University of Amsterdam
Master System and Network Engineering
Research Project 2

Functional breakdown of decentralised social networks

Student:
Wouter Miltenburg
Wouter.Miltenburg@os3.nl

Supervisor:
Michiel Leenaars
Michiel.ML@NLnet.nl

July 6, 2015

Abstract

Current centralised social networks are used by a huge number of users for a variety of reasons. With Facebook, Google+, LinkedIn, or Twitter, users are not in control of their own data and access control is centralised and proprietary. Decentralised social networks could be a solution to these problems and gives the control of the data back to users. This research is focused on the question which of the decentralised social networks is currently most suited to be provided as a service by hosting providers. This paper will therefore provide information about the current implementations, the protocols used by the various implementations, and will give a functional breakdown of the various decentralised social networks. Various implementations have been analysed, namely diaspora*, Friendica, GNU social, pump.io, and RedMatrix. The paper also describes the set of standards and protocols used by the various implementations. As most implementations use their own protocol, or implement the standards slightly differently, there is no interoperability without the use of extra plugins or enabling certain features. While there are a lot of standards for facilitating the message exchange, there are however standards missing that would make interoperability possible, which is described in this paper as well. RedMatrix is currently most suited to be provided as an alternative to the current centralised social networks and that can be provided as a service by hosting providers. It has an efficient message distribution model, enhanced privacy features, and provides an unique feature named nomadic identities. A list of recommendations and future work is included in this report as well.

Contents

Chapter 1

Introduction

Social networks like Facebook, Google+, LinkedIn, or Twitter are used by a huge number of people for a variety of reasons. People want to stay in contact with each other, stay updated about what is happening around them, or are reaching out for help. Inside a commercial (or in fact any third party) social network, users are not in control of their own data, and access control is centralised and proprietary. Facebook's network and its back end might well be a whole distributed infrastructure but it is considered as a black box from a user's or developer's perspective. If there is an infrastructure or connectivity problem within a single company, the whole service can come to a grinding halt. Users do not have access to their data at that moment and when a service is discontinued or the user is no longer interesting, acceptable, or eligible as a customer for some reason all of the user's data is lost along with their connections to other people and organisations - in effect their entire social graph.

However, there is a solution for this problem by transforming the social network landscape from a centralised model to a decentralised one. This research started with the question on which implementation of a distributed social network can be included in the ARPA2 project [23]. The ARPA2 project wants to bring modern and secure Internet technology to end-users by providing a hosting platform that can be installed by a variety of hosting companies. If this hosting platform is adopted by hosting providers it can be offered as a solution to a great number of users.

This research will focus on the features that are currently made available by various independent projects. Which of them are mature enough, which of them are scalable, what kind of open protocols are used such that interoperability can exist between installations or other types of services, and what kind of non existing standards are currently limiting the development of these decentralised social networks.

1.1 Related Work

Daniel R. Sandler and Dan S. Wallach propose the FETHR system [48] that connects micropublishers in a single global network. It describes what kind of users really make an impact on the workload of the overall system and also describes the lightweight HTTP-based protocol that is used to gossip new messages among subscribers.

Quite some research has already been done in P2P systems for social networks. Cuckoo by Xu et al. [57, 58] proposes an overlay network for providing a scalable and reliable microblogging service. This is done whilst offloading the traffic in its own P2P system, such that the Twitter servers conserve their bandwidth and processing power. There is also Litter by Juste et al. [35] that makes use of already existing peer-to-peer technologies and the paper describes Litter's implementation of sending and receiving messages. T. Perfitt and B. Englert propose Megaphone [43], a secure microblogging system that makes use of user certificates and where the network is arranged based on these certificates.

Thiel et al. [54] already looked at the decentralised networks that exist and which are suitable based on a requirements driven approach. However, this research only looked at a few implementations and did not cover everything that will be researched in this project.

The aforementioned research was of great help when we analysed the existing decentralised social networks. It gave us an idea how messages can be distributed inside a network, although most of the mentioned papers are about peer to peer systems, some of the solutions could be implemented in a non peer to peer system as well. The paper by Thiel et al. gave us an idea how we could analyse the existing decentralised social networks. Other papers that are of relevance to this research, and which have been used throughout this project, will be referenced accordingly in this paper.

1.2 Research Questions

The research is formed around the main question shown below.

