses another communication channel to exchange sensitive data.

The goal of this thesis was to work out a concept for a hybrid Online Social Network (OSN) and to validate it with a prototype. Within the scope of the concept, the interests of the stakeholders were identified, and requirements for the hybrid solution were defined. For the implementation, concrete solution strategies for architecture and client were mentioned. The prototype was created as an Android app for Twitter. For secure data exchange, the IPFS protocol for decentralized data storage in combination with the distributed GUN database was used.

The result shows that the concept is implementable and that the defined goals lead to a high-quality solution. The Hybrid OSN prototype meets the previously defined requirements almost entirely. For Likes and Tweets, the user can decide if the official Twitter network or the private network should be used for data exchange with other users. The solution is user-friendly and requires minimal configuration effort.

Zusammenfassung

Trotz zahlreicher Skandale um den Datenschutz in etablierten sozialen Netzwerken (Facebook, Google+), erfreuen sich die Plattformen noch immer großer Beliebtheit. Alternativen sozialen Netzwerken, die den Fokus auf den Schutz der Daten ihrer Benutzer gelegt haben, fehlt es an Attraktivität, sodass sie regelmäßig scheitern. Wenn die Nutzer den sozialen Netzwerken trotz mangelhaftem Datenschutz treu bleiben, müssen Wege gefunden werden, wie ihre Daten dennoch geschützt werden können. Ein hybrider Ansatz ermöglicht die gewohnte Nutzung der Plattform und verwendet einen weiteren Kommunikationsweg zum Austausch schützenswerter Daten.

Das Ziel dieser Thesis war die Ausarbeitung eines Konzepts für ein hybrides Online Social Network (OSN) sowie die Validierung durch einen Prototyp. Im Rahmen des Konzepts wurden die Interessen der Stakeholder aufgezeigt und Anforderungen an die hybride Lösung definiert. Für die Umsetzung wurden konkrete Lösungsstrategien für Architektur und Client benannt. Der Prototyp wurde als Android App für Twitter erstellt. Für einen sicheren Datenaustausch wurde das IPFS-Protokoll zur dezentralen Speicherung von Daten in Kombination mit der verteilte Datenbank GUN eingesetzt.

Das Ergebnis zeigt, dass das Konzept umsetzbar ist und die definierten Ziele zu einer qualitativ hochwertigen Lösung führen. Der Hybrid OSN Prototyp erfüllt die zuvor definierten Anforderungen nahezu vollständig. Für Likes und Tweets kann der Nutzer selbst entscheiden, ob über das offizielle Twitter Netzwerk oder das private Netzwerk die Daten mit anderen Nutzern geteilt werden sollen. Die Lösung ist nutzerfreundlich und bedarf nur minimalem Konfigurationsaufwand.

Contents

List of Figures

List of Tables

1 Introduction

The World Wide Web celebrated its 30th birthday in March 2019. From the beginning in 1989 up to now, the Internet has evolved and has changed the lives of many people forever. Thanks to digitalization, analog processes such as banking and shopping can now be handled conveniently online. Moreover, the ongoing technology improvements catalyze the digitalization and make it possible to be online almost anywhere. The Internet is also not stopping at the digitalization of social life. Social networks like MySpace and Facebook became popular at the beginning of the 2000s, enabling networking with other users and sharing personal content. Social networks are still very popular today. Facebook, the largest Online Social Network (OSN), has 2.3 billion active users per month, connecting almost a third of the world's population [37]. With the use of such social platforms, the user makes himself transparent for the service provider of the OSN. Personal data are a valuable asset whose protection is of particular importance. By using the platform and sharing this information, the user puts trust in the service provider. Trust that his data will be handled responsibly, stored securely, and not made available to third parties. However, the past has revealed that this is not always the case. Several incidents became public showing the lack of protection and the misuse of personal data on OSNs.

1.1 Motivation

Numerous scandals about data protection in OSNs have proven that user data are not sufficiently protected. In March 2018 it became known that the data of 87 million Facebook users were made available to the company Cambridge Analytica [43]. During a security investigation in March 2019, Facebook found that the passwords of several hundred million users were stored unencrypted in plain text [29]. After an analysis by Google revealed a severe bug in the API that allowed the personal data of 52.5 million users to be retrieved, they decided to close their platform Google+ [63].

However, although these circumstances are well known, users remain mostly loyal to their OSN. As a result of the Cambridge Analytica incident, the number of daily Facebook users dropped only briefly in Europe but is in the meantime back on the previous level [37]. Alternative OSNs, which focus on protecting their users' data, regularly fail to get a sufficiently large user base or establish a business model to ensure their operation. For example, the decentralized OSN diaspora* has less than 700 000 users after nine years and the OSN OpenBook[1] needed a second Kickstarter crowdfunding[2] round after the first one failed [27].

The binding to the respective OSN is so strong that switching to another, more secure OSN does not seem to be an option. To improve the protection of users' data on existing platforms, other ways need to be examined. The Doctoral College „Privacy and Trust for Mobile Users" works on „Privacy and Trust in Social Networks (Research Area B)" and researches on the topic of increased privacy protection. Especially the subarea B.2 „Privacy Protection via Hybrid Social

[1] https://www.openbook.social/en/home
[2] https://www.kickstarter.com/projects/1520156881/openbook-privacy-friendly-fun-and-honest-social-ne

Networks“ is about hybrid solutions which propose to add privacy-preserving approaches to established OSNs. The secure data exchange should be carried out via a Peer-to-Peer (P2P) overlay network. Since data play a prominent role in the business model of OSNs, it is necessary to provide anonymized data from the private network to keep the model running. As part of these researches, this work is motivated to provide a detailed concept for a hybrid solution to protect the user’s data and verify the idea with a prototype. [57, 58]

1.2 Contribution

The goal of this work is to define the requirements for a hybrid solution and to prove its feasibility in the form of a prototype. Within the scope of the concept for a hybrid solution, the requirements for the OSN, the hybrid client app and the network for secure data exchange have to be defined as well as potential problems and limitations. Based on these requirements, the elaboration of solution strategies for the implementation is possible.

For the prototype, an Android application that exchanges private data with other users via a P2P network is created. The previously defined requirements should be fulfilled in the best possible way. Both the selection of the OSN and the technologies used need to be carefully evaluated.

With the Hybrid OSN app for Twitter, we present a solution that allows private data to be shared securely with other users of the same app without complicated configuration. Thus, everyone can protect their privacy and still use Twitter as before.

1.3 Outline

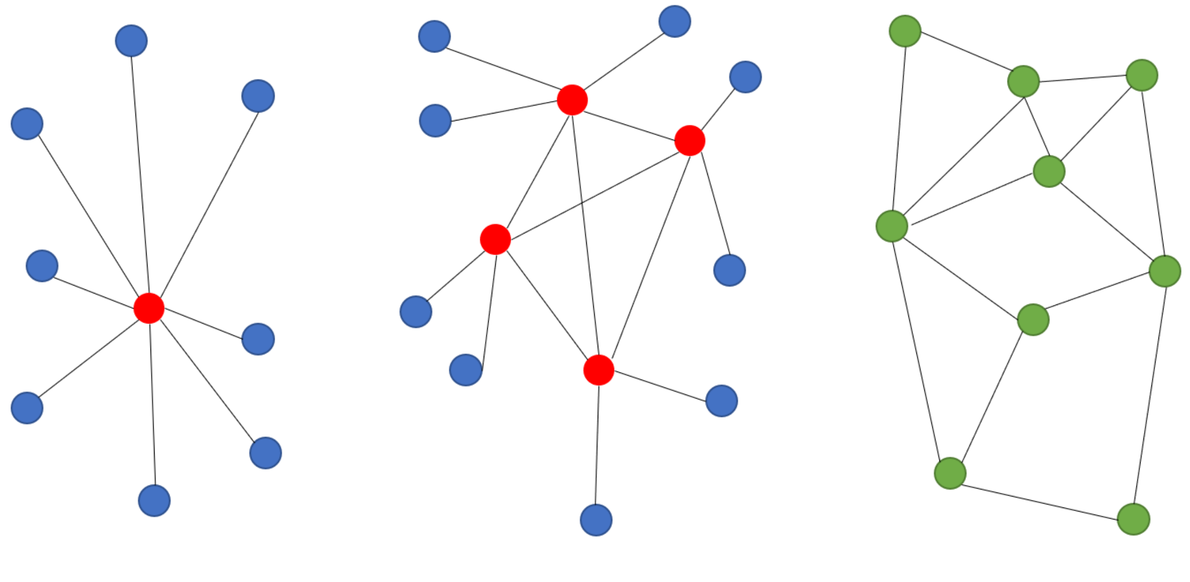
The remainder of this thesis is structured as follows. In Chapter 2, a comprehensive overview of different network and system architectures is given for a better understanding of this work. In particular, the basics of software system architectures and their characteristics are described, as well as P2P networks and dApps. Chapter 3 is about relevant work on privacy protection and projects in the context of this thesis. In Chapter 4, the general concept of a hybrid OSN is discussed, and requirements to the solution are defined. Furthermore, several solution strategies are carried out. Chapter 5 describes our implementation of the concept in the form of an Android app for the hybrid use of Twitter. The design decisions and architecture are considered as well. Chapter 6 evaluates the Hybrid OSN prototype with the previously defined requirements. Besides, the limitations and threats are discussed. Finally, Chapter 7 summarizes this work and gives an outlook for future work.

2 Background

This chapter provides explanations and background information on relevant technologies for a better understanding of this work. This includes the characteristics of different software system architectures and their separation into centralized, decentralized and distributed structures. The basic concepts of P2P networks are also described. Furthermore, the main concepts of dApps and their special features are explained.

2.1 Software System Architecture

The software system architecture describes the relationships and properties of individual software components. It is a model that describes a software on a high-level design. The structure of an architecture can be represented mathematically as a graph, with the nodes representing the individual software components and the edges their relationships to each other. Although the individual components can be executed on the same computer, they are usually interconnected via networks. In general, a distinction is made between centralized, decentralized and distributed architectures as shown in Figure 2.1. In the following, the characteristics and peculiarities of the different architectures are described in detail. [66, 47]



(a) Centralized **(b)** Decentralized **(c)** Distributed

Figure 2.1: Schematic representation of various software system architectures. In centralized applications (a), all clients (blue) are in connection with a central server (red), while in decentralized applications (b) several servers provide for improved stability. In distributed applications (c), all nodes have the same role with the same tasks, hence they are called peers (green).

2.1.1 Centralized Applications

In a centralized application (see Figure 2.1a), the software essentially runs on a central node with a static address with which all other nodes communicate. This central node is called a server and the nodes connected to it are called clients. The clients use the services of the server. Typically, web applications are designed as centralized applications. The clients are usually (mobile) apps or browsers that act as the interface to the user. Examples are social networks such as Facebook and Twitter.

The advantages of a centralized architecture include that new clients can easily join the system simply by connecting to the server. Also, the maintenance of the software is easy to perform, since only the server node needs to be updated. These structures are also advantageous for the speed and the associated user experience since servers are generally reachable via fast connections, have sufficient resources and do not need to use complicated routes across several nodes. Due to the special role of the server, the authentication of a client in the system is easy to perform. Only the server has to check whether the client performs the actions to which it is authorized and can intervene if necessary. If a client is offline, this has no negative impact on the architecture because the software essentially runs on the server. Clients only need to have low resources.

The biggest disadvantage with this architecture is that failure or the blockade of the server leads to a collapses of the entire system. So, the server is the single point of failure of the whole system. Due to the design, the server has all the data. This makes him not only a popular target for hackers but also brings the clients into total dependency on the server. With regard to web platforms like Facebook, having access to all the data can be used to provide an improved user experience by adapting the software to the user's needs. But it can also be used for targeted advertising or sensible data can even get sold to third parties. Furthermore, the server decides which data to send to the client, which brings with it the danger of censorship. As the number of clients increases, the server must scale to have sufficient resources.

2.1.2 Decentralized Applications

What are the advantages and disadvantages of centralized applications, is mostly reversed in decentralized systems. Unlike centralized applications, decentralized applications do not have a single point of failure (see Figure 2.1b). However, there is no node in the system that has all the data which makes accessing information sometimes hard. There are any number of nodes that perform the tasks of a server and by exchanging with other servers in total form a complete system.

Another advantage is that each server only handles the number of users covered by its resources. New servers contribute additional resources to the overall system. Distributed applications cannot be disabled easily because a new server can be started at any time that does not fall under a lock. Because the data are not centrally available, they cannot be processed, so that user data are better protected. Also, there is no longer a prominent attack target. Hence, hacking a single server loses lucrativeness.

The drawbacks include the difficulty of finding data because they are spread across multiple servers. Search functionalities are thus hard to implement. If a server is taken offline, the data are no longer available, even if the system itself remains functional. To tackle this issue, data needs to be replicated. Since there is no longer a central point that is managed by an operator, rolling out updates is difficult. This raises the challenge that server nodes of different versions of the same application can still work together.

Examples of decentralized applications are the social networks diaspora* and Mastodon. In these networks, each user can operate a server on his own. Unlike Facebook, the decision on the existence of the service is therefore not in the control of the operator, but the user.

2.1.3 Distributed Applications

A feature of distributed applications is the computation across all nodes (see Figure 2.1c). At the same time, a single task is executed in parallel on several nodes of a system, and the entire system delivers the result in total. The boundaries between decentralized and distributed networks are blurred. A decentralized network consisting only of servers corresponds quasi to a distributed network. In distributed applications, there is no hierarchy with servers or clients. Each node is equal to the other nodes with everyone performing the same tasks. For this reason, a node in this architecture is called a peer. The structure is then referred to as Peer-to-Peer (P2P). With such structures, there are no scaling problems, since each node contributes the required resources itself. BitTorrent is an application that belongs to this architecture type. It solves the problem of downloading large files efficiently. When downloading a file, it is offered by several nodes so that different pieces can be loaded in parallel from various sources in difference to HTTP where only one source provides the file. Each peer not only downloads files but also provides files to other users. In order to avoid that nodes solely download (so-called Leechers) but also contribute, they are penalized with slow download speeds. This principle is known as *tit for tat*.

2.1.4 Comparison

Table 2.1 compares the main features of the different architectures as described before.

	Centralized Systems	Decentralized Systems	Distributed Systems
Scalability		+	++
Maintenance	++	+	
System Stability		+	++
Performance	++	+	
Data Availability	++	+	

Table 2.1: Comparison of different software system architectures on scalability, maintenance, system stability, performance, and data availability. The pluses indicate how positive something is relative to the other systems.

2.2 Peer-to-Peer Networks

The distinctive feature of P2P systems is that each participant has the role of both a server and a client. The participants are therefore equal and provide each other with services, what is reflected in the naming. P2P networks are usually characterized as overlay networks over the Internet. Concerning the structure of the overlay network, a distinction is made between structured and unstructured networks. The P2P principle became well-known in 1999 with the file-sharing application Napster. The software connected its users and allowed accessing (mainly copyrighted) songs among the participants without having to offer them from a central server. Popular applications of P2P networks are file sharing (e.g., BitTorrent), instant messaging (e.g., Skype) and blockchain technology (e.g., Bitcoin).

Their independence particularly characterizes P2P networks: there are no control points and not necessarily a fixed infrastructure which leads to minimal operating costs. Besides, P2P networks are self-organized and self-scaling, as each additional user contributes its resources. However, there are also some challenges in P2P networks that need to be solved for successful operation. These include finding peers in the network (peer discovery) and finding resources (resource discovery). Especially in file sharing networks, solutions have to be found how to motivate users to upload data and not only use the download one-sidedly. The replication of data and the associated availability must also be taken into account in solutions. Another critical issue is the Internet connection of individual participants, which may not be powerful or permanent. [65, 66, 47]

2.2.1 Unstructured P2P Networks

In unstructured P2P networks there are no specifications for the overlay network, so the peers are only loosely connected. Due to the loose structures, the failure of one peer has no significant influence on the function of the rest of the network. Another advantage of the unstructured topology is the lower vulnerability.

While such loose structures are easy to create, the performance of the entire network suffers. A multicast request is sent to all connected peers, who forward the request again and flood the entire network. If a peer can respond to the request, it responds by the same route that the request used to reach it. Each request has a validity time (Time to Live (TTL)) before it is discarded. Popular files with wide distribution can thus be found quickly. However, rare files are difficult to find because the TTL may has expired before the request could be completed. A flooding search is not efficient and provides a large amount of signaling traffic. An example of this approach is Gnutella.

To address the problem of inefficient and complicated searching for resources, central points are created to answer the search requests. Network implementations using this structure are Napster, FastTrack, and BitTorrent. Figure 2.2 shows a comparison of the search process in the respective networks.

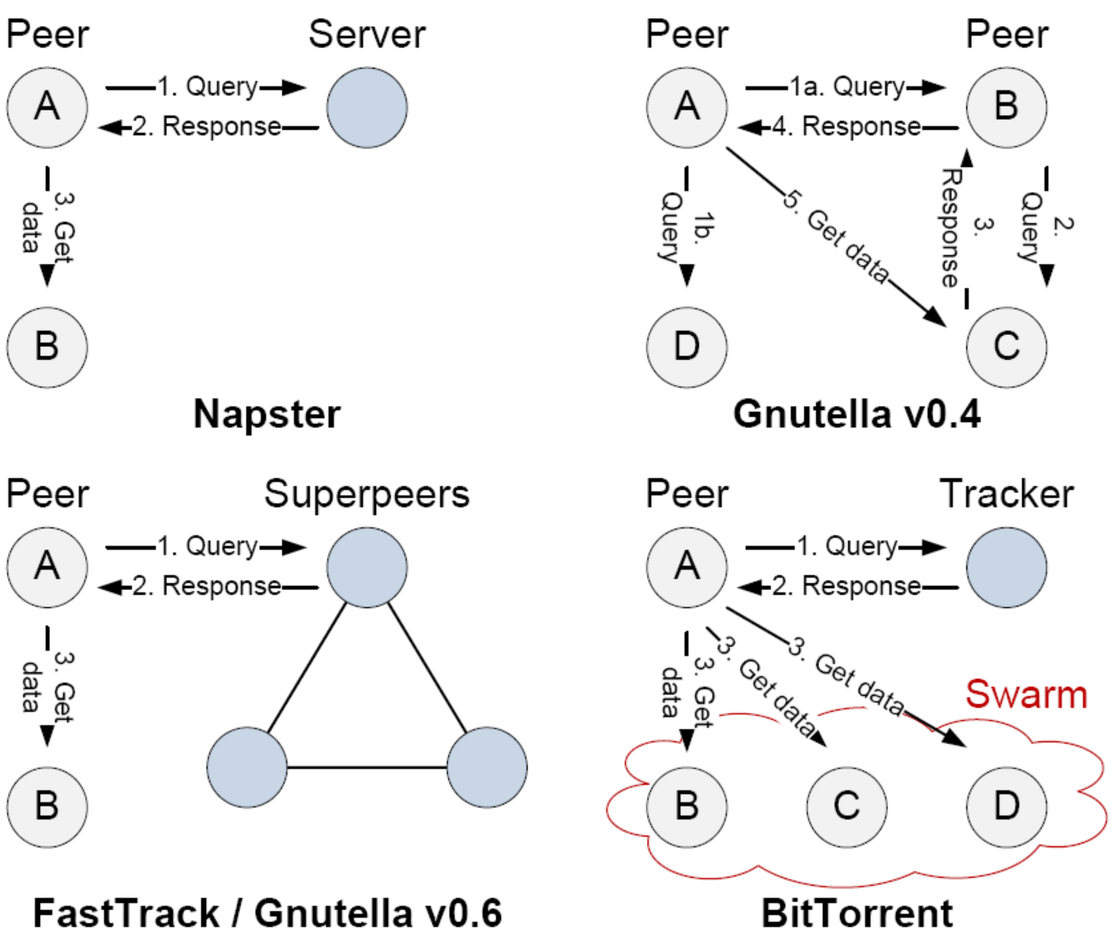


Figure 2.2: File retrieval process in unstructured P2P systems exemplary shown for Napster, Gnutella v0.4, FastTrack/Gnutella v0.6 and BitTorrent [52]

2.2.2 Structured P2P Networks

In structured networks, compliance with the structure is strictly controlled. By defining a particular structure, for example a circle or a tree, search processes in the overlay network can be designed efficiently and deterministically. Routing algorithms determine how a node is arranged in the overlay network. The performance of the entire network depends directly on the arrangement of the nodes and how quick changes (joining and leaving nodes) are detected.

Usually, the routing algorithms are based on a Distributed Hash Table (DHT). Hash tables are data structures in which key-value pairs are stored, whereby the key must be unique. The corresponding value can then be queried via the key. The keys are ids, which are generated with a hash function (e.g., SHA-1). For the addresses of the nodes and the files, ids are created equally, so that they lie in the same address space. For finding a file, it is searched at the node with the same or the next larger id. If it is not available there, it does not exist on the network.

For joining a network, either one or more peers must be known as the entry point, or this information must be obtained from a bootstrap server. When entering a structured network, the joining node is assigned a unique id and thus positions itself in the structure. The routing tables of the nodes affected by the structural change must then be updated. When leaving a network, this happens either planned, and all affected nodes are informed to update their routing tables, or unexpected. Therefore, nodes must always check the correctness of their routing tables.

Known routing algorithms that use DHTs include Chrod[64], CAN[53], Pastry[56], Tapestry[74] and Kademlia[51]. Among other things, they differ in their distinct structure and the hash functions used.

2.3 Web 3.0 - Decentralized Applications (dApps)

The term Web 1.0 refers to the beginnings of the Internet, which consisted of simple static web pages. The central idea was to present or consume content. The characteristic of Web 2.0 is the user's participation in the creation process. Thus, a series of platforms (blogs, social networks) was launched on which users can provide content and connect to each other. Typically, Web 2.0 platforms have a centralized structure, which entails the problems mentioned in the previous chapter. On the occasion of the 30th anniversary of the World Wide Web in March 2019, the initiator Tim Berners-Lee summed up that the Internet was misused - partly due to the system design [28].

With the next version 3.0 of the web, more transparency, security, and fairness should be created. However, while there is broad agreement on what is meant by terms Web 1.0 and Web 2.0, there is no uniform definition of Web 3.0 that has prevailed to date. There are many ideas, but no final solution yet. An interpretation of what Web 3.0 is, is all about decentralization, hence it is also called the Decentralized Web (dWeb). In this context, Web 3.0 is considered an umbrella term for a group of emerging technologies such as blockchain, crypto currencies, and distributed systems which are interconnected to create novel applications, so-called Decentralized Application (dApp). Although decentralized applications have existed for a long time (e.g., BitTorrent), these applications do not meet the criteria of a dApp.

2.3.1 Characteristics of a dApp

As with client-server applications, a dApp is divided into front and back end. The main difference is that the back end is not represented by a centralized server and thus a single point of failure, but by code that is executed in a decentralized P2P network. In the back end so-called smart contracts are used to perform logical operations, and instead of a database, a blockchain is used. The front end can be created either as an app or website, but it is necessary that calls can be executed from the front end to the back end.

Johnston et al. and Siraj Raval name the following four criteria for a dApp [46, 54]:

- **Open Source**: Trust and transparency are created by the disclosure of the source code. This also enables improvement through contributions from other developers.

- **Blockchain based**: The data of the application are stored cryptographically in a public, decentralized blockchain. The immutability of the blockchain can be used in conjunction with smart contracts to build consensus and trust.
- **Cryptographic token**: Certain actions in the network can only be performed by paying with a token. This token can either be purchased or given to the user in exchange for data, storage space or similar. Especially as a reward for positive actions, users should be rewarded with tokens. However, no entity should have control over the majority of the tokens.
- **Token generation**: The application must generate tokens according to a standard cryptographic algorithm (e.g., Proof of Work and Proof of Stake Algorithm) or distribute them among the users at the beginning of its operation.

In addition to the criteria of a dApp, Johnston et al. also describe a classification system for dApps considering if they have their own blockchain or they use another dApp's blockchain [46]:

- **Type 1**: Use their own blockchain (e.g., Bitcoin, Ethereum, EOS).
- **Type 2**: Protocols, to use another blockchain of type 1 with own tokens (e.g., Omni Protocol).
- **Type 3**: Protocols with own tokens, which in turn use protocols of a dApp of type 2.

Over 1900 dApps use the Ethereum platform [14] and in terms of market capitalization, 94 of the top 100 tokens base on Ethereum [17]. Ethereum is a public, decentralized blockchain and at the same time a system for the execution of smart contracts. The associated token is called Ether (ETH) and is required to execute the smart contracts on the Ethereum Virtual Machine (EVM). Since the storage of large amounts of data in the blockchain is expensive, the data are often stored elsewhere, and only one reference is written into the blockchain. IPFS has become the preferred decentralized storage location. Examples of dApps that use the Ethereum - IPFS combination are the social networks Peepeth and AKASHA, which are introduced in the following chapter.

Among the disadvantages of dApps are the costs associated with actions and the handling of tokens. The tokens are stored in a wallet, and this wallet has to be accessible by the dApp. Therefore, a special web browser or an extra browser add-on is required. Before the first usage, the user has to obtain the token somehow. The acquisition is usually complex and therefore represents a barrier for some users to use the application. Because of this, unfortunately, only experienced users can fulfill the prerequisites to use a dApp.