What current implementation of a social decentralised network could be considered as an alternative to the current centralised social networks and could be offered as a service by hosting providers?

As a result of the main question, shown above, there are the following sub questions.

- Which functionalities exist in the typical social networks that we know nowadays?
- Which alternative open source projects are available that are mature enough and which provide these functionalities in a decentralised model?
- How do these different alternative open source projects differ from each other in a practical sense (e.g. security, standardisation, ID re-use, and scalability)?
- Which implementation is most suited to create a decentralised social network that can be provided as a service by hosting providers?

1.3 Approach and Methods

This research project mostly applied theoretical research. We have first analysed some existing large centralised social networks (i.e. Facebook, Google+, LinkedIn and Twitter) and made a list of features that are supported by these social networks. From this list we have created a list of basic features that must be supported by the implementations. After this had been done we made an inventory of the implementations that existed. Only the implementations that provided the basic feature set were considered an option along with some other requirements. After this had been done the features were analysed that are implemented by the different decentralised social network solutions. This also involved analysing the protocols that are used in the systems to see if interoperability can be obtained between the different solutions. Another aspect of the systems that was researched, is if the solutions can be scaled. To be able to answer all of this, a test setup was made that ran all of the analysed implementations. During the analysis of the implementations, we have also looked at the network traffic to get a better understanding of intercommunication between nodes. After we analysed the implementations, it was possible to see the limitations of the different implementations and we were able to conclude that indeed distributed social networks may realistically be provisioned through hosting providers. We identified a number of possible candidates and one implementation, at that moment, seemed to be most suited to be provided as an alternative to the current centralised social networks. Recommendations and suggestions were also included in this report if some limitations were observed or to improve the overall functionality of an implementation.

1.3.1 Use cases

We started by creating a number of use cases that should be supported by the distributed social networks. These use cases have been made up during the research and are based on other use cases that have been used in previous research, which will be outlined in Section 2.1. The use cases are general enough such that they are applicable to real life scenarios, general enough to cover a range of different use cases that are similar to the ones mentioned in this report, and are used during the analyses of the implementations. The use cases are shown below.

- Friends that are in a bar and who want to send photos privately among each other which should only be shared in that specific group.
- A civil rights group that wants to keep a big audience up-to-date about a campaign launch.
- Citizen journalists that want to have a convenient way to share a short status update about emerging news, but also share a quick update with a fellow journalist that

should stay private. It is also important that this data can be discovered by, for example, a news website and that people can respond to this news.

The aforementioned use cases are quite different from each other. The first use case is targeted at a small group of people that want to share data privately amongst the participants of the group. Since the shared data might contain photos that should really stay private, a photo might contain an embarrassing picture of a member of the group, this data should not leak to other users that are not part of the group. However, people may know that the friends are connected with each other (e.g. are friends with each other and this may be shown on their public profile).

In the second case (a civil rights group launching a campaign) the message should be able to reach all of its followers. It could be the case that a campaign should stay private to the subscribers of this group, at least initially, for strategic and/or security reasons. Participants may also have the need to privately respond within such a campaign.

In the third case a citizen journalist wants to quickly send a short status update about emerging news, for example, a big fire in a city. However, it is also important that the journalist can keep in touch with other journalists. In such a use case, the journalists might have the need that people can not see that these journalists are related to each other (e.g. have a friendship) or that the identities used for sending messages can not be linked back to the journalists. It is also necessary that these news updates can be discovered by other (news) web sites, since this citizen journalist might not have as many followers as regular journalists. This makes it harder for other websites to discover these updates. When the social updates are made available to the subscribers it is important that people can respond to these updates.

Chapter 2

Theory

This chapter will look at the theory behind the different decentralised social networks and the current centralised social networks. It is therefore important to first get an understanding why people use social networks like Facebook or Twitter, before discussing the decentralised social network models.

2.1 Social networks and its users

In this section we will outline why people mostly use social networks. Although this project is not focused on analysing the reasons why people use a social network, it is however still interesting and relevant for this project to look at the work that has been done already in this field. With the knowledge why people use a certain social network we can more specifically look for decentralised social networks that might succeed in today's Internet.

Ashwini Nadkarni and Stefan G. Hofmann review in their paper the existing literature on the psychological factors contributing to Facebook usage [40]. In their paper they propose a dual-factor model of Facebook use where, according to that model, Facebook usage is primarily motivated by two basic social needs. These are *the need to belong* and *the need for self-presentation*. The paper gives more information about the two models and how this relates to the use of Facebook and that even variations of Facebook use can be seen in different cultures.

Steinfield et al. describe in their paper the relation between the use of Facebook and the formation and maintenance of social capital [26]. In their paper it is noted that Facebook is mostly used for bridging (e.g. keeping in touch with people that might be far away). They also suggest that social networks like Facebook could support *a variety of populations, including professional researchers, neighborhood and community members, employees of companies, or others who benefit from maintained ties* [26].

Alexander Altmann compares and reviews different social networks in terms of their architecture, security and privacy [20]. He make use of certain use cases that should be supported by the social networks and discusses if these social networks supports these and outlines what should be improved in these social networks. Some interesting insights are

shown in this paper and it shows that people want more privacy. He also outlines that there is a use in having multiple profiles such that certain use cases can be supported.

The research that has been mentioned in the previous paragraphs all boils down to a few features that should be supported by social networks. People mostly want to stay in touch with each other, which requires that there should be a contact list to reach other persons. There should be a form of identity such that another person can be reached and information can be exchanged. Most of the information exchange is in text and it must be possible for a person to exchange this information in text (e.g. publish a message on a public feed). Instead of making a post public, a person might want to target a more specific audience. People want to respond to these updates and it should therefore be possible to comment on such an update. An update is not only limited to text and hypertext: anything that is digital (or has a digital representation) can be shared inside an online social network. Some people post media such as photos, graphics or movies to establish an idealised image. Here again, people should be able to comment on these pictures as well. Some social networks provide the functionality to like[1] or favorite a certain message or photo. Privacy is important as well, which can be differently classified and categorised. First there is the notion of privacy that is related to confidentiality. People should not be able to read each other's messages without the users' consent, there should be a possibility of exchanging messages privately. The other notion is more related to trust, trusting the social network or the provider such that they do not use one's data for advertisements or glean in the data for data analyses. The previous mentioned features are basic features that should be supported by a social network, since these features are the reason of existence of social networks. Although different social networks implement these features differently, and might even provide more features than the ones described previously, they all boil down to the aforementioned items. In Section 2.3 it is outlined what features are currently supported by the existing large centralised social networks and which features need to be supported by the decentralised implementations.

2.2 Protocols

In this section we will look into the variety of protocols that exist to facilitate the exchange of messages in decentralised social networks. We outline the protocols that have been specifically developed to support decentralised social networks, which is a network where a

[1] A like or 'plus one' is a terse form of social comment that due to the lack of expressiveness of the vocabulary necessarily can have a range of linguistic intentions. It is typically (but not in all cases) a form of phatic communication: it is not intended to impart information or evoke a new response, but is used to express or create the atmosphere of shared feelings and goodwill. In its essence it is meant to indicate to the group that a certain member agrees with some prior statement and/or endorses some contextual combination of objects, services or persons. It is intentionally passive.

variety of interactions take place between users on different nodes (i.e. commenting, liking, sending friendship requests, and discovering new users).

2.2.1 DFRN

The Distributed Friends & Relations Networks protocol [39] is a protocol used for sharing information about users between different servers with the users' consent. The protocol specification describes how users create a social relationship and describes that there is the notion of a single-way relationship and a duplex relationship. When a requester sends an invitation by using the DFRN-request page[2], which is the message to introduce the user and used for establishing a one-way relationship, the receiver would likely respond by using the DFRN-confirm page. When this is done it gives the receiver of the invitation the ability to talk with the user that initiated the invitation. The user that initiated the invitation therefore only sees the information on his feed from the identities that he established a relationship with. In the protocol specification itself it is noted that this seems unnatural at first but it gives the user complete control about their own privacy and the user is in complete control in the information that he wants to receive.

The other page specified in the DFRN protocol is the DFRN-notify page. This page is used to interact with another server that is interested in a user from the server that sent the message. It might, for example, be information about updated profile information of a specific user or updates of existing content. The messages itself are encoded in Atom [41] with the Atom "deleted-entry" [52] and threading [51] extensions. RINO encryption, which is explained in the same protocol specification, is used to protect the data that is exchanged between the servers. The RINO encryption layer is only used when DFRN-notify is used, as all other messages are addressed to the public or are not classified as a secret message.