The advantages of a dApp include not only reliability but also the avoidance of censorship and manipulation by design. Furthermore, there are no dependencies towards a service provider. However, the creation of a dApp is complex and it is difficult to install updates and bugfixes. Currently, there are still very few dApps, so interactions between different dApps are not able to create synergy effects.

3 Related Work

This chapter gives a comprehensive overview of different projects trying to protect the users' personal data in OSNs. Hereby, six different approaches are presented. First, extensions are considered which increase the security of established social networks. Then, alternative social networks will be presented that have placed the protection of personal data at the center. Furthermore, two next-generation social networks will be considered, which take advantage of the blockchain technology and belong to the group of dApps. Finally, the ActivityPub protocol is presented, which maps the communication in decentralized platforms. The chapter concludes with a summary of the related work.

3.1 Privacy through Extensions

Existing connections to other people and already created content can bind users to platforms. This so-called lock-in effect prevents users from switching to another platform. If switching to another platform is not an option, how can the use of the platforms be made more secure and user data better protected? In the following, two approaches (Twitterize and FaceCloak) are presented, which try to increase the privacy and anonymity on Facebook and Twitter.

3.1.1 Twitterize

With Twitterize, Daubert et al. present an approach on how a particular overlay network can protect data and anonymously exchanged within a social network [33]. The proposal refers to micro-blogging social networks like Twitter. As proof of concept, they created an Android app.

Design Principles

Daubert et al. stated various demands on the proposed solution. Concerning the protection of privacy in general:

- **Confidentiality**: Messages exchanged between sender and receiver should be transmitted secretly.
- **Anonymity**: A individual user should not be identifiable within a set of users (anonymity set).

Also, concerning the design of the implementation:

- **Understandable privacy**: The user should decide for himself which level of privacy protection he chooses for his messages. The system should reasonably communicate this.
- **Simple group management**: Group memberships should be managed quickly and easily.
- **Low overhead**: The client should have the full functionality of the original client and have as little overhead as possible for the extended functionality.

Architecture

The basic idea behind Twitterize is the encrypted, anonymous exchange of messages within a group via an overlay network. There is a group-specific hashtag for this. The messages are forwarded in the underlay network by users until they reach the recipient.

In order to establish communication within a group, the three following phases have to be passed through one after the other:

1. Generation and exchange of the keys of a hashtag
2. Flooding and subscription to build an overlay network
3. Exchange of messages

First, a user has to define the hashtag and generate a key for symmetric encryption. This key is used later to encrypt the messages as well as the hashtag itself. From the encrypted hashtag, only a hash value is used (so-called pseudonym) so that the actual hashtag stays private. In order to join the group, other users need the key and knowledge of the name of the hashtag. However, the key exchange should not be carried out via the social network. An exchange via e-mail, Near Field Communication (NFC) or QR codes is conceivable.

In second phase, the overlay network is constructed. Here, the private flooding mechanism is used [32]. A publisher creates an advertisement tweet consisting of a pseudonym and the first element of a hash chain and posts it in the underlay network. This tweet appears in the timelines of his followers. Then, each follower distributes the advertisement tweet on his timeline and thus reaches his followers. However, this tweet differs from the original tweet by extending the hash chain. It generates a hash of the previous hash chain and thus receives the next element of the chain. According to this principle, the entire network is flooded, and as a result, each user saves a triplet consisting of the pseudonym, the hash chain and the user previous to him in the chain in a database table.

If the advertisement tweet reaches a user, who is aware of the hashtag and the associated key, the user (so-called subscriber) responds with a subscription tweet. This subscription message is addressed to the user from whom the advertisement tweet was received and contains the user name (@user) and the pseudonym. The user thus addressed in turn posts a message consisting of the user name of the user from whom he received the advertisement tweet and the pseudonym. This happens until the subscription message reaches the original publisher. In this process, the pseudonym and the sending user are stored in another routing table. In this way, the overlay network is formed. Since each user only has a local view of the message flow, no conclusions about the sender and recipient are possible.

In the third phase, users can exchange messages using the previously shared hashtag. The messages are encrypted with the key belonging to the hashtag and then posted together with the hashtag to the timeline. By posting the message again, forwarders ensure that the message is forwarded to the recipient. The exact procedure is shown schematically in Figure 3.1.

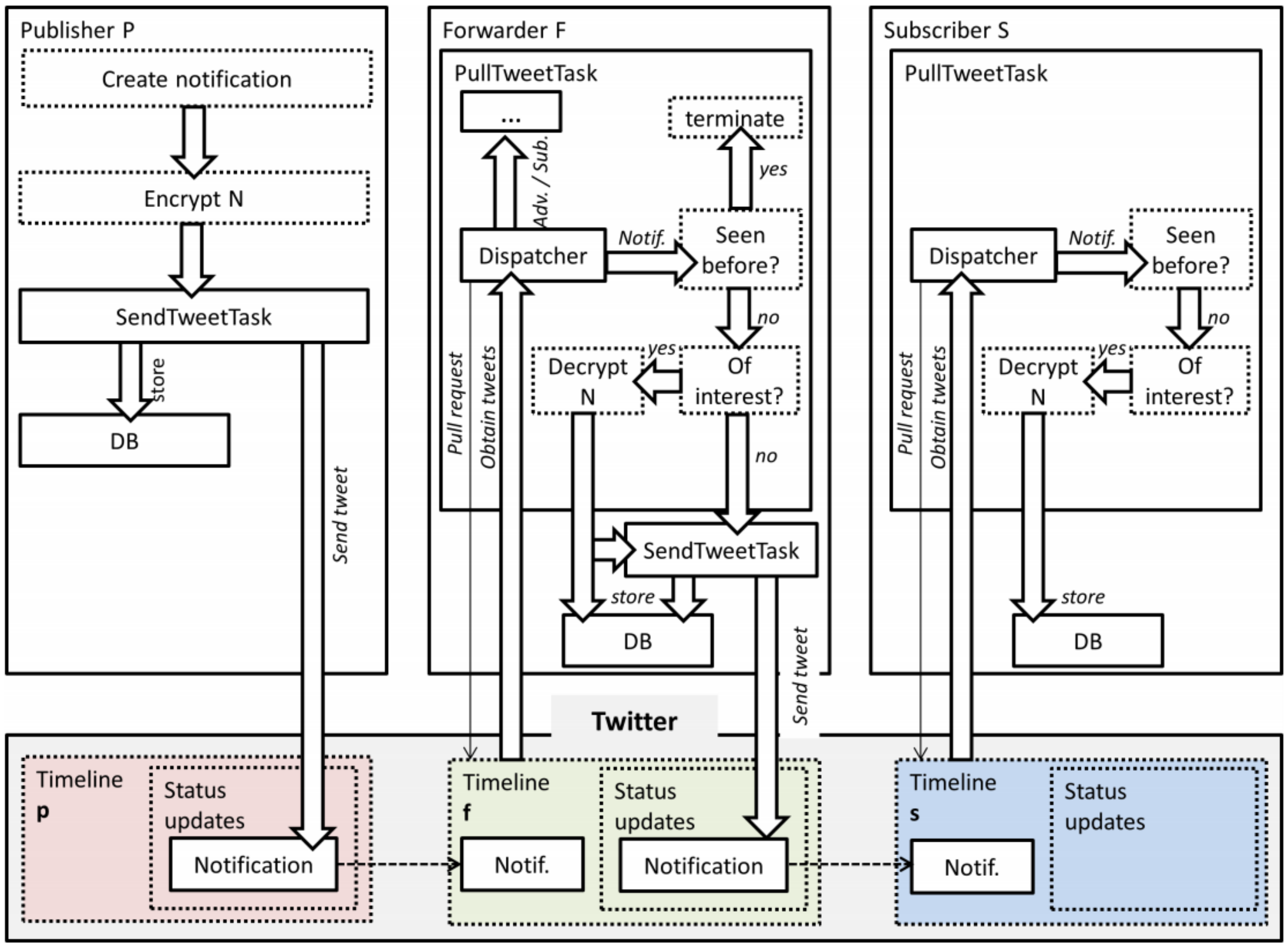


Figure 3.1: Information flow of a notification in Twitterize. The notification is created, encrypted and posted to the publisher's timeline. The notification is then picked up by a forwarder who follows the publisher. Next, the status update is being processed and finally posted on the forwarder's timeline. Lastly, the subscriber who follows the forwarder receives the notification. [33]

Proof of Concept

As proof of concept, Daubert et al. implemented Twitterize as an app for Android. The app was programmed in Java, the code remained closed source, and the app is not available from the Google Play Store. The representation of the data in the app takes place on different timelines.

In addition to the standard home and user timelines, each hashtag gets its timeline which is accessible via a tab.

For encryption 128bit keys with Advanced Encryption Standard (AES) Cipher Block Chaining Mode (CBC) is used. The app provides support for the key exchange between the users via NFC or QR codes. Since data structures, such as JavaScript Object Notation (JSON), are too verbose, the tweets are encoded using the Base64 algorithm. To distinguish between message types, rarely used UTF-8 symbols are utilized to give the messages a rough structure. The formation of the overlay network and the exchange of messages then works as described above.

The design of the Twitterize architecture meets the privacy requirements. The other requirements for implementation were also achieved during the creation of the app. With the exchange of keys via NFC or QR codes an easy way for key management was found and implemented. Just a few seconds of physical proximity are enough to form a group.

The avoidance of overheads was also successful, although the timeline is updated in the background every minute. Central Processing Unit (CPU) usage and power consumption were slightly above the values of the original Twitter app. For the storage of the additional data, only a little space is necessary, since each hashtag occupies just 48 bytes. Assuming that in the Twitter Social Graph two arbitrary users are connected via 4.71 following users in between, an average delivery time of 142 seconds was calculated for a message.

Restrictions arise due to limits on the use of the Twitter API and because the application must always be online to get the best user experience.

3.1.2 FaceCloak

Researchers Luo, Xiu, and Hengartner of the University of Waterloo in Ontario propose an architecture to protect personal information on social networking platforms [49]. Protection is achieved by transmitting fake data to the social network server and storing the real data encrypted on a third party server. Authorized users can then replace the fake data with the correct data when they visit the site containing protected data. The prerequisite is that all users use a specific browser extension that communicates with the third party server and replaces content. In concrete terms, this was implemented for Facebook and both a server and an extension for the Firefox browser were created and successfully tested.

Design Principles

FaceCloak's design is based on the following four principles:

- **Preservation of normal browsing experience**: In order to provide the best possible user experience, the solution should function largely automatically and require only a minimum of interaction.
- **No server-side changes**: The mechanism for protecting personal data should not require any server-side changes.

- **Self-containment and minimal user configuration**: Regardless of the technical abilities of a user, the configuration effort (e.g. the installation of a certain software) should be limited to a minimum and be feasible by everyone.
- **Incremental deployment**: Compatibility between users with and without using the special extension should always be ensured and should never prevent users from no longer being able to contact each other.

FaceCloak Architecture

After validating several available solutions for personal data protection, the researchers decided that a client-side architecture was the best solution for automatic protection. Figure 3.2 shows this architecture schematically.

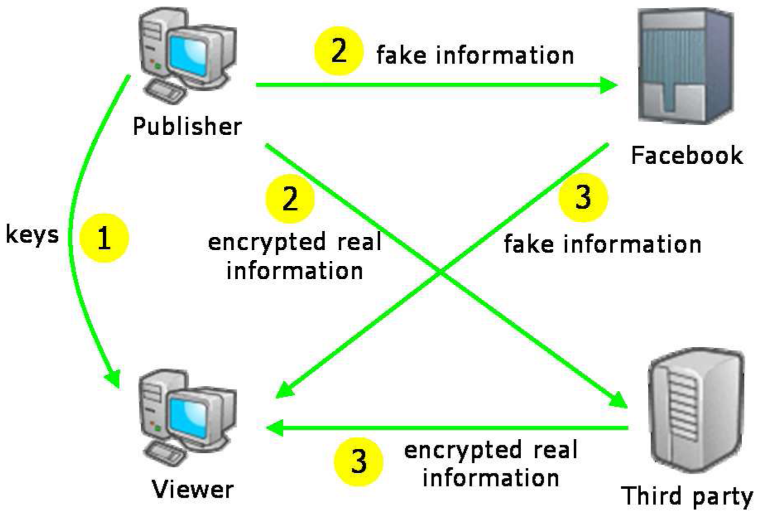


Figure 3.2: Schematic representation of the Setup Phase (1), Encryption Phase (2) and Decryption Phase (3) and the data flow taking place between the entities in FaceCloak's architecture. [49]

During the setup phase, the browser extension is installed, and the encryption keys are generated. Afterwards, the keys for decryption are shared with the trusted contacts. In phase two, when data worthy of protection is stored, it is transmitted in encrypted form to a third party server and stored there. Only fake data are transmitted to the social network server. In phase three, whenever an authorized contact calls up a profile page and fake data are transmitted by the social network, the extension takes care of the replacement with the real data.

In addition to adhering to the above design principles, the proposed architecture makes the following contributions:

- The functionality of the service and the interface is not limited by the use of FaceCloak.
- The user decides which information should be protected and which not.
- The architecture can be applied to any social network.

FaceCloak for Facebook

To protect the privacy of Facebook users, Luo, Xiu, and Hengartner have created a Firefox browser extension according to the previously described architecture, as well as a server application for storing encrypted real data [50]. The extension uses AES and a key length of 128 bits to encrypt the data. The indices for the encrypted data are calculated using SHA-1. The authors propose an e-mail for the key exchange. For this purpose, the browser extension automatically generates e-mail texts and recipient lists and forwards them to the standard e-mail client. The recipients then have to store the received keys in the extension manually.

In order to protect data with FaceCloak, the prefix @@ must be added to the information in a text field. For other form elements such as dropdowns, radio buttons or checkboxes, the extension creates additional options that also start with @@. When submitting the form, the extension intervenes and replaces the data marked with @@ with fake data. The data to be protected are encrypted with the stored keys and transferred as a key-value pair to the third party server where it is stored. FaceCloak can protect all profile information, but only for name, birthday, and gender algorithms for the meaningful creation of fake data are implemented. In addition to profile information, the extension can also protect Facebook Wall and Facebook Notes data. The contents of arbitrary Wikipedia articles are transmitted as fake data to avoid attracting attention with random and unusual character strings.

When loading a profile page that contains protected data, the extension with asynchronous HTTP requests retrieves the information from the third party server, decrypts it, and replaces the fake data. A large part of the replacement can thus be performed during the load process so that the user does not see the fake data. However, since Facebook also loads content asynchronously, some replacements can only be performed with a time delay and the fake data are shortly visible.

The keys have to be transferred to all devices and stored in the extension to use the same account. It is not possible to use multiple accounts with the same Firefox profile, as all data are stored in the extension and these are always bound to exactly one Facebook account.

The latest version 0.6 from August 2010 cannot be installed in the current Firefox (version 65). Furthermore, it is unknown if the server is still running. Therefore it is not possible to check if the extension still works. Due to the numerous updates and sometimes severe changes that Facebook has experienced in the last eight years, it is doubtful that the extension will still function today. At that time, however, it was successfully applied and proved that the proposed architecture worked.

3.2 Privacy-protecting Social Networks

In the business models of the large, popular OSNs, user data plays an essential role. The data are evaluated and used to make a profit, for example through personalized advertising. Anonymity and the protection of privacy are not among the overriding objectives.

In the following, two social networks, diaspora* and LifeSocial.KOM are presented, which have placed the protection of data at the center.

3.2.1 diaspora*

Inspired by a lecture on surveillance in centralized social networks by Eben Moglen on 5th February 2010, the four mathematics students from New York University Grippi, Salzberg, Sofaer and Zhitomirskiy had the idea for diaspora* [60]. Diaspora* is a decentralized social network with the following special features:

- **Decentralization**: Everyone can start their server with the diaspora* software and be part of the network. Alternatively, there are public servers accepting registrations from everyone, so there is no need to set up an own instance of diaspora* to participate in the network.
- **Privacy**: By running a server, the data remains with the user. Furthermore, it is possible to determine which users can see content.
- **Open Source**: The source code is disclosed and hosted on GitHub[1]. The transparency created in this way ensures trust in diaspora*.

For funding the development of diaspora*, $ 10 000 should be crowdfunded on Kickstarter[2]. The project was very well received so that after 14 days the target was reached [61] and in the end, a total of $ 200 641 was donated. In November 2010, the first alpha version of diaspora* was released. One year later there was a prominent feature update. In May 2012 it was announced that the diaspora* development should be continued within the Y Combinator start-up program[18]. Due to the commercial influence, there were fears that diaspora* could lose its independence. In August 2012, the developers announced that diaspora* is henceforth a community project [31].

The diaspora* back end is written in Ruby, the front end to the user is a website. A server running diaspora* is called pod. Each pod has its own domain, so users of a pod have a username similar to an e-mail address (for example, username@podname.org). Diaspora* has the typical functionalities of a social network (hashtags, @ mentions, likes, comments, private messages). What marked a peculiarity at the time of diaspora*'s appearance are so-called aspects. Aspects are groupings of contacts that can be specified as a target audience when posting content. Only the contacts associated with the aspect can see the post.

For staying in contact with friends on other platforms like social networks (Facebook, Twitter) or blogs (Tumblr, Wordpress), the initial idea was to connect these platforms. Data exchange should work both ways. Posts published on diaspora* should also appear on other platforms at the same time. Also, posts from the other networks should be viewed in diaspora*. Diaspora* should play the role of a social media hub. Unfortunately, the APIs of some platforms have become increasingly limited as instances of misuse of the interfaces have become public.

[1] https://github.com/diaspora/diaspora
[2] https://www.kickstarter.com/projects/mbs348/diaspora-the-personally-controlled-do-it-all-distr/description

The data of the users are stored unencrypted on a pod so that someone having access to the database can see them [3]. In order to protect his own data in the best possible way, the operation of a separate diaspora* instance is necessary. The communication between the pods is encrypted with Secure Sockets Layer (SSL) protocol [3]. Furthermore, the exchanged messages are first signed (Salmon Magic Signatures), then symmetrically encrypted with AES-256-CBC [7]. The AES key is encrypted with the public key of the recipient and sent together with the encrypted message.

Diaspora* does not use the ActivityPub protocol (see Chapter 3.4.1), but its own diaspora* federation protocol[2]. Other platforms such as Friendica[3], Hubzilla[4] or Socialhome[5] can also communicate via the diaspora* federation protocol. There is no official API, which makes app development difficult. Diaspora* points out that the website is also usable on mobile devices, so there is no need for a native application [3].

According to the statistics of the-federation.info on 24^{th} February 2019, 679 723 users were registered on a total of 251 pods. Over the last 12 months, 19 591 new users have joined the network. In January 2019, only 30 042 (4.4 %) users were active. However, the numbers are incomplete, as some pods do not share information and there may be more than the 251 listed pods. [15]

3.2.2 LifeSocial.KOM

Kalman Graffi and his team researched a P2P based social network in the Multimedia Communications Lab at Technische Universität Darmstadt. From the service provider's point of view, the aim was to save operating costs, and from the user's point of view to ensure secure communication. Concerning the scope of functions, the P2P OSN should not be inferior to the famous, centralized OSNs.

First, research on basic topics was necessary before creating the P2P OSN LifeSocial. In the following, the framework and its data structure [41], the implementation of the security requirements [40] and the monitoring of the network [42] are presented. Finally, the presentation of the LifeSocial prototype [39] follows.

Platform and Architecture

Initially, Graffi et al. designed a framework for P2P-based multimedia online communities. The framework uses a multi-layer architecture where the respective layers communicate only with their direct subordinate and superordinate layers (see Figure 3.3). To create a structured P2P network, FreePastry[6] was used, which implements Pastry [56]. The PAST extension is used for the storage and replication of data objects. For PAST, additional functionality for deleting entries in the DHT by overwriting them with empty objects has been added.

[3] https://friendi.ca/
[4] https://zotlabs.org/page/hubzilla/hubzilla-project
[5] https://socialhome.network/
[6] http://www.freepastry.org/FreePastry/

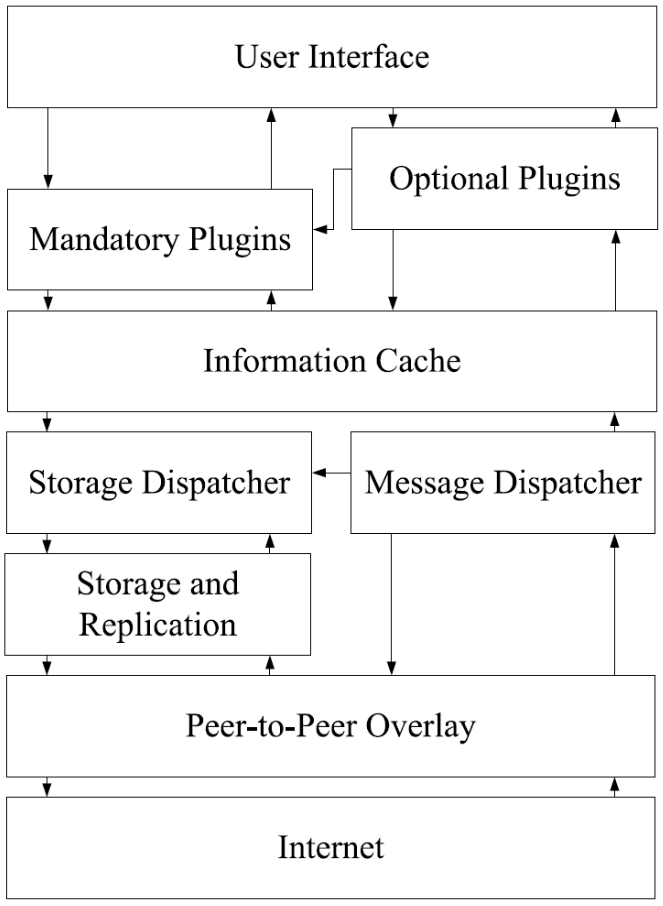


Figure 3.3: Layered architecture as used in the framework for P2P based social networks by Graffi et al. [41]

The task of the Storage Dispatcher is to perform the store, get and delete operations. If a peer is unreachable, the storage dispatcher will determine this and start a new storage job. The Message Dispatcher provides a service for higher layers to exchange information from plugin to plugin. It is used to send and receive messages.

The Information Cache works like a local DHT. For all requested object ids there is an entry consisting of timestamp and state (in cache, requested or not available). Thus, higher layers get a fast response and are not blocked by waiting for the data. Missing data objects are requested periodically by the Information Cache until they are retrieved.

The layers introduced so far provide the basis for the plugin layer. Plugins are small building blocks that contribute a specific functionality and have an interface for interaction with the user. A distinction is made between mandatory and optional plugins. Mandatory plugins must be available on every peer while the user freely chooses optional plugins. Plugins can be dynamically loaded into the system at runtime and represent a simple extension option for the OSN's functionality. The User Interface as the top layer represents the entirety of all plugin interfaces of the OSN.

The framework is implemented in Java, and the OSGi framework was chosen to load plugins at runtime.

Data Structure

In order to store the data effectively, Graffi introduced the Distributed Linked List data type, which meets the complex requirements of a P2P OSN and is storable in a DHT. Each storage object has a unique id for identification that consists of a plugin id and other plugin-specific information (see Figure 3.4). For example, the id of a data object about user Alice photo albums is obtained as a hash from `photoplugin:albumsof+user:alice`. The data object also contains pointers to other objects (photo album to individual images) or data (metadata or binary data).

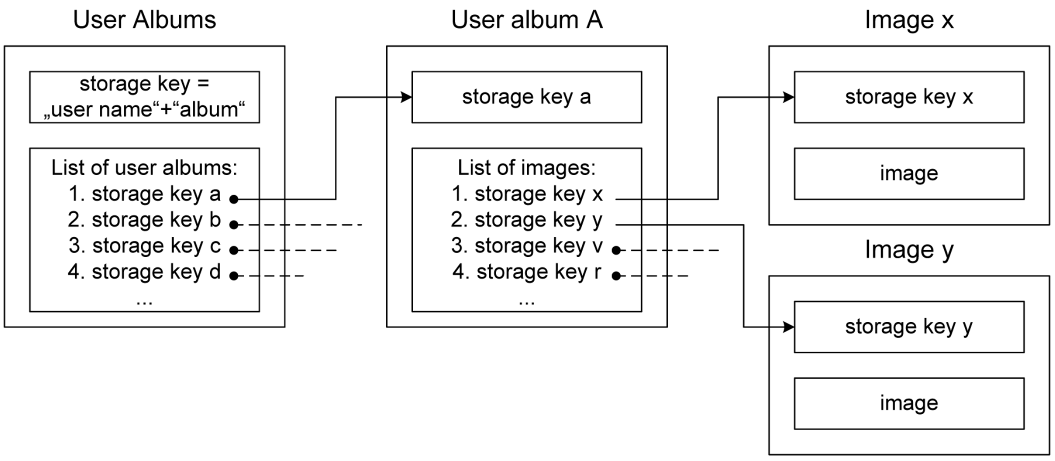


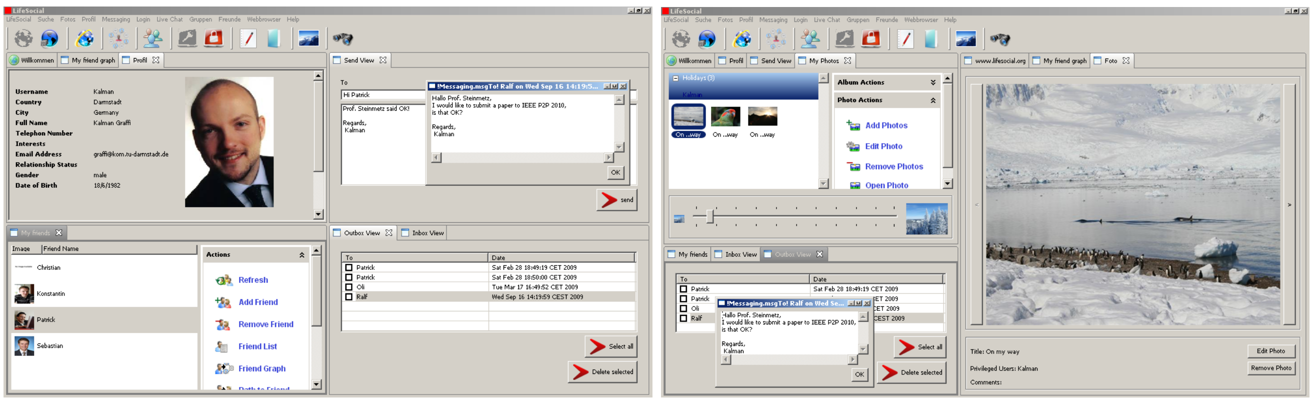
Figure 3.4: An example for the use of the data type Distributed Linked List. The user albums object on the left points to several album data objects which again point to the image objects. Each object has a unique id and a body to store data. [41]

Security

When registering, a new user enters a user name and password. The password is used to generate a key pair for asymmetric encryption. The public key serves as user id and node id at the same time. All data are encrypted for secure and authenticated communication. A new symmetric key is generated for each data object (so-called `SharedItem`) and used to encrypt the data. The symmetric key is then encrypted asymmetrically with the recipient's public keys and added to the shared item to an Access Control List (ACL). Users who are not in the ACL cannot decrypt the symmetric keys and therefore have no access to the data. Finally, the shared item is signed by the author. Signed `SharedItems` are called `CryptedItems`.

Prototype

LifeSocial is based on the previously described framework. The software runs under Windows, Linux, and MacOS. For mobile devices, there is no client app. Figure 3.5 shows two screenshots of the LifeSocial Graphical User Interface (GUI). The code of LifeSocial is not publicly accessible (closed source).



Username	Kalman
Country	Darmstadt
City	Germany
Full Name	Kalman Graffi
Telephon Number	
Interests	
Email Address	graffi@kom.tu-darmstadt.de
Relationship Status	
Gender	male
Date of Birth	18/6/1982

Image	Friend Name
	Christian
	Konstantin
	Patrick
	Sebastian

To

Hi Patrick

Prof. Steinmetz said OK!

Regards,
Kalman

!Messaging.msgTo! Ralf on Wed Sep 16 14:19:5...

Hallo Prof. Steinmetz,
I would like to submit a paper to IEEE P2P 2010,
is that OK?

Regards,
Kalman

To	Date
Patrick	Sat Feb 28 18:49:19 CET 2009
Patrick	Sat Feb 28 18:50:00 CET 2009
Oli	Tue Mar 17 16:49:52 CET 2009
Ralf	Wed Sep 16 14:19:59 CEST 2009

Title: On my way

Privileged Users: Kalman

Comments:

Figure 3.5: Screenshots of the LifeSocial GUI showing multiple plugins for profile, friends, message system and photos [39]

All functionalities (login, profile, friends, groups, and more) are designed as plugins to load them dynamically. Besides the usual social media functionalities, there are also plugins for file exchange, games (Tic Tac Toe) or a whiteboard. A separate store is planned, where users can download additional plugins.

Graffi moved to the University of Paderborn in 2011 and has been a professor at the University of Düsseldorf since 2012 [6]. LifeSocial has since been renamed to LibreSocial and has been looking for participants for an alpha test on their website[7] since 2016. The screenshots on the LibreSocial website show a completely overhauled user interface with a web front end. Since the project website was not changed since May 2016 and there were no further publications, the current status is unclear.

3.3 dApps - The Next Generation Social Networks

As mentioned in Chapter 2.3, dApps use emerging technologies, like the blockchain and cryptocurrencies, to build decentralized applications. With Peepeth and AKASHA, two dApp OSNs are presented in this chapter. They both use Ethereum as blockchain and IPFS to store data.

3.3.1 Akasha

In early 2015, Mihai Alisie (co-founder of Ethereum) had the idea for AKASHA. AKASHA is a social network that differs from other known social networks mainly in its decentralization and use of the blockchain. The absence of a central server meant that censorship was ruled out by design. This is realized by the two technologies Ethereum and the InterPlanetary File System (IPFS). Electron, React, Redux, and NodeJS complete the technology stack so that the primary programming language is JavaScript [22]. In addition to Mihai Alisie, ten other employees work at AKASHA. Furthermore, the founders of Ethereum (Vitalik Buterin) and IPFS (Juan Benet) advise the project [21].

7 https://libresocial.com/en/startpage/

Alisie sees AKASHA as „the missing puzzle piece that will enable us to tackle two of the most critical challenges we face today as a modern information-based society: freedom of expression and creative perpetuity“[22]. The central goal is therefore to prevent censorship and to obtain information over a long period.

AKASHA offers the typical functionalities known from other social networks. These include creating a profile and connect with other profiles. Posting content and commenting on other entries. Furthermore, there is a message system and the possibility to tip other users. Instead of a central server, AKASHA uses the Ethereum testnet Rinkeby and IPFS to store data, so AKASHA can be called a dApp. The messaging system is implemented via Whisper. ETH is required to execute all actions in order to write to the blockchain, but this can be easily requested in the test network.

After a proof of concept had validated the idea, the technology stack mentioned above was defined, and development began at the beginning of 2016. The first goal was to create a client based on Electron for Windows, Linux, and MacOS. In January 2017, the first functional alpha version was completed and tested with a closed circle of users. Over time, additional functions were added, bugs fixed, and performance optimized. In November 2017, a web version of AKASHA[8] was introduced. This was a big step towards a better user experience since the web client does not need to download the Ethereum Rinkeby blockchain which took around 30 minutes on the first run. But, the web version only works in browsers running MetaMask and IPFS Companion extensions. The public beta phase started in February 2018 with the primary goal to see how the application behaves under heavy load. [22, 23]

With the announcement of the web version, the team behind AKASHA also introduced their own AETH token. This token has the unique feature that it can take different states. By locking AETH, the user receives *Mana* which regenerates every day as long as the AETH remains locked. Performing interactions, for example liking, commenting or publishing posts, consumes *Mana* and ETH, but not AETH. When *Mana* is used to vote for something, the content's author receives the same amount as *Essence* which can be converted into new AETH tokens. The third state is called *Karma* and functions as a all time score of received *Essence* even though it has been used to mint AETH. *Karma* is necessary to unlock new features inside AKASHA. [23]

After a long period of silence, the entire project was converted in January 2019, almost three years after the start. The domain changed from akasha.world to akasha.org, and the focus shifted from a social network to an umbrella organization that unites several projects. In the AKASHA blog, Alisie writes about metamorphosis and compares the change from a caterpillar to a butterfly [24]. The alpha and beta phases are said to have corresponded to the caterpillar phase, the second half of 2018 to the chrysalis stage, and now the butterfly is supposed to unfold with all its beauty. However, it is left open how the new orientation will look like and how the social network will continue. On the website, there is a software section as well as a hardware section. But, there is no content available yet.

The public launch of the social network in version 1.0 was planned for the fourth quarter of 2018. With this launch, the change from the Rinkeby test network to the Ethereum main net-

[8] https://beta.akasha.world/

work should also be completed [23]. It is currently unknown when the public launch will take place.

3.3.2 Peepeth

Bevan Barton created Peepeth and launched the platform in March 2018. Peepeth[9] is a microblogging platform that is very similar to Twitter in functionality and design. There are also other parallels to Twitter: Instead of a blue bird in the logo, Peepeth uses a penguin and instead of *tweeting* users *peep*. The maximum post length is limited to 280 characters, which has no technical cause but was taken over from Twitter. The main difference to Twitter is the decentralization.

From Peepeth's point of view, social media is broken. The major problem is, that OSN service providers control the online identities of users, sell their data, and violate their privacy. The news feeds are manipulated to drive the user to a higher level of interaction at any price. Besides, the platforms are teeming with trolls, bullying and flame wars. Barton wants to counter these grievances. Therefore there is no advertising on Peepeth and no use of the user data for any purpose.

The website Peepeth.com is the front end of a dApp, which uses the Eteherum blockchain and IPFS. This front end can theoretically be exchanged arbitrarily, and Peepeth's data can be read and written because of the open blockchain protocol. No Ethereum test network is used, but the main network. The execution of transactions on the Ethereum blockchain is associated with costs. Peepeth pays the fee for its users. The necessary capital was collected via a crowdfunding campaign. However, when accounts distribute spam, Peepeth no longer carries the price for writing to the blockchain. The resulting charge should make spamming unattractive and reduce it to a minimum. Although Peepeth covers the costs, the user has to sign the transactions. Therefore, Peepeth requires a dApp browser (e.g., Opera) or a browser that has been extended by a wallet (e.g., using MetaMask browser extension). [25, 30, 10]

In order to keep transaction fees low, the actions executed on Peepeth are collected on the server hosting the front end and written to the blockchain in batches every hour. Several actions are bundled in one file and transaction. The actual contents are written as JSON file to IPFS and only the reference hash is stored on the blockchain. [30]

While the smart contracts are open source, the front end is closed source. So it is impossible to understand what is happening on the server hosting the front end Peepeth.com. Image files are not only stored in IPFS but also mirrored at Amazon Web Services (AWS) to provide a better user experience. The client does not communicate directly with IPFS, but the server behind the front end communicates with the two back end technologies IPFS and Ethereum blockchain, as shown in Figure 3.6.

The data written to the Ethereum blockchain cannot be deleted or modified. For Barton, this immutability is an advantage since everyone is forced to be aware of his actions and the self-confidence for his actions is sharpened [8]. Furthermore, this fact of immutability is the main argument for freedom of expression and against censorship. However, not all messages are

[9] https://peepeth.com/welcome

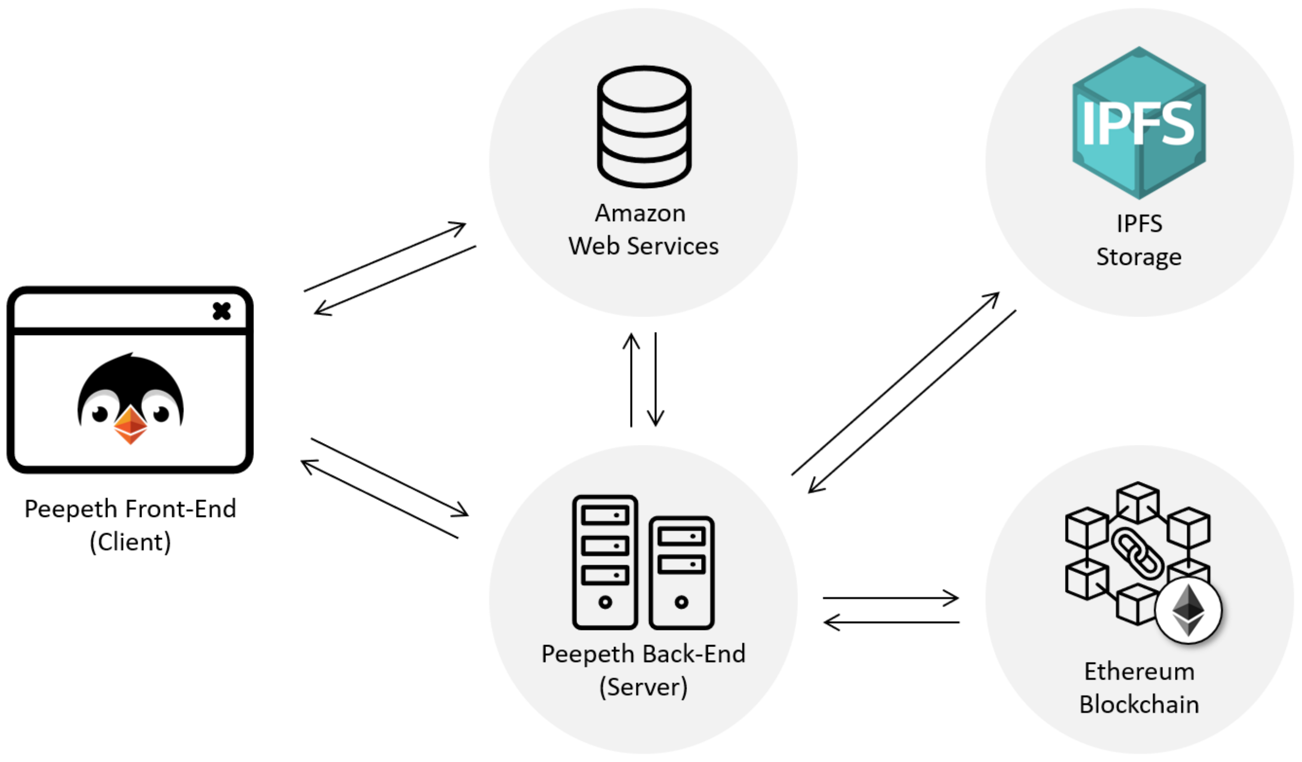


Figure 3.6: System architecture of Peepeth. The front end website communicates with a back end server and loads images from AWS. The back end is connected to the Ethereum blockchain and IPFS.

presented in the Peepeth front end. If they violate Peepeth's terms of use, a filter will sort them out. Peepeth calls this procedure „moderation“ and argues that this is by no means to be understood as censorship, but much more „on cultivating mindful engagement and positive contribution“ [11].

In addition to writing short messages, it is possible to like posts. However, there is only one like per day available, a so-called *Ensō*. „Ensō (Japanese for ‚circle‘) is a circle that is hand-drawn in one uninhibited brushstroke. It represents creativity, freedom of expression, and unity“ [8]. The resulting rarity should express the particular appreciation of a contribution. Furthermore, good content from other users can be rewarded with a tip. 10 % of the tip are taken to finance Peepeth. [8]

On 29th January 2019, Peepeth had 4 055 users who posted a total of 66 262 Peeps [12]. To get access, potential users have to apply first and receive a sign-up link by email to join the platform after some time. Upon the invitation of an active user, new users can join directly without waiting time. Users can verify themselves with their existing Github and Twitter accounts. In the future, it will be possible to use further platforms for the verification of an account. To verify an account, the user must post a „special message“, which also contains his Ethereum address. The link to this post must then be handed over to a smart contract, which confirms the ownership of the account.

The next milestone of Peepeth is to increase the user experience as it was communicated in the crowdfunding campaign. The first milestone has already been reached, which was the coverage of the users' fees for writing to the blockchain. Therefore, the complexity was massively decreased since no user has to buy ETH first. The next steps are the use without special software requirements (renouncement of particular browsers or MetaMask) and the development of an iOS app. However, only 177 bakers donated 140.56 ETH from the required 1 000 ETH. It is unclear to what extent the desired goals will now be achieved. [9]

3.4 Protocols

This part is about protocols that describe communication in decentralized applications. With the rise of the decentralized OSN Mastodon, the ActivityPub protocol became popular which is introduced in the following.

3.4.1 ActivityPub

ActivityPub is a protocol published by the World Wide Web Consortium (W3C) in January 2018 as an official standard [73]. The protocol regulates communication within an open, decentralized social network. There are two levels: client to server (Social API) and server to server (Federation Protocol). The two protocols are designed in such a way that they can be used independently of each other. If one of them is implemented, it is easy to implement the other. The Activity Streams [45] data format is used to describe activities in JavaScript Object Notation for Linked Data (JSON-LD) format. This data format is also an official W3C standard with the aim to record meta data of an action in a human-friendly but machine-processable syntax.

The principle behind ActivityPub is similar to that of e-mail. Servers can be uniquely identified via the domain. Within a server, each mailbox is accessible via a unique name. Thus, users can communicate with each other via different servers by having their messages forwarded to their mailbox.

Client to Server (Social API)

Users are called actors in ActivityPub and are represented by an associated account on the server. Since there can be several servers, it is important to emphasize that an account is bound to one server and user names must always be unique only within a server. Each actor has an inbox and an outbox on the server on which he is registered with his account. These are the two endpoints with which the client application communicates via HTTP requests. More is not necessary, because the server ensures that all messages and information for the user end up in his inbox and that the messages in his outbox are forwarded to the desired recipients (see 3.4.1). For ensuring that only authorized clients store content in an outbox, the sender must sign the posts.

The outbox of an actor holds all his published posts. When accessing the an actor's outbox without authorization, the server delivers all public posts of the actor. If access with authorization

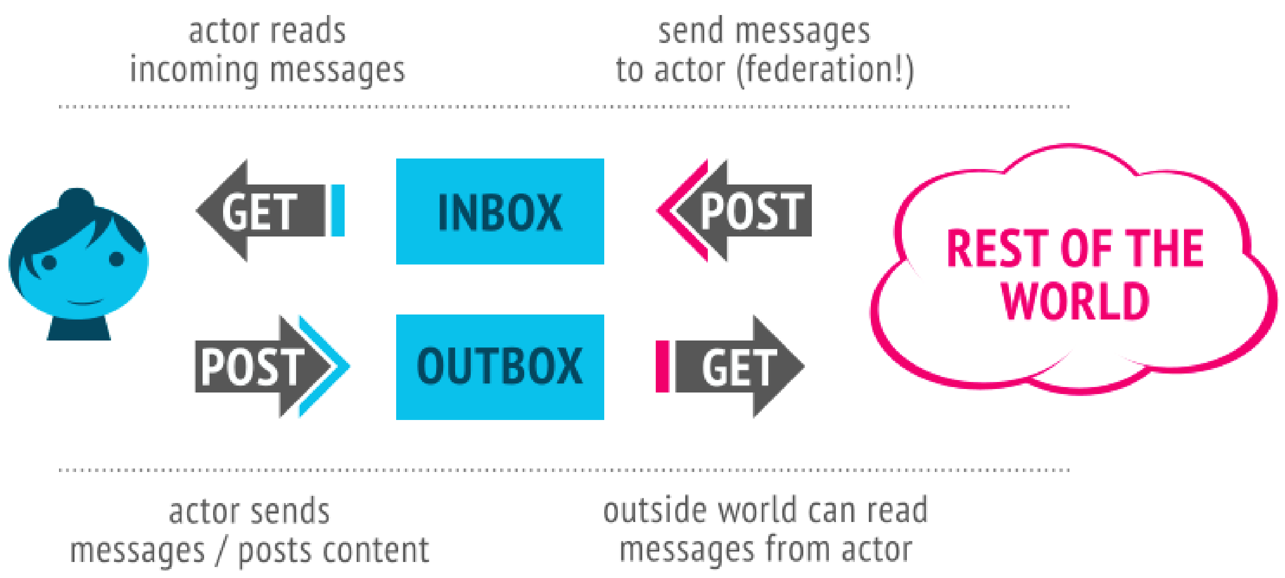

actor reads incoming messages
send messages to actor (federation!)
GET
INBOX
POST
REST OF THE WORLD
POST
OUTBOX
GET
actor sends messages / posts content
outside world can read messages from actor

Figure 3.7: Concept of the Social API protocol: The actor fetches messages with an HTTP GET request to his inbox and posts messages using an HTTP POST request. The inbox is filled by HTTP POST requests from other actors. [73]

occurs, the explicitly shared content is also transmitted. The corresponding actors can only access their own inbox. On access, the messages are downloaded from the server. New messages get in the inbox per HTTP POST request from another server.

Each actor has some so-called collections. Objects (depends on the collection, e.g., accounts, posts) can be removed or added to the collections. The collections are used to store information related to an actor. These are the collections each actor has:

- **Followers Collection**: List of actors who have posted the actor a follow-activity and follow him. Can be used as addressee when sharing a post.
- **Following Collection**: List of actors followed by an actor and whose content he is interested in
- **Liked Collection**: List of all objects the actor has liked

The following interactions are defined between the client and the server, but some of them are optional to implement: Create, Update, Delete, Follow, Add (add an object to a collection), Remove (remove an object from a collection), Like, Block, Undo. To address the activities, *to*, *bto*, *cc*, *bcc* provide the same possibilities as they are known from e-mail. The server then has to ensure that the activity also reaches the addressed actors.

Server to Server (Federation Protocol)

This protocol defines the exchange of activities between actors of different servers. The network between the servers with all actors is called a social graph. The interactions defined in the Social API must be implemented by the server so that they reach the addressed actors. The starting point is always the outbox of an actor. A new activity is created using an HTTP POST request to the outbox. If, for example, the follower collection of the actor is selected as the addressee, the server has to ensure that every actor in the follower collection receives the activity in its inbox. The recipients and their addresses can be found by following the links in the activities. It is also up to the server to ensure that there is no duplication of content.

Application Examples

The most popular application example is the social network Mastodon[10]. Mastodon is a decentralized network based on free and open source software. Everyone is invited to host their own platform. With currently 2 million users it is the most significant implementation of the ActivityPub Protocol [16]. The Federation Protocol is used for communication between the individual servers. The Social API is not used, instead Mastodon offers its own API[11] for communication with the client.

Networking with other users is not limited to mastodon instances. Each service that has implemented the ActivityPub Protocol allows its actors to network with actors of entirely different applications because the communication is standardized. So it is possible without problems to follow an actor of the video platform PeerTube[12] as Mastodon actor and be notified when he uploads a new video there.

In addition to cross-platform networking, another advantage is caused by decentralization and independence. No matter what happens to Mastodon, the network can still exist and, thanks to the open protocols and open source software, it can be developed and used without restrictions. If Facebook would go offline, all contacts would be lost, and the platform would never be accessible again.

3.5 Summary

The relevant work was divided into four groups. Twitterize and FaceCloak are among the extensions that make the use of existing OSNs more secure. Twitterize forms an overlay network for hashtags on Twitter that allows secure and anonymous communication. The browser extension Facecloak protects privacy by transmitting fake data to the Facebook servers, which are recognized when displayed and replaced by the real data. The decentralized social networks diaspora* and LifeSocial promise the protection of data through their decentralized design. The LifeSocial application creates a P2P network among the participants, so no additional servers are necessary. Each user provides resources so that the network scales automatically. The diaspora* application runs on servers and is accessible via a web front end. Each user can operate his own diaspora* instance and thus manage his data. Emerging technologies like blockchain and cryptocurrencies are used to create dApps. In this chapter two social network dApps AKASHA and Peepeth were introduced. Both use the Ethereum blockchain and the IPFS protocol to store data.

[10] https://joinmastodon.org/
[11] https://docs.joinmastodon.org/api/guidelines/
[12] https://joinpeertube.org/en/

Finally, the ActivityPub protocol was presented, which defines communication in decentralized social networks.

4 Concept of a Hybrid Online Social Network

The reason for the inadequate protection of personal data lies in the centralized system structure used by all leading social platforms (e.g., Facebook and Twitter). With this structure, the data are stored centrally and mostly unencrypted. The service provider therefore inevitably has access to this data. It is not transparent to users, which data the service provider record during the use and what happens to the data. On the one hand, the user data are evaluated to improve the user experience (suggestions for content matching the user's preferences using recommender systems), but on the other hand also to make a profit. With personalized advertisement and, in the worst case, by selling the data, revenues are generated. Furthermore, the protection of data against access by third parties via official interfaces (harvesting) or unauthorized hacking cannot be ruled out (e.g., Facebook's incident with Cambridge Analytica in March 2018 [43]). Last but not least, due to applicable law, it may be necessary for data to be transferred to secret services or government agencies (e.g., Facebook had more than 100 000 requests from governments between January and July 2018 [36]).

The criticism of the protection of privacy on the Internet, especially in social networks, is not new. Already in 2010, the founders of diaspora* discovered that no social network sufficiently protects the privacy of users [60]. Although the problems and dangers of centralized OSNs are known for a long time and new scandals regularly become known to the public, the users remain mostly loyal to the respective social networks. As a result of the Cambridge Analytica incident, Facebook lost only a few users in Europe but is in the meantime back on the previous level [37]. Alternative social networks, which focus on privacy protection (e.g., Vero[1], Ello[2]), lack attractiveness regarding their design, complexity, and functionality. As a consequence, they do not get enough users and often fail after a short time. The connection to the respective social network is so strong that the barrier for switching to another, more secure social network is not overcome. The amount of content already created, the social network built up, and a large number of contacts using the same platform all create this so-called lock-in effect.

If switching to another platform is not an option, it is necessary to look for better ways to protect privacy on existing platforms. The Researcher Training Group (RTG) „Privacy and Trust for Mobile Users“[3] in area B „Privacy and Trust in Social Networks“ conducts research in this field. Subarea B.2 focuses especially on the protection of privacy in hybrid social networks. A hybrid solution allows the users of centralized OSN to use the platform as usual. Though, additional procedures for the protection of privacy are integrated so that users gain control over their data. The hybrid approach uses the idea of a private P2P network that allows users to communicate securely. The business models of service providers often have data from their users as a central element. Therefore, a solution must be found to provide anonymous data from the private network so that the business models can continue to function. [57, 58]

In the following, a concept for a hybrid OSN will be worked out that takes into account the interests of the different stakeholders. Besides, functionality requirements and poten-

[1] https://www.vero.co
[2] https://ello.co
[3] https://www.informatik.tu-darmstadt.de/privacy-trust/privacy_and_trust/index.en.jsp

tial limitations are listed. Finally, a solution strategy and a possible architecture are presented.