The last page specified is DFRN-poll, which is used to poll information from a user. The distinction between polling and notifying someone is explained in the protocol specification as follows.

> There are two types of communication in DFRN broadcast or public communication, and directed or private communications - which are only visible within targeted individuals or groups. There are likewise two methods of information discovery. You will usually ***notify*** recipients of targeted and timely information, whereas public broadcasts and side conversations that they aren't involved with directly are picked up by others ***polling*** your site from time to time to check for updates. (From: [39])

[2]The page is merely the URL where the POST or GET requests are sent to.

As shown above, a clear distinction has been made in messages that are sent directly (e.g. a message sent to a group of people) or with a public broadcast (e.g. an announcement by a company). The most important aspect in the protocol is authentication. An RSA key pair is generated per relationship with a minimum key length of 2048 bits, which are generated when DFRN-confirm is used, and these keys are used during authentication and authorisation of a certain identity. The specification also outlines a reputation system, where one's friends can be asked if they know a certain identity before one makes the decision of adding this identity to his circle of friends. The friends can reply to the reputation request and specify if this identity can be trusted. For more information about the protocol itself and the message flows, it is advised to read the protocol specification [39].

2.2.2 DSNP

The Distributed Social Networking Protocol (DSNP) [55] is based on an RSA based identity. There are several keys that are used for different purposes. This can range from the key that is the most critical one, which needs to be properly secured, and which is used for identity movement or deletion. Another key is used for signing login tokens that are used to prove recent login activity. The other two keys, which are less critical, are the signing key used when the user is logged in and another key for when the user is not logged in. The reason for having a fourth key, which is used when a user is not logged in, is based on the nature of social networks. Social networks still carry on when a user is not logged in, people still send friend requests to an identity. The protocol also describes a passwordless login method, which seems to be used for the same purpose as magic auth in Zot2, which is explained in Section 2.2.10. If a user wants to visit a friend's website, authentication must be performed if that identity wants to access data that has been restricted to a certain set of users. Due to the passwordless login method, the user does not notice the authentication that is performed behind the scenes.

The protocol also introduces the option of deniability.

> Broadcast and direct friend-to-friend messages must be signed to prove they come from a valid contact. Signing is both good and bad. It is good because it eliminates the possibility of attackers injecting messages into our news feed. It is bad because we give people proof that we have said something. While not expected and always unfortunate, we have to consider the possibility that something we say can be used against us. We must introduce some deniability.
>
> (From: [55].)

The protocol describes a procedure where the signing keys can be revealed such that a user can deny that certain messages have been created and subsequently signed by him.

2.2.3 Libertree

The Libertree specification [53] is another protocol[3] that allows messages to be sent between servers that are members of the Libertree network. Every connection that is made between a pair of Libertree servers must be authenticated first. However, before authentication takes place it is required to have performed an introduction. If authentication has been successfully performed, by the use of public key cryptography, the servers can exchange messages. This can vary from posting messages to commenting, liking a message, and removing posts. The exact message flows between the servers are outlined in the Libertree specification [53].

2.2.4 OpenBook

The OpenBook protocol [27] has not been updated for a while, but it provides basic functionalities that can be used by a decentralised social network. It is a protocol over HTTPS based on JSON and Markdown. The protocol can be used to share content and send feedback between hosts. It is not really clear how authentication is performed and what the user's necessary involvement is to allow a host or identity to access certain content. In their protocol specification they state that host authentication is done by verifying the SSL or TLS chain, however it does not really specify use cases where different users use the same host. As the host is verified and not an identity (e.g. a profile or user) it is not really clear how such use cases should be implemented in an implementation of a social network. The POST object is a message that is sent to another server. It contains, amongst a variety of other information, links, likes and it can also be used to subscribe to content.

2.2.5 OStatus

OStatus is a specification [47] for distributed status updates or microblogging and lets people on different social networks follow each other. It uses a group of protocols namely PubSuHubbub (PuSH), Activity Streams, Salmon as explained in Section 2.2.6, Portable Contacts and Webfinger as explained in Section 2.2.8. PubSubHubbub [29] is used in OStatus as the solution for distributing data to several subscribed nodes. A site can subscribe to a particular feed on a hub server that is associated with a particular feed of a user