4.1 Stakeholders

Everyone who is affected by a hybrid extension to an OSN has interests which should be considered. These so-called stakeholders can be people, groups or organizations which have the same opinion and the same goals. In the case of an OSN, there is a service provider who operates the platform. The users are split into those who care about their privacy and therefore want to use a hybrid solution and those who just want to use the OSN as it is. The developers of the hybrid solution have interests as well, and lastly, the government is highly interested that law and order are respected.

Even if the stakeholders are not necessarily directly involved in the development process of a hybrid solution, it is still essential to understand the views and interests of the various parties and take them into account. Table 4.1 shows the interests and points of view of the parties involved directly and indirectly with the hybrid extension of the OSN.

Steakholder	Interests	Attitude towards hybrid OSN	Influence on hybrid OSN
Service Provider	• Collect as much user data as possible • Binding users to the platform • Profit maximization	• Negative, as undesirable	• No direct influence on the hybrid solution • By identification of the hybrid OSN users their exclusion • Blocking the hybrid OSN • Great influence
OSN User	• Unrestricted use of the OSN • No disadvantages due to hybrid OSN users	• Neutral	• No influence
Hybrid OSN User	• Unrestricted use of the OSN • Decision-making sovereignty over what happens with data	• Positive, as desired	• Feedback for improvement • Low influence
Developer hybrid OSN	• Scalable, secure, fast data exchange solution • No costs for infrastructure and development	• Positive	• Great influence, since decision-making sovereignty
Governments	• Compliance with laws • Supervision • Censorship	• Depending on policy and legislation • At best, positive and supportive • In the worst case, negative and preventing	• At best, promotion and financial support - great influence • In the worst case legal action - great influence • No other influence

Table 4.1: The table shows the interests, attitudes towards and influence on the hybrid solution of several stakeholders.

4.2 Requirements to the Hybrid OSN

The solution should meet specific functional and non-functional requirements so that users are willing to use another client.

Functional requirements:

- **Standard functionality**: With the hybrid solution, all standard functionalities of the OSN should be unrestrictedly usable. Otherwise, the user would be forced to continue using the original client. The parallel use of two clients for the same OSN is not user-friendly and would be an argument against the use of the restricted hybrid client.

- **Client-side solution**: Since there is no control over the OSN's servers and software, changes can only be implemented on the client side. The alternative Facebook client app „Friendly for Facebook“ for example changes the design on the client side and removes advertisements.

- **Data sovereignty**: The user can actively decide whether his data are stored on the OSN's servers or exchanged with other users via a private, secure channel. This implies that any functionality and the associated data exchange can also be carried out in a private, secure way. The user thus gains sovereignty over his data and can decide for himself whether to entrust it to the OSN.

- **Authorized data access**: Especially the service provider of the OSN should not have access to the private data of a user and should under no circumstances be able to relate them to him. As in the OSN, the author himself should be able to determine with which users he shares his private data and grants them access.

- **Encryption**: Private data are a sensitive commodity and should, therefore, be protected in the best possible way. In particular, encrypted transmission and encrypted data storage are necessary to protect it from unauthorized access by third parties. It should be ensured that the encryption guarantees sufficient protection and is difficult for attackers to crack.

- **Flexible data format**: The future development of the OSN is not foreseeable, and it is not able to take an influence on it. The data format for information exchange in the private network must therefore be able to react flexibly to possible changes. If, for example, dislikes can be assigned in addition to likes in the future or a new post type for videos is added, it should be possible to implement this quickly in the data structure of the hybrid solution.

- **Platform independence**: The hybrid solution should be platform-independent. Most OSNs have clients and web front ends to be accessed on all platforms. For a hybrid solution to be accepted and become a permanent alternative, it must work on the same platforms. Otherwise the user has to use the official clients again. A solution working on the computer at home that is not working on the smartphone restricts the user and avoids the adoption of the hybrid solution.

- **Anonymized data for the service provider**: A particular requirement is the provision of anonymized data from the private network for the service provider. In order to guarantee the operation of the platform, service providers must earn money. The hybrid solution should accept this circumstance and support it by the provision of anonymized data since data usually plays a central role in the business model. The service provider should be able to retrieve the anonymized data via a defined interface.

Non-functional requirements:

- **Minimal additional effort**: Installation, configuration, and operation should be as simple as possible. Increased complexity is an additional barrier to the acceptance of the solution. All users of the OSN, especially technically inexperienced users, should be able to use the hybrid solution. For the success of the hybrid solution, wide distribution is essential, so no user groups should be excluded.
- **Minimal side effects**: There should be no limitations for regular users who do not use the hybrid solution. Cryptic messages, disturbing content and other disturbing elements should be avoided. Annoying messages could lead to the user being blocked by other users or cutting off contact with them.
- **Free service**: Since the use of all significant OSNs is free of charge the user also expects the free use of a hybrid solution. Costs would have a deterrent effect in this use case and prevent the user from accepting the solution.
- **Compliance with policies**: By registering, the user agrees to the terms of service/use. It is important not to violate these conditions; otherwise, the user would have to bear the consequences. The severity of the consequences depends on the respective terms of use and the type of violation. For the hybrid solution, it is therefore essential to adhere to the OSN terms of use as well. Otherwise, legal use is not possible.
  The Twitter Terms and Conditions allow the crawling of content within the rules set in the robots.txt file [70]. Facebook completely prohibits the use of automated methods, thus excluding the possibility of crawling [35]. Compliance with the guidelines therefore also directly restricts the possibilities of technical implementation.
- **Good user experience**: The user should be offered the best possible user experience to increase the acceptance for the solution. A bad user experience causes frustration and will in the long run not lead to success of the hybrid solution. Good user experience includes:
  - Short loading times, so waiting times are kept to a minimum. Hence, finding and loading data hast to be quickly.
  - The user interface of the hybrid solution should be simple and understandable and have a modern and appealing design.
  - When integrating into the OSN user interface, the existing layout should not be broken, and additional control elements should be inserted in the same design.
  - Privacy should be protected by design. The opposite is known as dark design pattern „Privacy Zuckering“ (named after Facebook CEO Mark Zuckerberg), when controlling the privacy is purposely made difficult [13].

4.3 Restrictions

When designing the hybrid OSN, there are some limitations that need to be considered, for which appropriate solutions have to be found. These restrictions include:

- **API of the OSN**: Ideally the OSN offers a public API with full functionality. An API consists of multiple endpoints that can be used to retrieve, add, or modify specific data. Since the user's data should be protected as best as possible, access via the API is usually restricted. The restriction may be due to a limited number of requests per time interval or a limited range of offered functions.

- **Crawling the OSN web pages**: If there is no official API or if it is sharply restricted, the contents can theoretically also be extracted by crawling. However, this brings with it several challenges. Modern web pages load many contents asynchronously so that the initial Hypertext Markup Language (HTML) does not yet contain these contents. Furthermore, there are sophisticated mechanisms that notice crawling and lock out crawlers. Likewise, it may be difficult to add data to the OSN. For security reasons, in most cases, special tokens are sent along with each request to detect and prevent abuse and fake requests.

- **Creation, operation and licensing costs**: Costs for the creation, operation and licensing of third-party software may be incurred. At best, conscious decisions lead to the avoidance of expenses.

- **Operating system or runtime environment**: Nowadays OSNs can be used on almost all devices; independent of their operating system. In order to achieve the same user experience, the hybrid OSN should be usable on the same variety of operating systems. Any restrictions imposed by the operating system (user and application rights, connectivity) must be taken into account during creation.

- **Resources**: The mostly mobile devices (Twittter is used by more than 80% of users on mobile [19]) running the hybrid OSN may have limited resources (storage space, processing power, Internet connection/data volume, battery). When making design decisions, it is important to plan as resource-conserving as possible and to find scalable solutions. Overall, the overhead for the hybrid extension should be as low as possible compared to the original application.

- **Availability of data**: The data that are exchanged securely and not via the OSN's servers must always be available. Whether a user is offline or how old the data are, should have no effect its availability.

While the restrictions on the hybrid client itself can be actively influenced and resolved, the restrictions on the OSN cannot be controlled. If the OSN does not provide any interfaces and the hurdle of data exchange with the servers is insurmountable, this can completely prevent the creation of a hybrid client.

4.4 Quality Goals

The following quality objectives (Table 4.2) have been defined to ensure that the quality of implementation is as high as possible. By adhering to the quality goals, errors and problems should be avoided, the application should remain maintainable, and new developers should be enabled to get started quickly.

Quality Goal	**Motivation**
Analyzability	- Module, class and method names in English - Detailed documentation of the public interfaces - Compliance with the Clean Code principles
Changeability	- Common programming language - Program modules against interfaces to keep them interchangeable
Testability	- The architecture should allow easy testing of all building blocks
Transparency	- The application should be Open Source

Table 4.2: The quality goals are the relevant requirements and the driving forces that software architects and developers should consider.

4.5 Solution Strategy - Architecture

Various models can be used to implement a secure data exchange between the users of an OSN via additional network. The solution strategies shown below differ primarily in the question of where data are stored and how it can be found.

One possibility is to use an extra infrastructure to store the data, as shown in Figure 4.1a. An additional server stores and distributes the private data to be protected. Using a server has the advantage that the data are always available and there are no dependencies to other hybrid OSN users. Furthermore, resources only have to be available centrally and not locally on the user's device. At the central location, the data can be indexed and explicitly queried. However, the operation and maintenance of one or more servers are problematic. In principle, the question for the service provider has to be clarified, because the reliability of the infrastructure is essential. FaceCloak (see Chapter 3.1.2) used an architecture based on this structure.

Instead of operating a separate, additional server, it would also be possible to use a third-party, existing infrastructure. These include, for example, blockchains or P2P file-sharing networks that could be used for data exchange. Since no influence can be exerted on existing infrastructure, its use entails further restrictions and potential risks.

A decentralized solution strategy would create a network among users of the hybrid application (see Figure 4.1b). No extra infrastructure would have to be operated. The users would then have a typical peer role. By using this model, it is difficult to keep data available and accessible even if the user is permanently or temporarily offline. The problem needs to be solved. Furthermore, the resources on the devices are limited, so that effective and economical solutions are needed. Another challenge is the addressing of peers. Since they typically do not have a static Internet

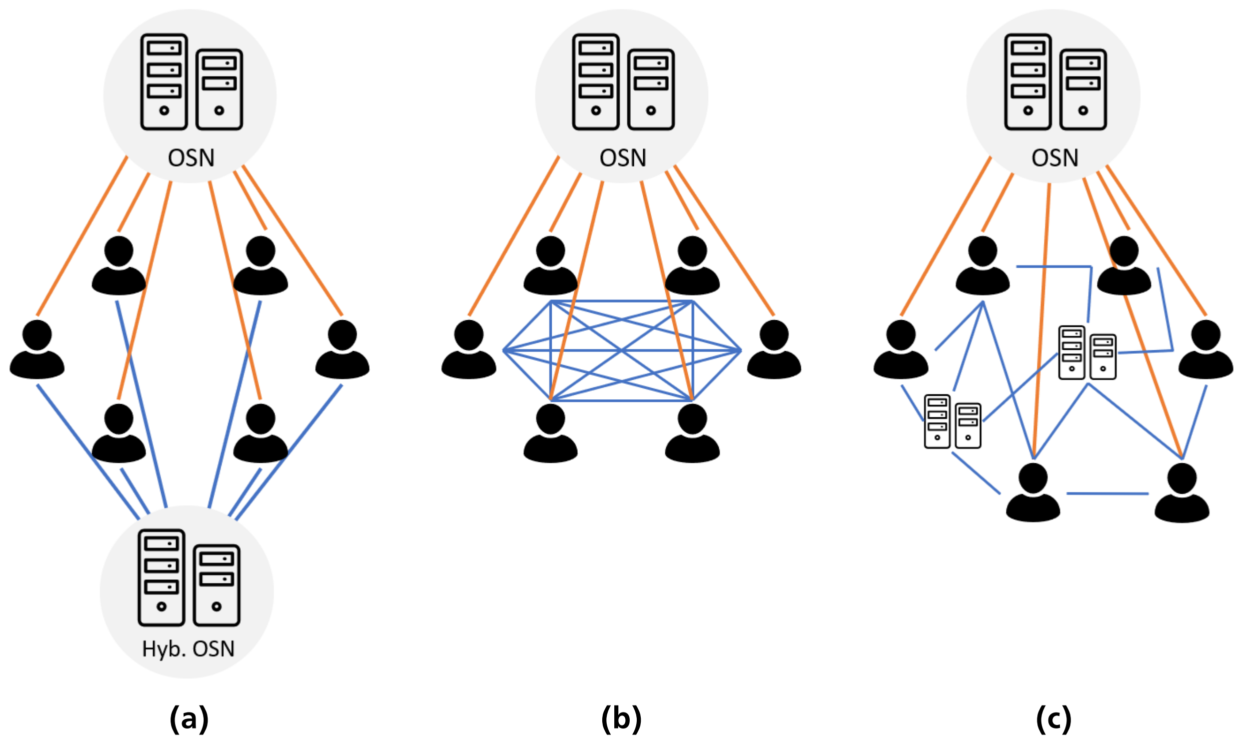


Figure 4.1: Architectures for secure data exchange among users: (a) by the use of an additional server, (b) via a P2P network connecting all users or (c) via a hybrid P2P network with servers acting as super-peers.

Protocol (IP) address, solutions have to be found for accessibility. Since there is no central, global index, finding data is even more difficult.

Adding servers to the P2P network would create a hybrid solution (see Figure 4.1c). In this model, the servers would take on the role of a super peer, permanently reachable at a fixed address, thus stabilizing the P2P network. The problem of data availability could be limited by storing much of the data at super peers. The problem of addressing would also be solved by establishing connections to other peers via the known super peers. However, the problem would remain with the cost and maintenance of the servers.

Table 4.3 lists the advantages and disadvantages of the different strategies for the hybrid OSN architecture.

4.6 Solution Strategy - Client

Concerning the implementation of the hybrid approach, two possibilities are conceivable. On the one hand, the extension of the original OSN client (app or web front end) as an add-on. On the other hand, the creation of an entirely new client.

When an add-on extends the OSN, there is no need to take care of the standard functionality of the OSN. Therefore, the development can entirely focus on the add-on and the private extension. At crucial points, the add-on extends the interface by additional elements that enable secure data exchange. The service providers usually do not offer developers an interface to

	Advantages	**Disadvantages**
Own infrastructure (centralized)	• Availability of data • Finding the data • Resources only have to be available centrally • No dependencies among hybrid OSN users	• Expenses • Who operates the infrastructure? • Compliance with legal requirements
P2P network (decentralized)	• Resources scale with increasing number of users	• Availability of data • Finding the data • Addressing the peers • Local resources limited
Hybrid P2P network (decentralized)	• Availability of data • Peer discovery	• Expenses • Who operates the infrastructure? • Finding the data
External infrastructure	• Ideally no costs • Resources are provided by the external infrastructure	• No influence on future development • Dependence on infrastructure entails risks

Table 4.3: Advantages and disadvantages of the different solution strategies for the hybrid OSN architecture.

extend the OSN with such own functionalities. With web front ends it is possible to manipulate the website content using browser add-ons. One example for doing so is FaceCloak. Since such browser extensions manipulate the Document Object Model (DOM), knowledge about the document structure is necessary for the successful function in order to make changes in the right places. The short release cycles of OSNs and the associated frequent changes to this DOM structure make it difficult to keep up with the changes. When consuming the OSN via the official apps on mobile devices, an extension or manipulation is not possible.

The alternative to the extension approach described above is a new hybrid client app. The entire functional range of the OSN must be implemented and kept up to date. As already mentioned with the restrictions (see Chapter 4.2), the functions are usually not entirely provided via an API, and the crawling of the content confronts several challenges. By having complete control over the development of the client, the additional protected, secure communication can be added at the appropriate points. In the best-case scenario, at least one hybrid app is available for every operating system for which an official OSN app exists.

Both approaches can be combined by displaying the (mobile) web page of the OSN in a WebView in a separate app and executing DOM manipulations via injected JavaScript code. For example, there are some alternative clients for Facebook (e.g. „Friendly for Facebook“[4] [5], „Metal Pro“[6]) that use this approach.

4.7 Summary

In this chapter, the concept for a hybrid OSN was worked out. This includes an analysis of the interests of the stakeholders and their interests. In addition to the service provider and the users (divided into those who use a hybrid solution and conventional users), the stakeholders also include governments and the developers of the hybrid solution. There are some limitations to the implementation of the concept. These limitations have been stated and presented. Also, functional and non-functional requirements were defined and discussed. Finally, solution strategies for the architecture and the client application were mentioned and their advantages and disadvantages considered.

4 https://play.google.com/store/apps/details?id=io.friendly

5 https://itunes.apple.com/de/app/friendly-for-facebook/id400169658

6 https://play.google.com/store/apps/details?id=com.nam.fbwrapper.pro

5 Proof of Concept

After the concept of a hybrid OSN was introduced in the previous Chapter 4, this chapter describes the implementation of the concept in a prototype. The prototype is an Android app for the hybrid use of Twitter. The app is named **Hybrid OSN**. In the following, the objectives and decisions made for the selection of the OSN and the technology are described. The architecture and its components, as well as individual processes, are then presented.

5.1 Objective

With the proof of concept, the basic feasibility of the idea of an extension of an established OSN by a secure data exchange should be proven. Within the framework of this thesis, the task was to provide the proof of concept as a native Android app. Concerning the architecture, the focus from the beginning was on a P2P solution. For this reason, a solution with additional servers was not pursued further in the creation process of the prototype. Furthermore, an interface should be available for the service provider of the OSN, through which anonymized information can be obtained from the privately exchanged data.

With the decision for a client app, the creation of an add-on, which was discussed in the previous chapter as a possible solution strategy, was thus fundamentally ruled out. Nevertheless, consideration always included how the architecture could be this open and flexible to enable all kinds of extensions and clients participating in the P2P network.

Since it is only a proof of concept, the mapping of the complete functionality of the OSN was not the highest priority. However, again, this was taken into account by considerations and decisions, for example how data formats can be arranged so flexible that every function can be mapped.

Also, the focus was on compliance with the quality goals and implementation of the functional and non-functional requirements as defined in the previous chapter. How well this has been achieved will be evaluated in the following chapter for evaluation and discussion.

5.2 Selection of the OSN

To select a suitable OSN for the creation of a hybrid client, three factors were considered:

1. the significance of the OSN,
2. the adequacy of data protection,
3. the API functionality.

Facebook was the obvious first choice due to the numerous negative headlines about data protection. With over 2.3 billion users per month (average Q4 2018), it is currently the most widely used social network in the world [37]. In the recent past, it has often been criticized for its handling of its users' data. In particular, the scandal surrounding the data analysis company Cambridge Analytica, which had access to the data of up to 87 million users, hit Facebook hard [43, 62]. As a result, CEO Mark Zuckerberg had to face the US Congress and the EU Parliament in question rounds and did not leave a good impression by avoiding many questions [38, 72]. As a result of this scandal, there were further restrictions to the Facebook API [62].

However, the Facebook API is not suitable for creating a new client. The functionalities provided by the API offer the possibility to build an app that can be used within Facebook, for example for a game. So, it is not possible to give a „Like" for a post through this API, which is part of the core functionality of a Facebook client. As discussed in Chapter 4, it is possible to access the data through crawling. However, the rapid changes to the page structure would make this an arduous undertaking. Facebook writes in a blog post that the code changes every few hours [55]. Therefore it is almost impossible to adjust the crawler fast enough and roll out the updated code.

Even the mixed version of displaying and manipulating the mobile website in a WebView in a container app does not seem to be an option due to the short release cycles and frequent changes. Apps like „Friendly for Facebook" do not manage to keep up with the changes as reported in various user ratings on the Google Play Store. The false representations and bugs worsen the user experience and result in the frustration of the users.

For this number of reasons, Facebook dropped out as an OSN candidate for the prototype despite the particular interest. As a further candidate, the OSN Google Plus was dropped, as Google announced in October 2018 that it would discontinue its OSN [63].

Finally, Twitter was chosen for the prototype. With 321 million active users per month (average Q4 2018), it is one of the largest social networks [71]. It is particularly well suited for the creation of a hybrid client for two reasons: first, it has a comprehensive API that provides almost full functionality free of charge, and second, compared to Facebook, it offers only a few simple functions. These are the ideal prerequisites for the first proof of concept. Twitter offers several APIs for developers that serve different purposes. The current APIs are [69]:

- **Standard API**: the free and public API offering basic query functionality and foundational access to Twitter data.
- **Premium API**: introduced in November 2017 to close the gap between Standard and Entrprise API. Improvements over the Standard API: „more Tweets per request, higher rate limits, a counts endpoint that returns time-series counts of Tweets, more complex queries and metadata enrichments, such as expanded Uniform Resource Locators (URLs) and improved profile geo information"[67]. Prices to use this API start at 149$/month.
- **Enterprise API**: tailored packages with annual contracts for those who depend on Twitter data.
- **Ads API**: this API is only of interest for creating and managing ad campaigns.

In the case of the hybrid client, the standard API can be used. In this process, the registration of a „Twitter App“ is necessary to receive a consumer key and access token. These two authentication tokens are required to log in users via the hybrid app and successfully communicate with the API.

Twitter offers almost the entire range of functions via the API. The missing functionality (e.g., the targeted retrieval of replies to a tweet) is not so critical for building a client app. A significant limitation is a restriction on the number of requests. Twitter argues that this restriction is necessary to avoid the exposure of the system to too much load. It also aims to prevent bots from abusing Twitter. The exact limits can be found on a help page[1]. In the app stores of Google and Apple, there are a number of alternative Twitter clients (Twitterific[2], Talon for Twitter[3], Fenix 2 for Twitter[4]), which are also subject to these restrictions in terms of functionality and requests. Their success indicates that the restrictions do not affect the common usage seriously.

The API can be accessed using HTTP requests. The data exchanged are in JSON format. Furthermore, there are also various libraries (e.g., Twit[5]), some even are offered directly by Twitter (see Twitter Kit for Android[6] or iOS[7]) and simplify the use of the API.

5.3 Technology Decisions

In order to meet the requirements of the concept best, a detailed consideration of the technology to be used is necessary. These decisions include the application framework itself and libraries and protocols for data exchange in a P2P network. Basically, there are two conceivable approaches for the P2P network:

- The creation and buildup of a separate P2P network connecting the hybrid clients
- The use of an existing P2P network for own purpose

5.3.1 Creation of a P2P Network

The advantage of having an extra P2P network is that it is completely under control. Accordingly, it can be designed to fit exactly to the use case and require little or no compromise. However, setting up a P2P network is a big challenge and some hurdles must be overcome. These challenges include peer discovery (how peers find each other), global data exchange over the Internet and data storage, and availability of the stored data. In addition, all these requirements must scale. It should work for P2P networks with only a few peers and also for a few thousand or even more peers.

1 https://developer.twitter.com/en/docs/basics/rate-limits
2 https://itunes.apple.com/de/app/twitterrific-5-for-twitter/id580311103?mt=8
3 https://play.google.com/store/apps/details?id=com.klinker.android.twitter_l
4 https://play.google.com/store/apps/details?id=it.mvilla.android.fenix2
5 https://github.com/ttezel/twit
6 https://github.com/twitter/twitter-kit-android
7 https://github.com/twitter/twitter-kit-ios

In the following, we discuss two options to create the P2P network:

- The use of an established standard such as Wi-Fi Dircet and Web Real-Time Communication (WebRTC)
- The use of a library (e.g., Hive2Hive, Yjs) to create a dedicated P2P network

Wi-Fi Direct

Wi-Fi Direct is a standard (IEEE 802.11) for data transmission between two WLAN terminals. There is no need for an access point between the two devices. However, the distance that may lie between the two peers is thus limited. Without obstacles, which contribute to the attenuation of the signal, a distance of up to 200 m is possible [4]. However, the requirement for the P2P network to expand Twitter is different. Users can be scattered around the world and may be online through the mobile network. There is no P2P connection via Wi-Fi Dircet possible in this case.

WebRTC

WebRTC is an open standard[8] that provides various communication protocols for real-time communication between two or more peers. Above all, WebRTC is popular for its ability to easily perform video calls, as known from Skype, in the browser without a server. Additionally it allows file transfer between peers. In a separate P2P network, data could be exchanged between the clients. However, the connection between the clients is complex, beside two peers also a Session Traversal Utilities for NAT (STUN) server and optionally a Traversal Using Relays around NAT (TURN) server is involved. Furthermore, it is unclear whether the system scales.

Yjs

The JavaScript library Yjs[9] describes itself as „a framework for offline-first P2P shared editing on structured data-like text, richtext, JSON, or Extensible Markup Language (XML)“[44]. The library takes care of solving synchronization conflicts when editing distributed files. By choosing a connector, you can set the protocol for the communication between the peers. There is the possibility to use WebRTC, Extensible Messaging and Presence Protocol (XMPP) or websockets, but with some connectors running a server is a prerequisite. Further extensions can be used to supplement a database and data types, but the focus here is on the joint editing of data by multiple peers. For all connectors, the authors of the library recommend using an own server. Yjs is released under the MIT license.

[8] https://www.w3.org/TR/webrtc/
[9] https://github.com/y-js/yjs

Hive2Hive

Hive2Hive[10] is a Java library for „secure, distributed, P2P-based file synchronization and sharing“[48]. There is no less promise than „a free and open-sourced, distributed and scalable solution that focuses on maximum security and privacy of both users and data“[48]. In order to be able to use Hive2Hive globally via the Internet, it is necessary to operate at least one relay peer - five are recommended for sufficient data replication [59]. Since a permanent Transmission Control Protocol (TCP) connection between peer and relay peer is maintained, the power consumption is quite high. Constant TCP connection can be avoided by using Google Cloud Messaging. However, Google has already discontinued this service. Moreover, the development of Hive2Hive was stopped in March 2015 too, there is currently no solution to this problem. Hive2Hive is released under the MIT license.

GUN

In 2014, Mark Nadal began working on a decentralized database called GUN. GUN[11] is written in JavaScript and will be further developed by the community as an open source[12] project. It is licensed under the Apache License 2.0, so free use is allowed. The P2P database exists only within a browser and synchronizes itself with other peers over the Internet. It has offline support, so changes are automatically synchronized and merged as soon as a connection is established. The data are stored in a graph structure in the local storage of the browser. For connecting peers among each other, at least one relay server is required.

OrbitDB

The open source project OrbitDB[13] is a serverless, distributed, P2P database on top of IPFS. The data are stored in IPFS and IPFS Pubsub is used to sync the database automatically with other peers. Various types of databases are provided like log, feed, key-value, and counter. There is a JavaScript library for usage in browsers and NodeJS which is currently in alpha stage. OrbitDB is released under the MIT license and therefore free to use in private and commercial projects.

5.3.2 Using an Existing P2P Network

As mentioned in the previous section, setting up your own P2P network involves a number of challenges. If you use an already existing P2P network for your own purpose, these challenges can be elegantly avoided. For this purpose, however, a suitable P2P network must be found whose properties and possible uses meet the requirements of the hybrid OSN client. In the

[10] https://github.com/Hive2Hive/Hive2Hive
[11] https://gun.eco
[12] https://github.com/amark/gun
[13] https://github.com/orbitdb/orbit-db

following, various P2P networks are considered and examined for their usability for a potential deployment to extend an OSN.

Blockchain

In a blockchain, the private data of the users could be stored encrypted and shared with other users. The corresponding smart contracts would have to be designed for this purpose. Regardless of the blockchain, executing the smart contracts to store the data, however, consumes a token. The user has to get the tokens beforehand and deposit them in his wallet. To avoid this, a test network (e.g., Rinkeby at Ethereum) could be used. A test network would have the advantage that tokens could be made available to the user free of charge and that the execution of the smart contracts would thus also be free. However, test networks are used to test new software updates and usually do not offer the same security as the main network. They should only be used to test applications before releasing them to the main net. In order to protect against abuse, token requests are usually limited and take a while until they arrive in the user's wallet. Besides, it is often required to post specific content in a social network or to perform another task.

A solution based on a blockchain increases complexity for the user. Since the hybrid solution was required to keep the complexity and additional effort as low as possible, such a solution using a blockchain was categorically excluded.

IPFS

IPFS[14] is a distributed file system that brings together the ideas of other successful P2P systems (DHTs, BitTorrent, Self-Certified Filesystems (SFS) and Git) [26]. The original idea for IPFS came from Juan Benet in 2014. Protocol Labs[15], founded by Benet, now manages the open source project [5].

The goal of IPFS is to connect all computing devices to the same file system. The basis for this is a P2P network where the data are distributed and stored. Instead of a location-oriented approach as used in HTTP, IPFS uses a content-oriented protocol. A cryptographic hash function generates a hash (content identifier) for each file, which can be used to find the file on the P2P network using a DHT. Unnecessary redundancies are avoided in this way, as the content is only stored once and purposefully replicated. However, deleting data is impossible in IPFS. Only if all peers who have stored copies of the file leave, the file can no longer be recovered. Decentralization allows a file to be retrieved from several peers in parallel, thus making the download faster. Besides,

Due to its characteristics, IPFS is an ideal complement to the blockchain and the creation of dApps. Large amounts of data can be stored out of IPFS, and the address hash can be written into the blockchain. This principle is used for example by AKASHA and Peepeth.

[14] https://ipfs.io/
[15] https://protocol.ai

IPFS clients are available for Max OS X, Linux and Windows. Besides, there are client implementations in JavaScript and Go to include in other applications.

Conclusion

First, the goal was to build a P2P network among the users of the Hybrid OSN app to use it for secure data exchange. Due to its limitation to local use, Wi-Fi Direct was excluded as a possibility. The need for STUN and TURN servers and the uncertainty regarding the scalability of the system led to the elimination of the WebRTC standard as the basis for communication. While Yjs made a promising impression, the use case ultimately did not fit into a hybrid solution for OSN. The Java library Hive2Hive dropped out because of the need for at least one, ideally five, relay servers. Furthermore, the library is discontinued since 2015 and therefore questionable to what extent it is compatible with the current Android Software Development Kit (SDK) version. Since the potential solutions for building an own P2P network are limited or outdated, the idea of an exclusive P2P Hybrid OSN network was discarded.

As a further solution strategy, the use of third-party services from Google Firebase[16] and PubNub[17], which provide the functionality of a database, was considered. To a limited extent, the services offered can be used for free. From a particular size of the user base and correspondingly large load, costs for the service use would rise. In order to avoid these charges from the beginning, the idea of using these commercial third-party services was rejected.

The dApps AKASHA and Peepeth introduced in Chapter 3.3 use the Ethereum blockchain as a database and IPFS for data storage. This technology combination meets the technical requirements of a hybrid solution. However, the use of a blockchain was excluded due to the usage costs and complicated configuration for the user (wallet, acquisition of tokens). Nevertheless, IPFS is an excellent way to store data decentralized. Accordingly, only a replacement for the database functionality was needed. With OrbitDB and GUN, two technologies came into closer selection for use in Hybrid OSN. Since OrbitDB offers database functionality based on IPFS, it was initially preferred. However, when using Cordova (the basis of the used application framework Ionic), the JavaScript implementation of IPFS and the OrbitDB library, an error occurs. This error has been known for some time but is still unsolved due to its high complexity and the comparatively small number of affected users. Therefore, the final choice was GUN to implement the database functionality of the hybrid solution. At the time of the decision for GUN, it was assumed that the relay server used for testing purposes in the official GUN tutorials could also be used as a relay server within Hybrid OSN. However, later it turned out that the use is very strongly restricted and the limit is already reached after three private tweets. For this reason, a separate relay server was set up as a workaround. GUN provides the necessary code as well as the one-click setup at Heroku[18] on Github.

[16] https://firebase.google.com/
[17] https://www.pubnub.com/
[18] https://www.heroku.com/

5.3.3 Application Framework

With the Android SDK apps for Android can be written in Java, Kotlin, and C++. Tools of the Android SDK compile code, data, and resource files to an executable Android Package (APK). This way, only device-specific applications for the Android operating system can be created. [1]

The open source framework Ionic[19] is used to create hybrid apps and Progressive Web Apps (PWAs). A hybrid app combines the advantages of a native app with those of web applications. With HTML, Cascading Style Sheets (CSS), and JavaScript, this framework can be used to design platform-independent applications (including Android and iOS). The core of Ionic is based on the JavaScript Framework Angular and Apache Cordova, which serves as a bridge to access sensors and functions of the device. Ionic can be used free of charge and is published under the MIT license, so private and commercial use is permitted.

Comparing the standard way of programming an Android app to the use of Ionic, the platform independence and the JavaScript language are the two main advantages. With Ionic, the same code can be used to create apps for the majority of mobile operating systems. Additionally, as PWA the app runs in every modern web browser. The previously defined requirements of a hybrid solution demanded this independence, thus Ionic is a good match. Regarding the programming language JavaScript, many projects in the field of dApps and decentralization in general provide only libraries for JavaScript (e.g., GUN). Besides, JavaScript is the language to write add-ons for browsers. Even though Ionic cannot be used to create browser add-ons, the used JavaScript libraries are able to do so. Hence, the same technology could be used to deliver clients to all popular environments.

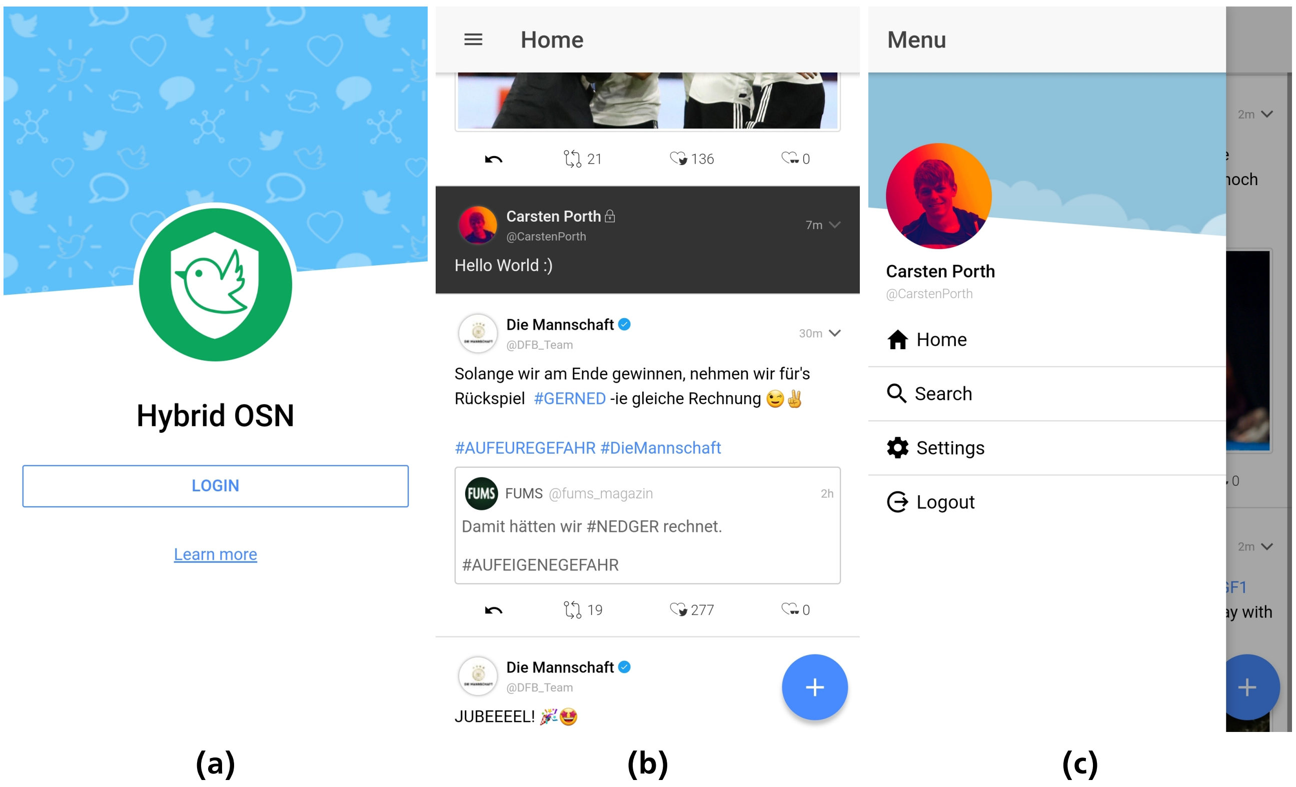
With React Native[20] and NativeScript[21] there are comparable frameworks to Ionic. They also use JavaScript and are used to create hybrid apps for several environments. But the provided components to create the user interface are most promising with Ionic and the initial learning barrier is low compared to the other two. Because of these advantages, Ionic was finally chosen for programming the Hybrid OSN client app.

5.4 Presenting Hybrid OSN for Twitter

When the user opens the app for the first time, the login page is displayed (see Figure 5.1a). Clicking on the login button starts the process. The user is directed to the Twitter website, where he has to log in. Only once, the user has to grant the Hybrid OSN app access to his Twitter account. This is the prerequisite to let the Hybrid OSN app talk to the Twitter API in the name of the user.

After a successful login, the user is forwarded to his home feed (see Figure 5.1b) which is also the starting point if the user is already logged in when opening the app. The home feed consists of the recent public and private tweets in chronological order of the accounts that the user

[19] https://ionicframework.com/
[20] https://facebook.github.io/react-native/
[21] https://www.nativescript.org/



Carsten Porth @CarstenPorth 7m
Hello World :)

Die Mannschaft @DFB_Team 30m
Solange wir am Ende gewinnen, nehmen wir für's Rückspiel #GERNED -ie gleiche Rechnung 😉✌

#AUFEUREGEFAHR #DieMannschaft

FUMS @fums_magazin 2h
Damit hätten wir #NEDGER rechnet.

#AUFEIGENEGEFAHR

Die Mannschaft @DFB_Team
JUBEEEEL! 🎉😍

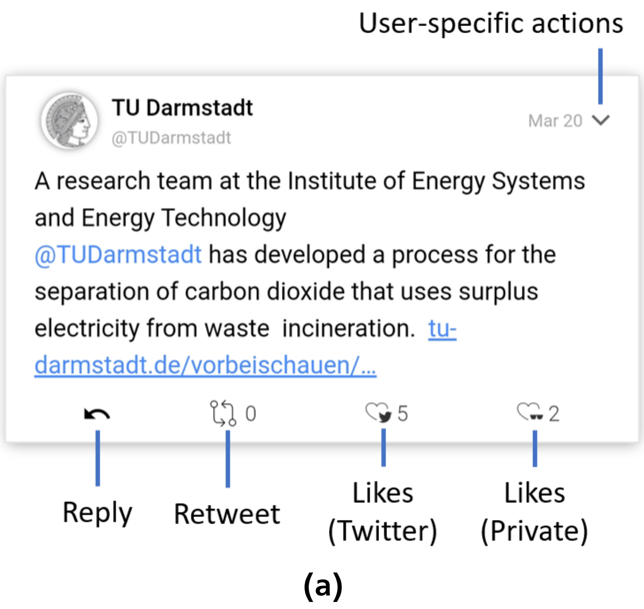
Figure 5.1: Screenshots of the Hybrid OSN app for the login page (a), the home feed (b) and the menu (c).

follows. Tapping on the hamburger icon opens the menu (see Figure 5.1c) where the user can access his profile, the home page, settings and search at any time. The user can also log out via the menu.

A tweet consists of information about the author and the time of the writing, the message itself and actions. Tapping on the arrow in the top right can be used to perform user-specific actions, such as blocking, muting and following the user. These actions are presented in an extra menu at the bottom of the screen (see Figure 5.2b). Below the tweet message, icons allow direct interaction with the tweet, including replying, retweeting and public and private liking (see Figure 5.2a).

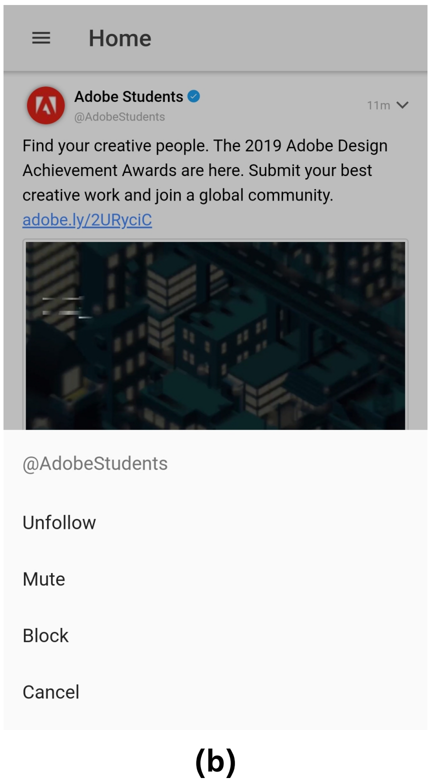
In the settings, keywords can be configured that are considered indicators of tweets that are particularly worthy of protection (see Figure 5.3a). When writing a new tweet, the user is warned accordingly if one of these words is part of the text. Also, private and public keys can be generated and stored on the settings page to encrypt and decrypt private tweets (see Figure 5.3b). The public key can be published by clicking on a respective button. For this purpose, the key is entered into the user's so-called public key history together with the start of validity, stored in IPFS and the hash is posted in the user's profile. When logging out, all stored settings are automatically deleted.

With the search, tweets and users can be looked up for a specific keyword (see Figure 5.3c). In the search results of the tweets, the most recent or the most popular tweets are shown, split into two tabs. However, private tweets are not included in this search.



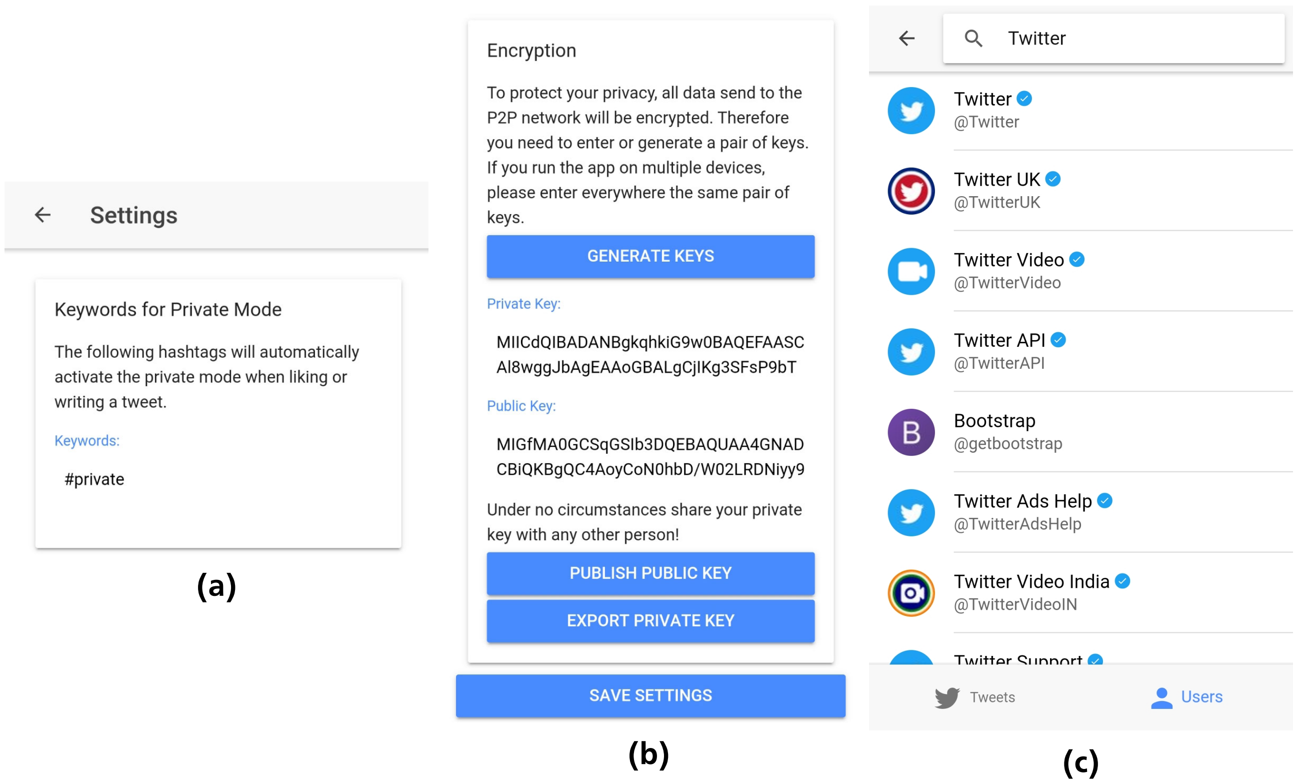
TU Darmstadt
@TUDarmstadt
Mar 20

A research team at the Institute of Energy Systems and Energy Technology @TUDarmstadt has developed a process for the separation of carbon dioxide that uses surplus electricity from waste incineration. tu-darmstadt.de/vorbeischauen/...



Find your creative people. The 2019 Adobe Design Achievement Awards are here. Submit your best creative work and join a global community. adobe.ly/2URyciC

Figure 5.2: Screenshots of the Hybrid OSN app for tweet and its actions (a) and the menu with user-specific actions (b)



Keywords for Private Mode

The following hashtags will automatically activate the private mode when liking or writing a tweet.

Keywords:

#private

Encryption

To protect your privacy, all data send to the P2P network will be encrypted. Therefore you need to enter or generate a pair of keys. If you run the app on multiple devices, please enter everywhere the same pair of keys.

GENERATE KEYS

Private Key:

MIICdQIBADANBgkqhkiG9w0BAQEFAASC Al8wggJbAgEAAoGBALgCjlKg3SFsP9bT

Public Key:

MIGfMA0GCSqGSIb3DQEBAQUAA4GNAD CBiQKBgQC4AoyCoN0hbD/W02LRDNiyy9

Under no circumstances share your private key with any other person!

PUBLISH PUBLIC KEY

EXPORT PRIVATE KEY

SAVE SETTINGS

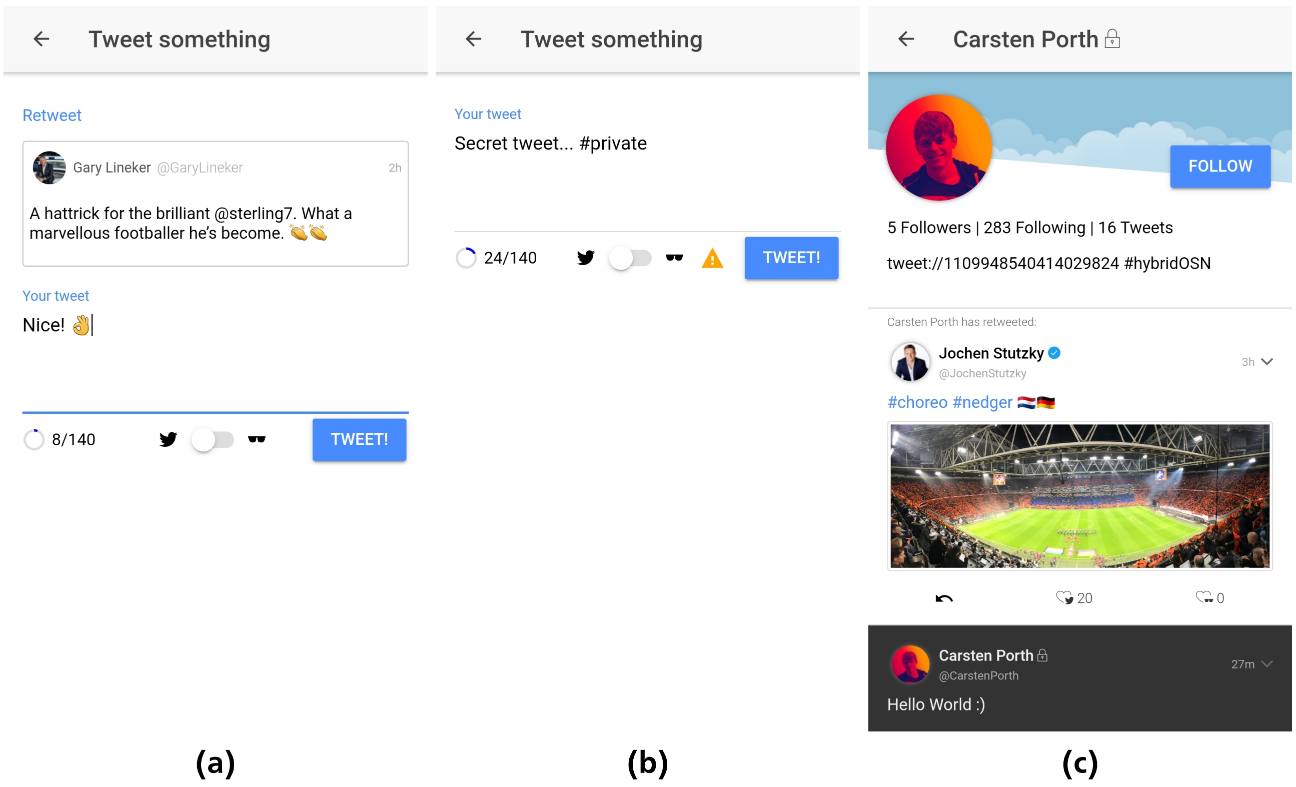
Twitter

Twitter @Twitter
Twitter UK @TwitterUK
Twitter Video @TwitterVideo
Twitter API @TwitterAPI
Bootstrap @getbootstrap
Twitter Ads Help @TwitterAdsHelp
Twitter Video India @TwitterVideoIN
Twitter Support

Tweets Users

Figure 5.3: Screenshots of the Hybrid OSN app for the settings page to configure keywords (a) and the encryption keys (b), and the search page with results for users (c).

The page for composing a new tweet is also used when replying to a tweet or retweeting a tweet. When replying and retweeting, the particular tweet is presented above the text field for the message (see Figure 5.4a). To keep an eye on the character limit when writing, the current number of characters is displayed below the text field. Next to it is an additional visualization of the progress in the form of a circle. A switch controls to which network the tweet is posted. By clicking on the „TWEET!“-button, the tweet is published. If the message contains one of the keywords defined in the settings, a yellow, pulsating warning triangle will appear next to the tweet button (see Figure 5.4b).



Gary Lineker @GaryLineker 2h

A hattrick for the brilliant @sterling7. What a marvellous footballer he's become. 👏👏

Your tweet

Nice! 👌

8/140

Your tweet

Secret tweet... #private

24/140

5 Followers | 283 Following | 16 Tweets

tweet://1109948540414029824 #hybridOSN

Carsten Porth has retweeted:

Jochen Stutzky @JochenStutzky 3h

#choreo #nedger 🇳🇱🇩🇪

20 0

Carsten Porth @CarstenPorth 27m

Hello World :)

Figure 5.4: Screenshots of the Hybrid OSN app for replying to a tweet (a), writing a tweet containing a keyword triggering the private mode (b) and a user profile (c).

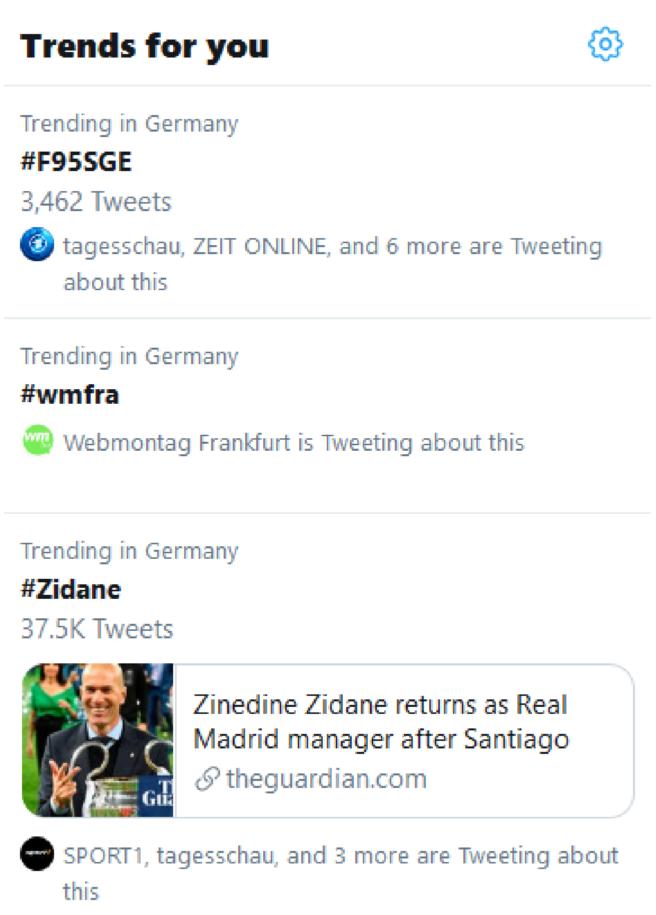
The upper part of the profile page displays information about the user (profile picture, name, Twitter handle, description, website, location) and the lower part displays a feed of his tweets (see Figure 5.4c). Furthermore, a button for following or unfollowing the user is placed in the upper part of the profile.

5.5 Providing Insights to the Service Provider

The requirements mentioned in 4.2 also include the provision of anonymized data for the OSN service provider. Since the business model of Twitter is based on personal data, and therefore the interests of Hybrid OSN are contrary to those of Twitter, the fulfillment of this requirement is extremely complex.

A prominent feature of Twitter is the analysis and promotion of trends (see Figure 5.5a). The trends are identified through frequently used hashtags and presented in a ranking. Such data can also be collected and evaluated in the private network without having to establish a connection to the users.

To collect this information, when a new tweet is posted to the private network, the contained hashtags are extracted and stored separately (see flowchart Figure 5.9). Similar to the presentation of trends on Twitter, the trends in the private network are also aggregated on a daily basis and presented on a website (see Figure 5.5b).



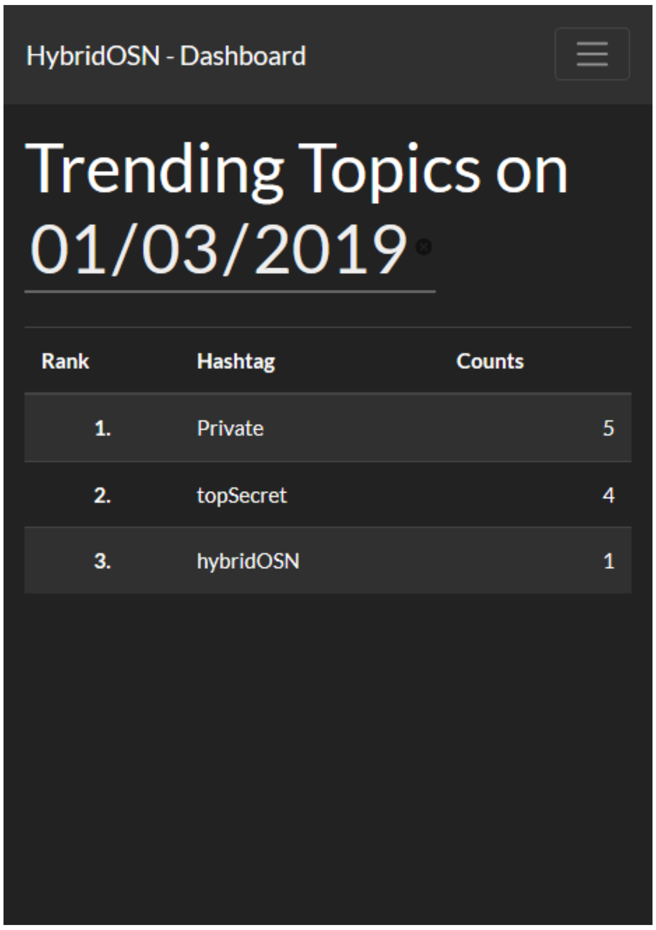
Trends for you

Trending in Germany
#F95SGE
3,462 Tweets
tagesschau, ZEIT ONLINE, and 6 more are Tweeting about this

Trending in Germany
#wmfra
Webmontag Frankfurt is Tweeting about this

Trending in Germany
#Zidane
37.5K Tweets
Zinedine Zidane returns as Real Madrid manager after Santiago
theguardian.com
SPORT1, tagesschau, and 3 more are Tweeting about this

(a) Twitter trends



HybridOSN - Dashboard

Trending Topics on 01/03/2019

Rank	Hashtag	Counts
1.	Private	5
2.	topSecret	4
3.	hybridOSN	1

(b) Hybrid OSN trends

Figure 5.5: Trending hashtags in Twitter and the private network side by side

Because GUN is JavaScript-based and therefore executable in the web browser, access to the data from a simple HTML web page can be performed using JavaScript code. The raw data are loaded, aggregated and displayed.

5.6 Building Block View

In this section, the context in which Hybrid OSN is located is first considered, and then a breakdown into the individual components is carried out. Furthermore, the function of the respective blocks is described in more detail. Finally, the function of individual components in interaction is explained using the examples of displaying the home timeline and posting a new tweet.

5.6.1 Scope and Context

Figure 5.6 shows a black box view of which other systems Hybrid OSN communicates with via interfaces. The systems are:

- Twitter API
- GUN
- IPFS via Infura
- User

Infura[22] is a service that provides access to Ethereum and IPFS via a simple interface. Communication with the API happens using HTTP requests. The connection of IPFS in Hybrid OSN can thus be carried out in a simple way. The use of an additional system entails an extra risk typically. However, there is a JavaScript client for IPFS, which can be integrated into Hybrid OSN and thus the dependency on Infura would be omitted. For the creation of the prototype, the decision was made to use Infura for reasons of simplicity. Infura can be used for IPFS free of charge and without registration.

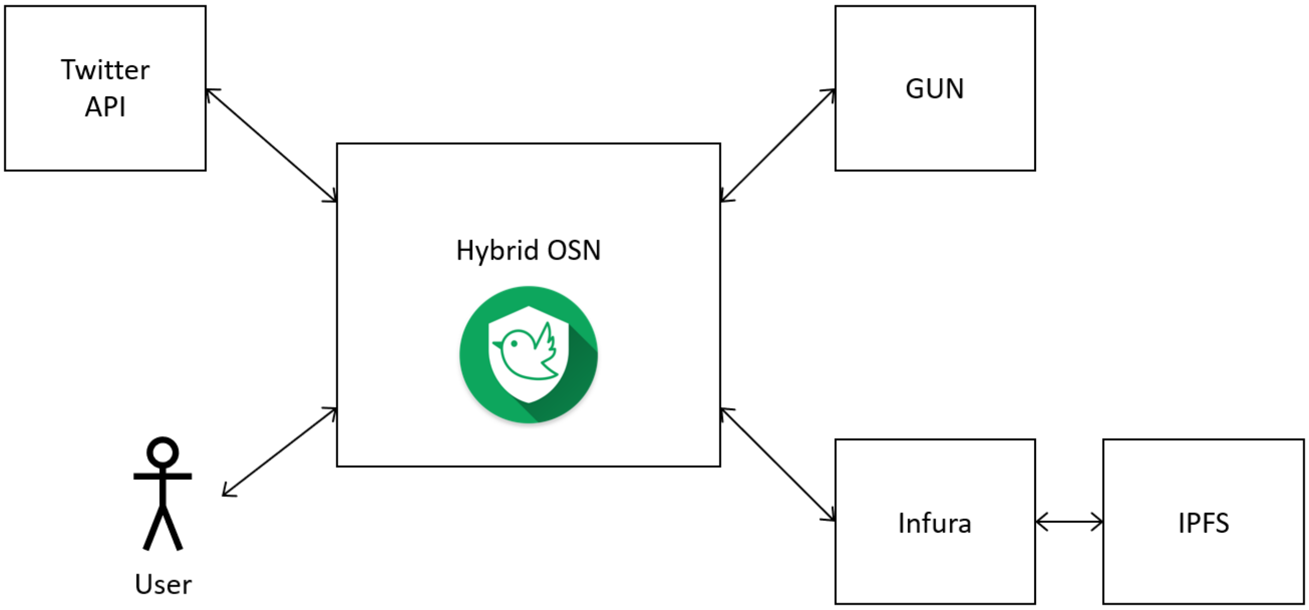


Figure 5.6: Black box view on Hybrid OSN showing the scope and context of the application. The adjacent systems are the user of the app, the Twitter API, the P2P database GUN, and IPFS via the Infura gateway.

5.6.2 White Box View

The Ionic framework uses Angular in the core. Accordingly, the Hybrid OSN app is in principle an Angular application. The essential building blocks are components, pages, and providers

22 https://infura.io/

(see Figure 5.7). In the following, these components are described in detail and examples are given of where they are used in Hybrid OSN.

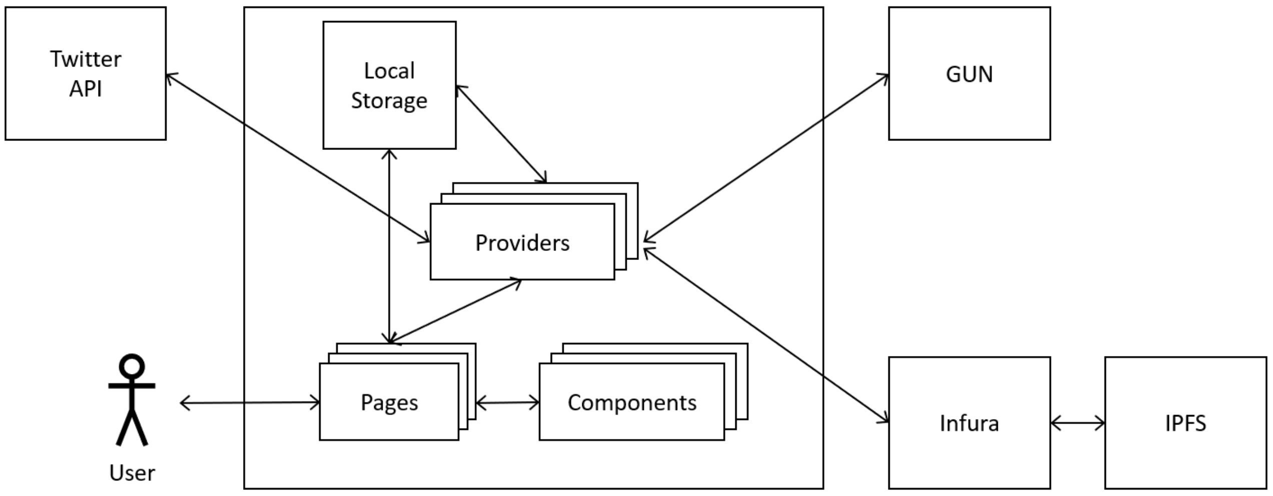


Figure 5.7: White box view on the basic building blocks and their connections. The user always faces and interacts with pages. These pages are built using components. Providers handle all data and communicate with the interfaces of other systems like GUN, the Twitter API, and Infura.

Providers

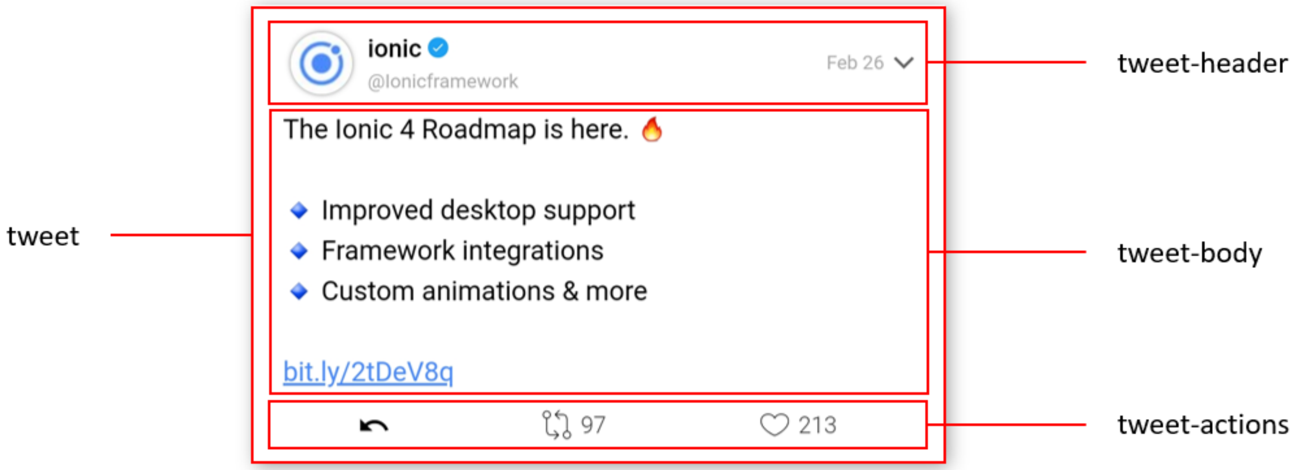
Data access is performed using providers (known as services in Angular). For the external services (Twitter API, P2P database, P2P storage), there is one provider each to handle the communication. Besides, providers are used as helper classes providing specific functionality that is used several times. This functionality includes, for example, encryption and decryption and the compilation of aggregated timelines. Providers are injected into components using the constructor. Table 5.1 lists all providers used in Hybrid OSN and their functional descriptions.

Provider	**Purpose**
Auth	Manage and perform authentication against the Twitter API. Responsible for login and logout.
Crypto	Provides methods for encryption, decryption, and key generation
Feed	Aggregation of private (P2P) and public (Twitter) tweets to compose a chronological timeline
P2P-Database-Gun	Interface for data exchange with GUN
P2P-Storage-IPFS	Interface for data exchange with IPFS via Infura
Twitter-API	Interface to use the Twitter API using the Twit package

Table 5.1: Providers used in the Hybrid OSN app in alphabetical order with their purpose.

Components

Components are the basic building blocks of a user interface. Figure 5.8 shows an example of the representation of a tweet in Hybrid OSN using various components. A component consists of an HTML template, CSS styling, and JavaScript logic, whereby the logic is typically limited to a minimum. Components can be used as elements in other components or pages. A component receives the data it is supposed to visualize. Furthermore, components can process events or return them to parent components for handling.


ionic
@Ionicframework
Feb 26

The Ionic 4 Roadmap is here. 🔥

- Improved desktop support
- Framework integrations
- Custom animations & more

bit.ly/2tDeV8q

Figure 5.8: Composition of the tweet component from three other components. Several tweet components are in turn combined to form a feed component.

Pages

Pages are specialized components that are used as a holistic view. A page is made up of several other components. The data to be displayed are loaded using providers. To be able to navigate between the individual pages within the app, the model of a stack is used (implemented by the NavController). The currently visible page is at the top of the stack. When another page is called, it is pushed onto the stack. Pressing „Back“ removes the top page from the stack and displays the page below it.

Table 5.2 lists all pages and their purpose. When the app is opened, it checks whether the user is already logged in. Depending on this, the user starts with the Login Page or the Home Page.

Page	Purpose
About	Information about the app, which can be accessed via the login page to get basic information about the app before logging in
Home	Chronological view of the latest tweets from Twitter and the private network
Login	Authentication against Twitter to use the Twitter API
Profile	Presentation of a user profile consisting of the user data (profile picture, brief description, location, website) and the user timeline
Search	Container page for searching for tweets and users, where tweets are also divided into popular and recent (see Search-Results-Tweets-Tab)
Search-Results-Tweets-Popular	Search results of currently popular tweets for a given keyword
Search-Results-Tweets-Recent	Search results of recent tweets for a given keyword
Search-Results-Tweets-Tab	Container page for the search results for tweets (recent and popular) in tabs
Search-Results-Users	Search results of users for a given keyword
Settings	Configuration of keywords that trigger the private mode and settings regarding encryption
Write-Tweet	Form for writing a tweet

Table 5.2: Pages used in the Hybrid OSN app in alphabetical order with their purpose.

Local Storage

As the name suggests, this is a local storage that is accessible by the app. With Hybrid OSN, this memory is used to store essential information for usage. These include the Twitter user id, the two tokens for accessing the Twitter API, the keywords that trigger the private mode, and private and public keys for encryption. Log out completely deletes the local storage.

5.7 Runtime View

This section describes particular processes and relationships between building blocks for two selected scenarios. The first is to write and post a tweet and the second is to display the home timeline. In addition to the description, the two scenarios are also documented by flowcharts.

The goal is to make clear how the building blocks of the system perform their respective tasks and communicate with each other at runtime. The two scenarios were therefore deliberately selected because of their particular importance and require special documentation due to their complexity.

5.7.1 Post a Tweet

On the write-tweet page, new tweets can be written and posted using a form. In addition to the input field for the message text, there is also a status display that informs the user about the character limit, a switch that decides about the target network, and a button to send. The process of writing and publishing a tweet is shown in the flow chart in Figure 5.9. Figure 5.10 shows the involved components in the block view.

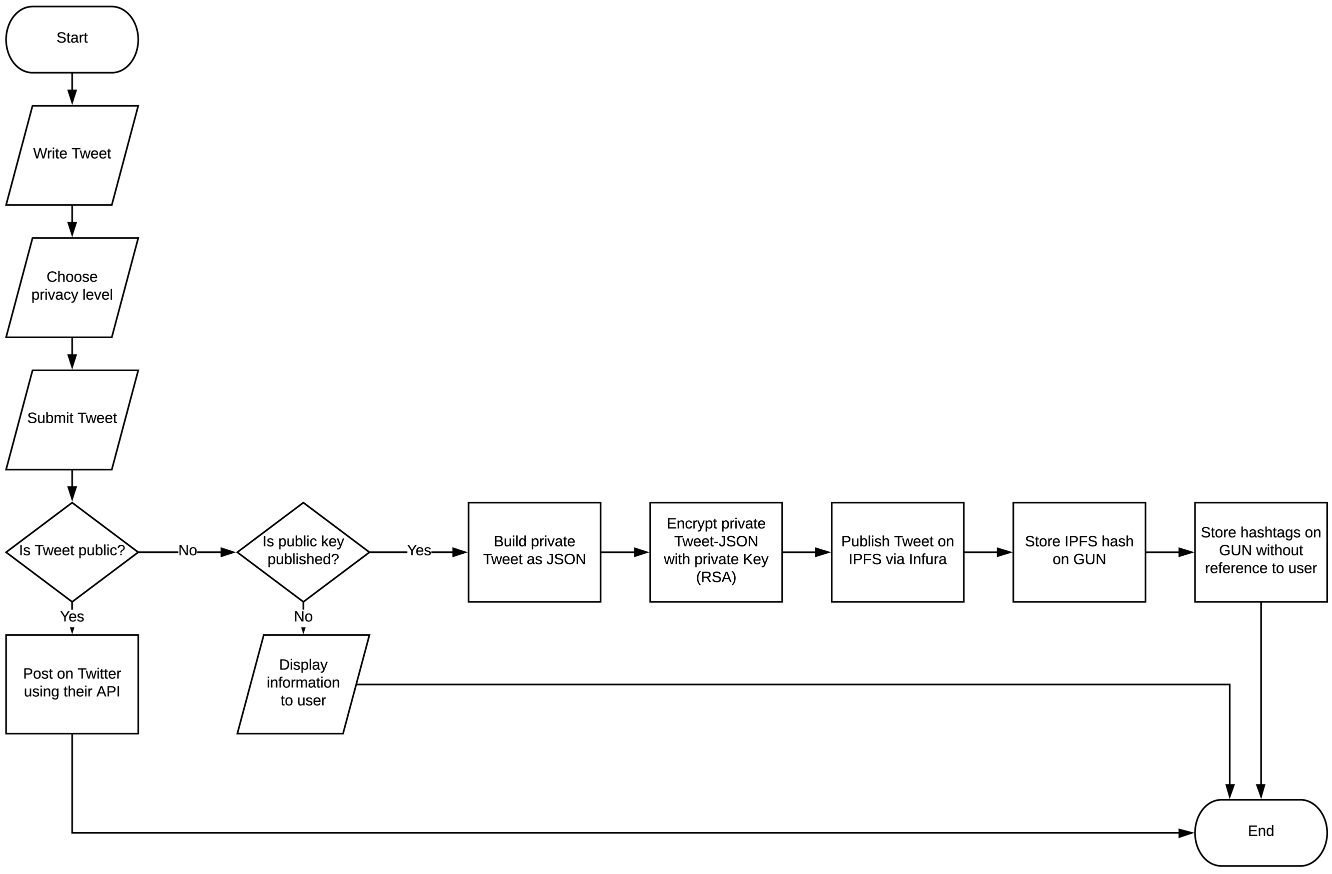

Start
Write Tweet
Choose privacy level
Submit Tweet
Is Tweet public?
No
Is public key published?
Yes
Build private Tweet as JSON
Encrypt private Tweet-JSON with private Key (RSA)
Publish Tweet on IPFS via Infura
Store IPFS hash on GUN
Store hashtags on GUN without reference to user
Yes
Post on Twitter using their API
No
Display information to user
End

Figure 5.9: Flow chart of the process for posting a new tweet either on the public Twitter network or on the private P2P network

After entering the message in the input field, determining the destination network, and pressing the „TWEET!“ button, processing starts in the WriteTweetPage class. If the message is destined for Twitter, the Twitter API provider sends an HTTP POST request with the data to the `statuses/update`[23] interface. In this case, nothing more needs to be done, as the Twitter API takes over the preparation of the data and extracts, for example, hashtags, mentions, and URLs.

When publishing in the private network, the system first checks whether the public key has already been published. The Crypto provider performs this check using the Twitter API provider (for further information about the handling of public keys see the section about security 5.8). If the public key has not yet been published, the user receives a warning, and the posting process is aborted. Otherwise, the private tweet will be constructed. The entered text is converted into

[23] https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/post-statuses-update

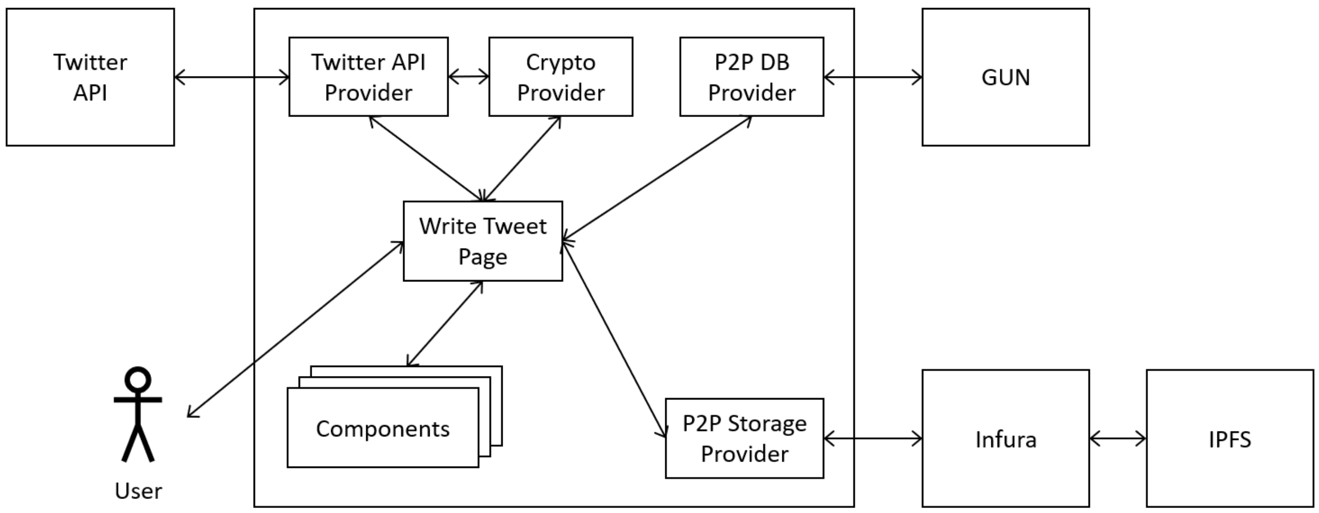


Figure 5.10: Building block view of the interaction between the different modules when posting a new tweet

a simplified tweet object (see Twitter documentation for original tweet object[24]) that contains the essential information.

Beside the message (`full_text`) the Twitter user id (`user_id`) and the timestamp (`created_at`) are set. In addition, there is a flag (`private_tweet: true`) to distinguish the private tweet, which later influences the design. The `display_text_range` indicates the beginning and end of the relevant part in `full_text`. For private tweets, this is always the entire text, for tweets from the Twitter API an additional, not needed URL may be appended, which is cut off by clipping. Furthermore, the tweet entities are extracted. The extraction includes URLs, hashtags and user mentions. An example of a private tweet is shown in Listing 5.1.

Listing 5.1: Private Tweet in JSON format

```
{
  "full_text": "Hello␣@twitter␣#hi",
  "user_id": "3237049834",
  "created_at": 1552290414930,
  "private_tweet": true,
  "display_text_range": [0, 18],
  "entities": {
    "hashtags": [
      {
        "hashtag": "hi",
        "indices": [15, 18]
      }
    ],
    "urls": [],
    "user_mentions": [
      {
        "indices": [6, 14],
        "id_str": "783214",
        "screen_name": "twitter"
      }
```

[24] https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html

```
    ]
  }
}
```

The crypto provider performs the encryption of the private tweet data. For asymmetrical encryption, the RSA algorithm is used. The P2P storage provider sends the encrypted data via an HTTP POST request to Infura for storage in IPFS. The response contains the hash which addresses the data in IPFS. This hash is stored in GUN together with the timestamp and the author's Twitter user id. For saving to GUN, the P2P DB provider is used. Besides, the previously extracted hashtags with the timestamp are also stored in GUN with the same provider so that the data in the dashboard are accessible to the service provider without having to conclude individual users.

5.7.2 Load the Home Timeline

When opening the home page, the logged in user gets the latest tweets of the accounts he follows chronologically presented. The tweets are loaded in batches of 20 tweets from the Twitter API and enriched with the private tweets for this period. If the user scrolls to the end of the feed, the reloading of the next batch is triggered and automatically inserted at the end. At the top of the feed, a „pull to refresh" action intents the feed reloading. The loading process is shown in Figure 5.11 as a flow chart and in Figure 5.12 as a building block view of the interacting components.

The starting point is the home page, which is accessed by the user. Several components display the data obtained from the feed provider. Using the Twitter API provider, the feed provider loads the latest 20 timeline tweets via the corresponding interface (`statuses/home_timeline`[25]) via an HTTP GET request.

The next step is to load the private tweets that match the period marked by the current timestamp and timestamp of the 20th (the oldest) tweet. Furthermore, the ids of the accounts the user follows (so-called friends) are required. These must initially be requested from the Twitter API (`friends/list`[26]) via the Twitter Feed provider using an HTTP GET request. The loaded user ids are cached in order to keep the number of requests to the Twitter API to a minimum for later loading processes. For each user id, a lookup for private tweets in the given period is performed. The P2P Database (DB) provider queries GUN. If there are private tweets, the hashes for IPFS are returned together with the `created_at` timestamp. If no private tweets are available for the period, the feed provider returns the data of the public tweets to the home page. Otherwise, the private tweets are loaded and decrypted. First, the P2P storage provider is used to load the data behind the hash addresses from IPFS via Infura. For this purpose, the hash is transferred to Infura with an HTTP GET request, and the data are received from IPFS as the response. The author's public key, which can be obtained from the user's public key history, is needed for decryption. The public key history is loaded and decrypted via the

[25] https://developer.twitter.com/en/docs/tweets/timelines/yapi-reference/get-statuses-home_timeline.html
[26] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-list

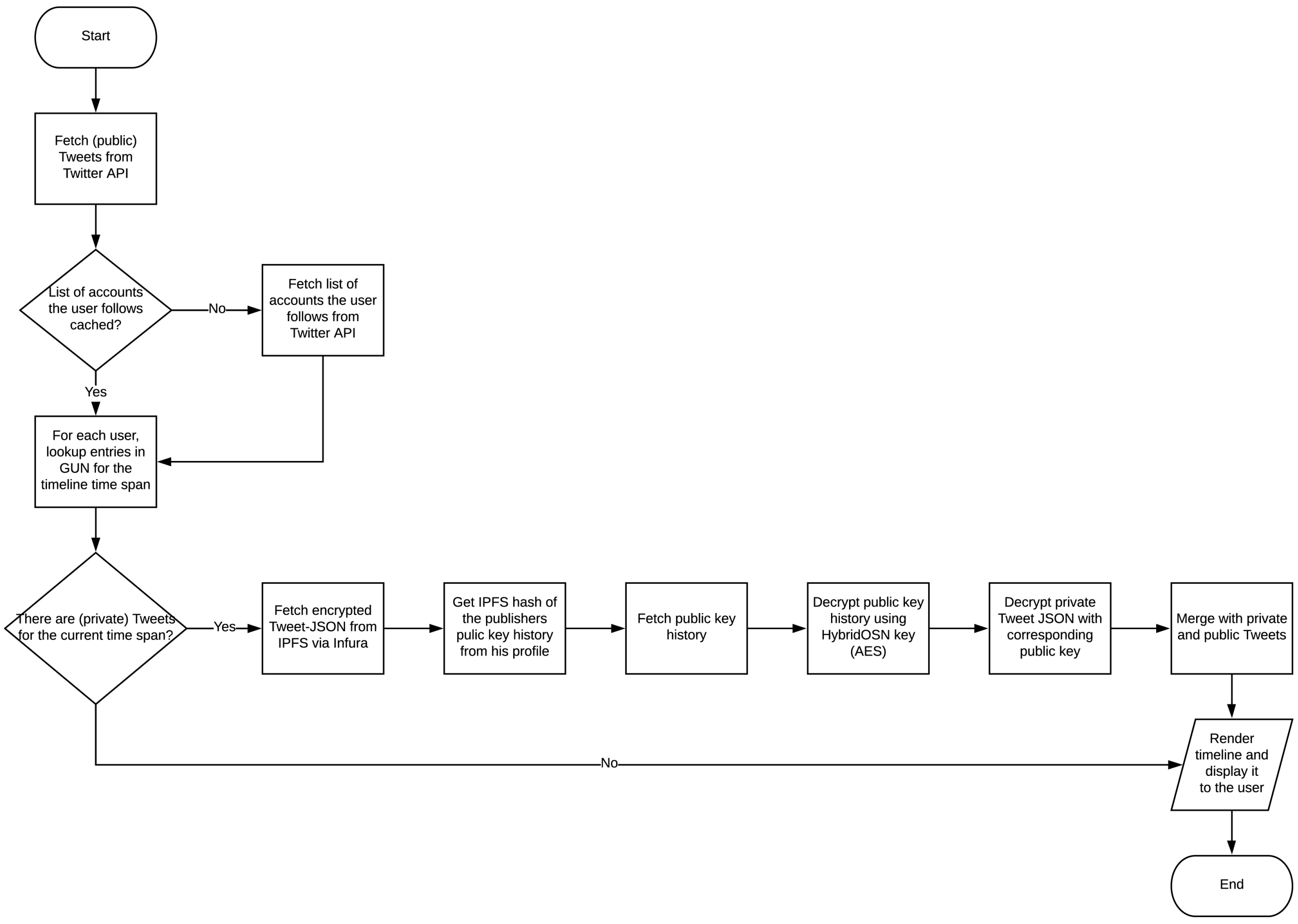

Start
Fetch (public) Tweets from Twitter API
List of accounts the user follows cached?
No
Fetch list of accounts the user follows from Twitter API
Yes
For each user, lookup entries in GUN for the timeline time span
There are (private) Tweets for the current time span?
Yes
Fetch encrypted Tweet-JSON from IPFS via Infura
Get IPFS hash of the publishers pulic key history from his profile
Fetch public key history
Decrypt public key history using HybridOSN key (AES)
Decrypt private Tweet JSON with corresponding public key
Merge with private and public Tweets
No
Render timeline and display it to the user
End

Figure 5.11: Flow chart of the process for loading the home timeline for a user

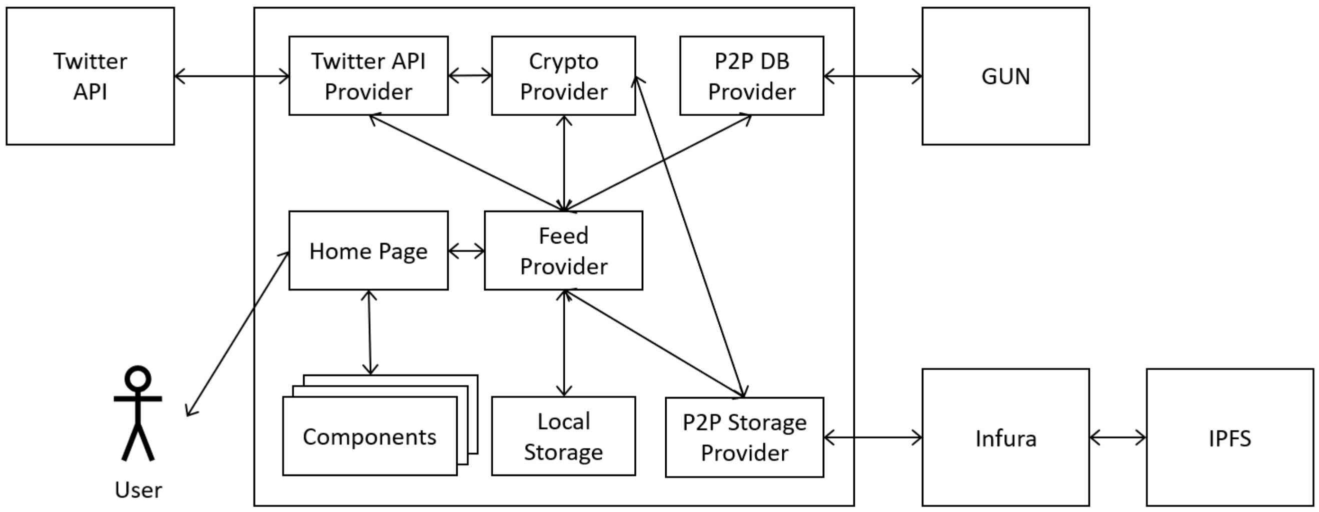


Figure 5.12: Building block view of the interaction between the different modules when loading the home timeline of a user

crypto provider, which in turn uses the Twitter API provider. Afterwards, the private tweet is decrypted.

Finally, the private and public tweets are merged and sorted according to their `created_at` timestamp in descending order. This data are returned to the home page. If the user triggers a reload by reaching the end of the feed or by „pull to refresh“, the previously described process starts again.

5.8 Security

Usually, asymmetric or a combination of asymmetric and symmetric encryption methods are used for secure message exchange. The advantage of asymmetric encryption methods is that messages can be encrypted with the known public key and then only the owner of the private key can decrypt them. It is not possible to determine the private key from the public key within a reasonable time. However, asymmetric encryption methods are more computationally intensive and therefore more time-consuming than symmetric encryption methods. For this reason, a combination of both methods can be used to exchange a symmetric key using asymmetric encryption. The asymmetric encryption ensures that only the two parties are aware of the symmetric key, which can then be used for symmetric encryption of the communication.

Tweets are usually posted publicly on Twitter. Only those who explicitly set their profile to „private“ can decide whom they allow as followers and thus recipients of their tweets. This type of publication and visibility should also apply to private networks. Since it is unclear who is the recipient of a private tweet, it is not possible to encrypt the message with the recipient’s public keys.

The following three requirements apply to encryption:

1. The author is verifiable. It is not possible to distribute tweets on behalf of another user on the P2P network.
2. A private tweet should have the same visibility to other users as a standard tweet on Twitter.
3. The service provider (Twitter) must not be able to decrypt private tweets or associate them with a user.

Concrete actions can be concluded to meet these requirements:

1. Private tweets must be signed or asymmetrically encrypted so that the author is identifiable.
2. Distribution of the public key for decryption must take place via the user’s profile. The profile is the only place guaranteeing that only authorized users can access the key and that the public key belongs to a specific user without any doubt.
3. The Hybrid OSN application must encrypt the public keys so that Twitter cannot read them and therefore cannot decrypt the private tweets.

5.8.1 Realization

In the app settings, an asymmetric key pair can be stored or generated, which is used to encrypt the private tweets. The RSA-OAEP algorithm is used here. Furthermore, by clicking on a particular button, the public key is published. The new key together with the current timestamp is recorded in the public key history of the user and stored in IPFS. Listing 5.2 shows an example for the public key history. In JSON format, public key and validity start are stored in an array. The address at which the public key history can be retrieved is posted as a regular tweet in the user's timeline during publication. So that this tweet can be found quickly and easily; the id of this tweet is saved in the profile description of the user.

Listing 5.2: Public key history in JSON format. The file is symmetrically encrypted before storing on IPFS.

```
{
    "keys":[
        {
            "key":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNZiscdaUuJ8pUFkPkTAh6oq...",
            "validFrom":1548591435051
        },
        {
            "key":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+14hMkrRfSwCTfbmkS+m6MQ...",
            "validFrom":1548193838409
        },
        {
            "key":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC5rYHZljdPXozCNCX86N8CTY...",
            "validFrom":1546382473017
        },
        {
            "key":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjssLPklHgNX/KyPACLOQAmB...",
            "validFrom":1545945585132
        }
    ]
}
```

By saving this information on the user's profile, it is ensured that only users who can read the user's regular tweets have access to the user's public keys and hence to his private tweets. The history allows the changing of the key pair and still ensure that older private tweets are still decryptable.

Since Twitter has access to all the data stored on its servers, it can also find the link to the public key history. Therefore, it is necessary to prevent Twitter from decrypting the private tweets of the user by retrieving the public key history. For this reason, the public key history is additionally encrypted symmetrically with the AES-Galois/Counter Mode (GCM) algorithm. The key is stored in the Hybrid OSN app and therefore unknown to Twitter.

When writing a new, private tweet, the system checks whether the public key has been published before posting. Only if this is fulfilled, the private tweet will be encrypted with the private key and posted.

5.9 Summary

In this chapter, we introduced the Hybrid OSN prototype for Twitter. First, the objectives were set, then, the selection of the OSN was discussed. The comprehensive API and the comparatively simple functionality led to a decision for Twitter. Afterwards, we examined various technologies for their suitability. IPFS was used to store the data and GUN as a P2P database. Also, we present a dashboard showing popular hashtags in the private network. Finally, the architecture of the application was documented, and mechanisms were introduced to protect the private data.

6 Evaluation

This chapter is about the evaluation of the Hybrid OSN app. It is discussed to which extent the previously defined requirements and quality goals were fulfilled and validated how realistic the requirements are in general. Afterwards, limitations are discussed, and a threat model mentions potential problems.

6.1 Fulfillment of Requirements and Achievement of Objectives

In Chapter 4.2, several functional and non-functional requirements were defined. In addition, in Chapter 4.4 multiple quality goals were set to ensure a high quality of the code. In the following, for each requirement and goal, a critical discussion is given on the achievement in the Hybrid OSN app.

6.1.1 Functional Requirements

With regard to the functional requirements previously defined in Chapter 4.2, their implementation in the Hybrid OSN app are discussed in the following.

Standard Functionality

As part of this work, a prototype with limited functionality was created. Accordingly, numerous functionalities, such as the direct messaging system, notifications, the posting of images, gifs, videos or surveys, and much more are not implemented. However, it was considered that the prototype implements all essential functionalities so that the basic use of Twitter is possible. These basic functions include displaying the home feed and user feeds (profiles), searching for users and managing the connection (follow, unfollow, mute, block), as well as liking tweets and posting new tweets (incl. reply, retweet).

While the majority of the missing functions were deliberately omitted for time reasons, the restricted API also sets limits. For this reason, this requirement must be evaluated as not fulfilled. But, due to the limitations of the API, it could not have been fully met. While Twitter is an extremely grateful example due to its simple functionality and generous API, the limitations of other OSNs are much more severe. A missing API and crawling prohibited by terms of service would make data exchange between a client and the OSN virtually impossible.

Client-side Solution

Since it is not possible to execute code on the Twitter's servers, no other solution than a client-side approach is possible. Accordingly, all functions are implemented on the client side, and the requirement is fulfilled.

As described in Chapter 4.5, a solution architecture either contains or does not contain additional servers. In the objectives in Chapter 5.1, a solution without additional servers was preferred. Though, since GUN requires a relay server to establish the connection between the peers, this self-imposed goal could not be achieved.

Data Sovereignty

The user can decide for himself whether the data are shared with other users via the private network or the Twitter servers when posting or liking a tweet. Since other functionalities that require data from the user (e.g., information in the profile, direct messages) have not been implemented, at least the prototype fulfills the requirement partly.

Nevertheless, data about user behavior can still be collected. The user is authenticated against the API so that his requests can be unambiguously assigned. Using this usage data, Twitter can record which profiles are called up and what the user searches for. This data can also be used to conclude a user's preferences and interests. Hybrid OSN offers no protection against this type of data collection.

Authorized Data Access and Encryption

Private data should be accessible only to those users for whom they are intended. But, they should not be accessible to the service provider. Twitter only distinguishes between public and private profiles. While with public profiles every other Twitter user has access to the profile and the tweets, with private profiles only authorized followers can view tweets. By distributing the public key to decrypt the data via the profile, the same user group is granted access to a user's private tweets, which can also see the official Twitter tweets. The additional symmetric encryption of the public key history by the Hybrid OSN app ensures that Twitter cannot decrypt the public key history and thus also the private tweets. Therefore, the requirement can be evaluated as fulfilled.

The implementation differs from the standard solution for this problem. Typically, data are encrypted with the recipient's public key to ensure that only the recipient can decrypt it with their private key. Though, in the case of Twitter and the public profiles, the recipient circle of a tweet is not known. Accordingly, it is not possible to explicitly encrypt data for a specific recipient.

An alternative to the implementation in Hybrid OSN would be to encrypt data with the symmetrical Hybrid OSN key for public profiles and to encrypt data asymmetrically with the public keys of the recipient for private profiles. The advantage would be that users with a public profile would not have to generate keys and the configuration effort would be reduced. A simpler configuration would have a positive effect on the requirement for minimal configuration effort. The disadvantage would be that a user with a private profile and a large number of approved followers would have to calculate the encryption and upload the data for each. This would be in contrast to the requirement to conserve resources.

Flexible Data Format

The flexibility of GUN and the JSON data format allow effective data storage. The established structures can also be easily adapted for future requirements. Due to the immutability of the data stored in IPFS, it is not possible to retroactively change the data structure. Should this nevertheless be necessary, the changes would have to be implemented in the program logic. However, the requirement must be regarded as fulfilled.

Platform Independence

The requirement for a platform-independent solution contradicts the actual requirement to create an Android client. Nevertheless, the requirement was taken into account in the decisions made. The targeted selection of the Ionic framework laid the basis for platform independence. Even though Ionic was primarily used to create an Android application within the scope of this work, it should be possible to use the same code with some effort to create an app for iOS or a PWA. Since Twitter can be used on almost all platforms up to the feature phone, the idealistic goal of creating a similar client variety is almost utopian. Taking all that into account, the requirement is therefore only partially fulfilled.

Anonymized Data for the Service Provider

Anonymous data from the private network should be shared with Twitter to prevent the service provider's business model from failing. Approximately 87% ($ 791 million) of Twitter's revenues in the fourth quarter 2018 were generated by advertisements [71]. Thus, the central pillar of Twitter's business model is personalized advertising. Accordingly, data should be of particular interest for Twitter if it is related to a person. Only if Twitter gains knowledge about a user, this knowledge can be used to place targeted advertisements. However, it is precisely this connection between data and user that should be avoided by using Hybrid OSN.

As a result, the implementation of the requirement is almost impossible. Even though all data would be transferred to Twitter in an anonymous form for further processing, this data would be entirely worthless for Twitter's business model. In this context, it should be emphasized that the use of the API does not result in any advertising being displayed to the user. Twitter therefore deliberately accepts that client applications that use the API do not contribute to the generation of profits. Against this background, the implementation of this requirement is questionable although it is an essential aspect in the hybrid OSN concept.

In order to meet the request and at least to prove the feasibility of the anonymized data provision, the hashtags used were provided anonymously. Twitter could use this data to improve trend recognition and inform users about current, much-discussed topics. Thus, a full analysis of popular hashtags throughout the system enables precise statements.

6.1.2 Non-Functional Requirements

The following part is about the non-functional requirements as defined in Chapter 4.2. It is discussed to which extent they are considered in the Hybrid OSN prototype.

Minimal Additional Effort

The Hybrid OSN app is available as APK file and can be installed on an Android device. Since it is only a prototype so far, the app is not available from the Google Play Store. Theoretically, this is possible and would simplify the installation and updating of the app.

The app works out of the box and does not require any particular configuration. For passive use, i.e., just consuming the content, the app works from the beginning. No additional registration is necessary; the Twitter login is sufficient. For the encryption of the private tweets, a key pair is required, which can be generated in the settings by clicking on a button. The configuration effort is therefore limited to a minimum. If the key pair is not yet stored, the user is notified before posting the private tweet and referred to the necessary configuration on the settings page.

If the user uninstalls the app or logs out of his account, the configurations and thus also the public and private key are deleted. If the keys have not been saved elsewhere and cannot be restored, the next time the user uses Hybrid OSN again, there are no restrictions on generating a new key pair. With the public key history, it is verified that old tweets are always decryptable.

During the conception phase, this requirement was always given high priority. The implementation as a dApp and the associated need for corresponding tokens and a wallet were especially rejected because of this requirement. The current implementation completely meets the requirement for minimal configuration effort.

Minimal Side Effects

In the context of the publication of the public key history, changes that are visible to other users become noticeable in two places. First, the IPFS hash of the public key history is posted in a tweet. Second, a reference to this tweet is stored in the profile description of the user. While the tweet will show up in all followers' timelines, a change in the profile description has no direct influence on the user experience of other users. During regular use, the public key history is rarely extended by new entries. Thus the side effects are limited to a minimum, and the requirement is fulfilled accordingly.

Compliance with Policies

The Twitter Developer Terms and the Twitter Terms and Conditions were respected. For example, API guidelines do not allow the redistribution of Twitter content. For this reason, when tweeting or quoting tweets over the private network, only their ids are stored, not the entire content. Hence, the requirement is fulfilled.

Good User Experience

A good user experience should be achieved by short loading times and understandable and appealing design. The loading times of the private data in Hybrid OSN consist of two parts. First, objects of interest must be identified with GUN. Then, the data has to be loaded from IPFS. In both cases, it has a significant influence on the loading time whether there are enough peers with the searched information in the network. Both systems are designed for short reaction times. However, the loading times cannot be given in figures, since no reliable result could be measured due to the non-reproducible circumstances concerning peer availability.

There are no objective standards for an understandable and appealing design. The following examples are intended to show how these demands on the user interface were nevertheless met.

- Icons extend items in the menu. The icons allow quick identification of the purpose of the page without having to read the title.
- The structure of a tweet corresponds to the structure known from Twitter so that no acclimatization is necessary.
- When writing a tweet, a circle animation visualizes compliance with the character limit. The user is thus visually informed about how many characters have already been used and how many are still possible.
- When sharing data, the private mode is indicated by a sunglasses icon, the public mode by the Twitter logo. This can be found when writing a new tweet as well as when likening tweets.
- Private tweets are represented by a dark background so that the data source can be identified easily.
- Loading processes are represented to the user by a loading animation. When posting a private tweet, the user is also informed about the current processing step.
- When using sensitive keywords in tweets, the user is informed about this circumstance and recommended to post the tweet privately. On the settings page, the user can configure the keywords himself.

6.1.3 Quality Goals

Quality goals, as defined in Chapter 4.4, should motivate developers to write high-quality code that is easy to customize and maintain. Concerning the analyzability of Hybrid OSN, all modules, classes, methods, and variables were titled with English names. The public methods of the provider classes were documented, and the Clean Code principles were adhered to.

With JavaScript, the application is written in a language that ranks on eighth place in the TIOBE Index, which measures the popularity of a programming language [20]. At GitHub, JavaScript leads all statistics [34]. Most new repositories are created for JavaScript projects, and most contributions are written in JavaScript. Due to its popularity, many developers should be able to use the Hybrid OSN codebase and find their way around quickly. The interfaces to GUN and IPFS have been outsourced to individual providers so that they are interchangeable. Thus, the technology can be easily exchanged.

Tests are used to check the proper functioning of the application. For time reasons, this quality goal was neglected for Hybrid OSN and thus not achieved. But, in principle, the testing of ionic applications is possible.

The source code should be managed as an open source project to increase confidence in the application. Although version management is used with Git and is also centrally managed on the Git server of the Telecooperation Lab group, the code is not freely accessible here. Thus Hybrid OSN cannot be considered an open source application.

6.2 Limitations

By using the Twitter API, Hybrid OSN is very strongly bound to Twitter. Restrictions to the API would have a significant impact on the Hybrid OSN client. In order to link content and actions to users and tweets, a referencing of the respective id is necessary. If these ids disappear from the system because tweets are deleted or users leave the platform, the private data can no longer be assigned and lose their significance.

Private tweets are loaded into the user's timeline by active pulling. With the user ids of the friends' accounts, their private tweets are looked up. Since the data structure in GUN is optimized for searching by user ids, searching for a keyword or hashtag is not possible in an elegant way. To perform such a search, first, all hashes need to be extracted from GUN and downloaded form IPFS. Second, all this data needs to be decrypted and searched locally for the given keyword. For a small amount of data, this is maybe practical. However, it does not scale and is, therefore, no permanent option.

For the hybrid client to achieve the best possible result, a copy of all the functionalities of the original OSN must be implemented decentrally. In particular, the problem of finding specific content has to be solved. GUN fulfills this requirement only partly since only user ids can be looked up. Besides, GUN needs a relay server to connect peers. This server represents an unwanted single point of failure.

Since there is currently no pushing from the back end or periodically updating, push notifications are missing. The user always has to open the app and check for updates himself. If someone else, who is no friend of the user, mentions the user in a private tweet, there is no possibility he will ever notice.

The data stored in IPFS cannot be deleted and subsequently edited. Also in the Hybrid OSN app, the deletion of single private tweets is not implemented at the moment. The users of Peepeth strongly criticized this circumstance [25], hence this is also a limitation for the Hybrid OSN users. While tweets cannot be edited afterward, they can be deleted so that there is a difference to the hybrid implementation.

Data exchange via GUN only works if peers are online at the same time. As Hybrid OSN is an application for a smartphone, a permanent connection is not guaranteed. Furthermore, the connection is not maintained in the background, but only when actively using the application. Especially in the initial phase with few users, this is a substantial limitation. Super peers could solve this problem, but this requires the installation of more servers.

6.3 Threat Model

In the threat model of Hybrid OSN the potential threats for different sub-areas are shown, and the particular risk discussed. The worst would be if private data could be decrypted and assigned to a user or if identity abuse would be possible. However, other dangers such as identification by the service provider or manipulation of data must be analyzed.

6.3.1 Service Provider – Twitter

Hybrid OSN users can be easily identified by the service provider Twitter, even if they only use the app passively to read private tweets of other users and do not write private tweets themselves. For using the Twitter API, it is essential to register an app to get an app token. This app token is attached to all requests sent to the Twitter API. When logging in on Hybrid OSN for the first time, the user accepts to use the app to access Twitter.

So far not implemented, but theoretically possible is that each user creates an app for the use of the API on their own. The obtained app token could then be stored in the Hybrid OSN app, and the use of the application could be obscured. In this case, the identification possibility via the Hybrid OSN app token is omitted, and the passive use would be possible without danger. However, the Twitter developer terms forbid the use of multiple applications for a single use case [68]. This restriction is primary for a single developer trying to bypass the request limits. It has to be further evaluated if this rule also applies to multiple developers with only one application each.

Active use requires a public tweet and a reference in the profile description for the distribution of the public key history. Although the contents are inconspicuous, they are still sufficient for the identification of a Hybrid OSN user.

6.3.2 GUN

In Hybrid OSN, GUN takes the role of a database shared by the peers. The dashboard also establishes a direct connection. The data are publicly accessible and editable. The stored data are a combination of hashtag and timestamp, which serve as information for the trends in the Hybrid OSN dashboard. For every private tweet of a user, there is an entry consisting of Twitter user id, IPFS address hash, and timestamp. Also, there are the private likes, for which there is a counter to the tweet id. For preventing the hashtag and private tweet timestamps from connecting, the time of the hashtag timestamp is set to 00:01. The trends in the dashboard are aggregated by the day, so the exact time is not essential.

The greatest threat is that an attacker may modify or delete data. By deleting entries, private tweets would no longer be found and thus no longer displayed. Changing the IPFS hash would mean that the data could not be found and would also not be displayed. Manipulation of the timestamp would result in private tweets being loaded at the wrong time interval when the feed is loaded and thus positioned at the wrong place. Furthermore, the timestamp in GUN is used to use the appropriate public key from the public key history for decryption. Under certain circumstances, the wrong key would be selected and the private tweet could not be decrypted.

Creating entries for private tweets does not have a significant effect because the associated content stored in IPFS must be encrypted with the private key, which is unknown to a third party. Adding wrong or modifying existing hashtag entries for trend detection is also possible and poses a significant risk as it allows manipulation of the trends. Ultimately, it is not possible to verify which hashtags were used and how often. The same applies to private likes. Since in this case the complete information is stored in GUN and can be changed, it is not possible to determine whether data has been manipulated.

6.3.3 IPFS

Since IPFS is publicly accessible, anyone can add and retrieve data. Though, it is not possible to change or delete data. A hash of the content addresses the data stored in IPFS. Since the content is entirely unknown (especially by encrypting the plaintext content), it is not possible to conclude the hash. A targeted search for private data in IPFS is therefore impossible. The encrypted data also does not contain any clues that allow conclusions to be drawn about Hybrid OSN. In combination with the publicly available information from GUN, all private tweet data could be found in IPFS. But, because of the encryption of the content, the data are worthless.

Due to the decentralization of the system, the availability of IPFS is always guaranteed. However, only as long as there are peers who make the service possible. If a peer leaves the network, its data are also lost if not reproduced beforehand. Therefore, there is no guarantee for the permanent availability of data.

6.3.4 Encryption – Leakage of Keys

On the one hand, the public key history is symmetrically encrypted; on the other hand, the private tweets are asymmetrically encrypted. The keys for asymmetric encryption are generated independently by each user and are therefore individual. With symmetric encryption, just one key is used, which is stored in the source code of Hybrid OSN. In this way, only the Hybrid OSN app can decrypt the public key history of a user and therefore decrypt its private tweets.

Disclosure of the source code would reveal the symmetric key. The service provider would then have all the necessary information and access to all data to read private tweets and assign them to users.

6.3.5 Authorized Access

A user's private tweets should be readable by all users who can also read the public tweets - except for the service provider Twitter. Therefore, a user posts the IPFS hash that leads to the public key history on his timeline. There is a threat that authorized users may create a copy of the decrypted public key history and pass it on to third parties. Since the data in IPFS is permanent and therefore not erasable, it can be decrypted at any time later with the appropriate public key.

If a user decides to change his profile to „private" in the account settings, the profile will no longer be publicly accessible. Solely accepted followers should then be able to read public and private tweets. A non-approved user is still able to fetch the encrypted private tweets from IPFS. However, since the link to the public key history is no longer accessible, the private tweets decryption is not possible. If non-approved users or third parties already have the link to or a backup of the public key history from the past, all private tweets of the past can still be decrypted. Whenever a profile is changed to „private" a new pair of keys should be generated to ensure that future private tweets are only readable to approved users. Otherwise the latest key is still valid and could be used to encrypt future private tweets.

7 Conclusions

In this chapter, we summarize the work that has been done in this thesis. Besides, we describe the main contributions for privacy protection in OSNs using a hybrid solution. Finally, we give an outlook on how the results of this thesis can be used for further analysis and research.

7.1 Summary

This thesis first introduced the problems around privacy protection in centralized, well-established OSNs, and the motivation for this work. Afterwards, we provided the relevant background information of software system architectures in general, and in particular for P2P networks as well as arising dApps. Then, we presented other approaches trying to increase the user's privacy. These approaches are extensions for established social networks, decentralized OSNs including dApp OSNs, and protocols for the communication in distributed networks. Based on the work of the „Research and Training Group“ and their research on „Privacy Protection via Hybrid Social Networks“, we elaborated a concept for hybrid solutions and defined multiple requirements. Besides, we gave various solution strategies for the implementation of the concept regarding the architecture and the client itself. Then, we presented our prototype for Twitter. First, we compared OSNs and technology and examined their suitability for a proof of concept. Second, we described the architecture and implementation of the Android app. Finally, we discussed to what extent the hybrid OSN prototype meets the previously defined requirements including our objectives, the functional and non-function requirements as well as the quality goals. We also analyzed limitations and provided a comprehensive threat model.

7.2 Contributions

In the work presented in this thesis, first, we took up the idea of a hybrid OSN from the Research and Training Group. This idea about an extension for secure and private data exchange in established OSNs was examined and enriched with precise requirements. These demands involve functional and non-functional requirements, as well as quality goals to ensure a good code quality when implementing. Furthermore, we evaluated the opinions and needs of different stakeholders and discussed restrictions. Conclusive, we presented possible solution strategies.

To prove the feasibility of the hybrid OSN concept, we created a unique prototype for Twitter as an Android app. This proof of concept uses the technologies GUN and IPFS to provide its users with the possibility of a secure data exchange while still using the default functionality of the OSN. We worked out a solution to save data in a flexible, extensible JSON format and protect it through the application of symmetrical and asymmetrical encryption algorithms. Further noticeable features include a user-friendly interface and the avoidance of side effects to other users caused by the use of Hybrid OSN. Since the need for configuration was kept on an absolute minimum, everyone is capable of protecting its data by using this app. A dashboard showing the trends in the private network was made to provide the service provider with anonymized data.

Finally, the evaluation of the prototype against the previously defined requirements demonstrated that the concept is feasible. However, it also became clear that not all requirements are completely fulfillable and the application of the concept to other OSN may be very challenging.

7.3 Future Work

The contributions and the presented results of this thesis provide several possibilities for future work. First, the Hybrid OSN prototype app can be further improved by adding more functionality, for example through implementing the direct message system.

While GUN solves the problem of a distributed database, it has various limitations and was furthermore identified as a weak point in the threat model. In future work, either the usage of GUN could be improved, or a better solution for a distributed database could be found and implemented.

Field studies of users using the Hybrid OSN app could be performed as for future work to validate user acceptance of the solution. Through analysis of the user behavior, the app could be optimized to encourage usage of the hybrid ONS app. Additionally, it could improve the concept of hybrid solutions.

Another topic for future work is the anonymized data sharing with the service provider. In-depth analysis of the service provider's demands and the available private date need to be carried out. It has to be evaluated which kind of anonymized data are worthy for the service provider.

In this thesis, the developed concept was validated for Twitter. Therefore, it would be an exciting challenge to apply the concept to other OSNs, like Facebook, Instagram or Snapchat. In this process, the stated requirements could be refined, and thus the overall concept improved.

Regarding the implementation of hybrid solutions in general, future work could examine how a framework providing the basic functionality of distributed P2P extensions may look like. Since they all need to store and retrieve data somehow, basic functions provided via a clean interface by a library could improve the creation of hybrid clients for all any platform.

Bibliography

[1] Android - Application Fundamentals. https://developer.android.com/guide/components/fundamentals.html. Online, accessed 22.03.2019.

[2] diaspora* federation protocol. https://diaspora.github.io/diaspora_federation/index.html. Online, accessed 21.03.2019.

[3] FAQ for users - diaspora* project wiki. https://wiki.diasporafoundation.org/FAQ_for_users. Online, accessed 21.03.2019.

[4] How far does a Wi-Fi Direct connection travel? https://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel. Online, accessed 21.03.2019.

[5] Juan Benet | LinkedIn. https://www.linkedin.com/in/jbenetcs. Online, accessed 22.03.2019.

[6] Kalman Graffi | LinkedIn. https://www.linkedin.com/in/kalman-graffi-24832812/. Online, accessed 21.03.2019.

[7] Magic Signatures - diaspora* federation protocol. https://diaspora.github.io/diaspora_federation/federation/magicsig.html. Online, accessed 21.03.2019.

[8] Peepeth | About. https://peepeth.com/about. Online, accessed 29.01.2019.

[9] Peepeth | crowdfunding. https://peepeth.com/a/crowdfunding. Online, accessed 29.01.2019.

[10] Peepeth | Free peeping! https://peepeth.com/a/free. Online, accessed 29.01.2019.

[11] Peepeth | Moderation. https://peepeth.com/a/moderation. Online, accessed 29.01.2019.

[12] Peepeth | Peep stats. https://peepeth.com/stats. Online, accessed 29.01.2019.

[13] Privacy Zuckering. https://darkpatterns.org/types-of-dark-pattern/privacy-zuckering. Online, accessed 22.03.2019.

[14] State of the DApps — 2,667 Projects Built on Ethereum, EOS & Steem. https://www.stateofthedapps.com/dapps/platform/ethereum?page=1. Online, accessed 21.03.2019.

[15] The Federation - a statistics hub | Diaspora* stats. https://the-federation.info/diaspora. Online, accessed 22.03.2019.

[16] The Federation - a statistics hub | Mastodon stats. https://the-federation.info/mastodon. Online, accessed 22.03.2019.

[17] Token Market Capitalizations | CoinMarketCap. https://coinmarketcap.com/tokens/. Online, accessed 21.03.2019.

[18] Eine neue Ära beginnt: diaspora plant mit Y Combinator den großen Neustart. https://www.foerderland.de/digitale-wirtschaft/netzwertig/news/artikel/eine-neue-aera-beginnt-diaspora-plant-mit-y-combinator-den-grossen-neustart/, May 2012. Online, accessed 21.03.2019.

[19] Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage. https://developers.google.com/web/showcase/2017/twitter, May 2017. Online, accessed 22.03.2019.

[20] TIOBE Index for March 2019. https://www.tiobe.com/tiobe-index/, March 2019. Online, accessed 22.03.2019.

[21] Mihai Alisie. Juan Benet, the creator of IPFS, joins the AKASHA Team as Advisor. https://akasha.org/blog/2016/05/31/juan-benet-the-creator-of-ipfs-joins-akasha, May 2016. Online, accessed 21.03.2019.

[22] Mihai Alisie. Unveiling AKASHA. https://akasha.org/blog/2016/05/03/unveiling-akasha, May 2016. Online, accessed 21.03.2019.

[23] Mihai Alisie. New Horizons. https://akasha.org/blog/2017/11/14/new-horizons/, November 2017. Online, accessed 21.03.2019.

[24] Mihai Alisie. AKASHA 2019: Metamorphosis Part I. https://akasha.org/blog/2019/01/15/akasha-2019-metamorphosis-part1, January 2019. Online, accessed 21.03.2019.

[25] Bevan Barton. Peepeth is now free! https://www.reddit.com/r/ethereum/comments/9p24mb/peepeth_is_now_free/, October 2018. Online, accessed 29.01.2019.

[26] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[27] Daniel Berger. Openbook: Zweiter Anlauf für die Facebook-Alternative - heise.de. https://www.heise.de/newsticker/meldung/Openbook-Zweiter-Anlauf-fuer-Facebook-Alternative-4142266.html, August 2018.

[28] Tim Berners-Lee. 30 Years On, What's Next #ForTheWeb? https://onezero.medium.com/30-years-on-whats-next-fortheweb-6ce844ed147f, March 2019. Online, accessed 21.03.2019.

[29] Pedro Canahuati. Keeping Passwords Secure. https://newsroom.fb.com/news/2019/03/keeping-passwords-secure/, March 2019. Online, accessed 22.03.2019.

[30] Andrew B. Coathup. Evolution of decentralised social media. https://medium.com/coinmonks/evolution-of-decentralised-social-media-dfe567d23e54, May 2018. Online, accessed 29.01.2019.

[31] Maxwell Salzberg Daniel Grippi. Announcement: Diaspora* Will Now Be A Community Project. https://web.archive.org/web/20121109114722/http://blog.diasporafoundation.org/2012/08/27/announcement-diaspora-will-now-be-a-community-project.html, August 2012. Online, accessed 21.03.2019.

[32] Jörg Daubert, Mathias Fischer, Stefan Schiffner, and Max Mühlhäuser. Distributed and Anonymous Publish-Subscribe. In Javier Lopez, Xinyi Huang, and Ravi Sandhu, editors, *Network and System Security*, pages 685–691, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[33] Jörg Daubert, Leon Bock, Panayotis Kikirasy, Max Mühlhauser, and Mathias Fischer. Twitterize: Anonymous micro-blogging. In *2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA)*, pages 817–823. IEEE, 2014.

[34] Thomas Elliott. The State of the Octoverse: top programming languages of 2018. `https://github.blog/2018-11-15-state-of-the-octoverse-top-programming-languages/`, November 2018. Online, accessed 22.03.2019.

[35] Facebook. Facebook - Terms of Service. `https://www.facebook.com/legal/terms/plain_text_terms`. Online, accessed 22.03.2019.

[36] Facebook. Requests for user data. `https://transparency.facebook.com/government-data-requests`. Online, accessed 22.03.2019.

[37] Facebook. Facebook Q4 2018 Results. `https://s21.q4cdn.com/399680738/files/doc_financials/2018/Q4/Q4-2018-Earnings-Presentation.pdf`, January 2019. Online, accessed 21.03.2019.

[38] Seth Fiegerman. Congress grilled Facebook's Mark Zuckerberg for nearly 10 hours. What's next? `https://money.cnn.com/2018/04/12/technology/facebook-hearing-what-next/index.html`, April 2018. Online, accessed 21.03.2019.

[39] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz. Lifesocial.kom: A secure and p2p-based solution for online social networks. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 554–558, Jan 2011.

[40] Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. Practical security in p2p-based social networks. In *2009 IEEE 34th Conference on Local Computer Networks*, pages 269–272. IEEE, 2009.

[41] Kalman Graffi, Sergey Podrajanski, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz. A distributed platform for multimedia communities. In *2008 Tenth IEEE International Symposium on Multimedia*, pages 208–213. IEEE, 2008.

[42] Kalman Graffi, Dominik Stingl, Julius Rückert, Aleksandra Kovacevic, and Ralf Steinmetz. Monitoring and management of structured peer-to-peer systems. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 311–320. IEEE, 2009.

[43] Paul Grewal. Suspending Cambridge Analytica and SCL Group From Facebook. `https://newsroom.fb.com/news/2018/03/suspending-cambridge-analytica/`, March 2018. Online, accessed 21.03.2019.

[44] Kevin Jahns, István Koren, Michael Noronha, Lars Karbo, and Lukas Drgon. Yjs repository on GitHub - README. `https://github.com/y-js/yjs`, February 2019. Online, accessed 21.03.2019.

[45] Evan Prodromou James M Snell. Activity Streams 2.0 - W3C Recommendation. https://www.w3.org/TR/activitystreams-core, May 2017. Online, accessed 21.03.2019.

[46] David Johnston, Sam Onat Yilmaz, Jeremy Kandah, Nikos Bentenitis, Farzad Hashemi, Ron Gross, Shawn Wilkinson, and Steven Mason. The General Theory of Decentralized Applications, Dapps. https://github.com/DavidJohnstonCEO/DecentralizedApplications, February 2015. Online, accessed 21.03.2019.

[47] James F Kurose. *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[48] Christian Lüthold. Hive2Hive repository on GitHub - README. https://github.com/Hive2Hive/Hive2Hive, March 2015. Online, accessed 21.03.2019.

[49] Wanying Luo, Qi Xie, and Urs Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 26–33. IEEE, 2009.

[50] Wanying Luo, Qi Xie, and Urs Hengartner. FaceCloak Download. https://crysp.uwaterloo.ca/software/facecloak/download.html, August 2011. Online, accessed 22.03.2019.

[51] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[52] Dmitri Moltchanov. Selected DHT algorithms: Chord and Pastry. https://www.cs.tut.fi/kurssit/ELT-53206/lecture03.pdf, October 2014. Online, accessed 21.03.2019.

[53] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[54] Siraj Raval. *Decentralized applications: harnessing Bitcoin's blockchain technology*. O'Reilly Media, Inc., 2016.

[55] Chuck Rossi. Rapid release at massive scale - Facebook Code. https://code.fb.com/web/rapid-release-at-massive-scale/, August 2017. Online, accessed 21.03.2019.

[56] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[57] RTG Privacy and Trust for Mobile Users. Research Area B: Privacy and Trust in Social Networks. https://www.informatik.tu-darmstadt.de/privacy-trust/research_5/research_area_b__privacy_and_trust_in_social_networksresearch_area_b__privacy_and_trust_in_social_networks/index.en.jsp. Online, accessed 22.03.2019.

[58] RTG Privacy and Trust for Mobile Users. Subarea B.2: Privacy Protection via Hybrid Social Networks. https://www.informatik.tu-darmstadt.de/privacy-trust/research_5/research_area_b__privacy_and_trust_in_social_networksresearch_area_b__privacy_and_trust_in_social_networks/b__2_privacy_protection_via_hybrid_social_networks/index.en.jsp. Online, accessed 22.03.2019.

[59] Nico Rutishauser. Hive2Hive - Guide for System Admins. https://github.com/Hive2Hive/Android/wiki/Guide-for-System-Admins, March 2015. Online, accessed 21.03.2019.

[60] Maxwell Salzberg. Kickstarter Pitch. https://web.archive.org/web/20110814222702/http://blog.joindiaspora.com/2010/04/27/kickstarter-pitch.html, April 2010. Online, accessed 21.03.2019.

[61] Maxwell Salzberg. We Made It! https://web.archive.org/web/20110713115706/http://blog.joindiaspora.com/2010/05/08/we-did-it.html, May 2010. Online, accessed 21.03.2019.

[62] Mike Schroepfer. An Update on Our Plans to Restrict Data Access on Facebook. https://newsroom.fb.com/news/2018/04/restricting-data-access/, April 2018. Online, accessed 21.03.2019.

[63] Ben Smith. Project Strobe: Protecting your data, improving our third-party APIs, and sunsetting consumer Google+. https://www.blog.google/technology/safety-security/project-strobe/, October 2018. Online, accessed 21.03.2019.

[64] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[65] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[66] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[67] Adam Tornes. Introducing Twitter premium APIs. https://blog.twitter.com/developer/en_us/topics/tools/2017/introducing-twitter-premium-apis.html, November 2017. Online, accessed 22.03.2019.

[68] Twitter. Developer terms - More about restricted uses of the Twitter APIs. https://developer.twitter.com/en/developer-terms/more-on-restricted-use-cases. Online, accessed 22.03.2019.

[69] Twitter. Get started with the Twitter developer platform. https://developer.twitter.com/en/docs/basics/getting-started. Online, accessed 22.03.2019.

[70] Twitter. Twitter Terms of Service. https://twitter.com/en/tos. Online, accessed 22.03.2019.

[71] Twitter. Q4 2018 Earnings Report. https://s22.q4cdn.com/826641620/files/doc_financials/2018/q4/Q4-2018-Slide-Presentation.pdf, February 2019. Online, accessed 21.03.2019.

[72] Guy Verhofstadt. Mark Zuckerberg failed to address European concerns about Facebook. https://edition.cnn.com/2018/05/23/opinions/mark-zuckerberg-european-parliament-facebook-verhofstadt-intl/index.html, May 2018. Online, accessed 21.03.2019.

[73] Christopher Lemmer Webber, Jessica Tallon, Erin Shepherd, Amy Guy, and Evan Prodromou. ActivityPub - W3C Recommendation. `https://www.w3.org/TR/activitypub/`, January 2018. Online, accessed 21.03.2019.

[74] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53, 2004.

Image Sources

Figure 3.6:

- Servers by ProSymbols from the Noun Project, `https://thenounproject.com/search/?q=server&i=2023410`, accessed 21.03.2019
- Database by Kimmi Studio from the Noun Project, `https://thenounproject.com/search/?q=database&i=2321456`, accessed 21.03.2019
- Blockchain by Matthias Hartmann from the Noun Project, `https://thenounproject.com/term/blockchain/2306694/`, accessed 21.03.2019
- Browser by QualityIcons from the Noun Project, `https://thenounproject.com/search/?q=browser&i=2328976`, accessed 21.03.2019
- Peepeth logo, `https://s3-us-west-1.amazonaws.com/peepeth/logosquare.png`, accessed 21.03.2019
- IPFS logo, `https://github.com/ipfs/logo`), accessed 21.03.2019
- Ethereum Logo, `https://de.wikipedia.org/wiki/Ethereum#/media/File:Ethereum_logo.svg`, accessed 21.03.2019

Figure 4.1:

- Servers by ProSymbols from the Noun Project, `https://thenounproject.com/search/?q=server&i=2023410`, accessed 21.03.2019
- User by Wilson Joseph from the Noun Project, `https://thenounproject.com/search/?q=user&i=911745`, accessed 21.03.2019

Figure 5.6, Figure 5.7, Figure 5.10, Figure 5.12:

- Stick figure by Adam Beasley from the Noun Project `https://thenounproject.com/search/?q=stick%20figure&i=203593`, accessed 21.03.2019

Acronyms

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
APK	Android Package
AWS	Amazon Web Services
CBC	Cipher Block Chaining Mode
CPU	Central Processing Unit
CSS	Cascading Style Sheets
dApp	Decentralized Application
DB	Database
DHT	Distributed Hash Table
DOM	Document Object Model
dWeb	Decentralized Web
ETH	Ether
EVM	Ethereum Virtual Machine
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
NAT	Traversal Using Relays around NAT
NFC	Near Field Communication
OSN	Online Social Network
P2P	Peer-to-Peer

PWA	Progressive Web App
SDK	Software Development Kit
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TURN	Traversal Using Relays around NAT
TTL	Time to Live
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol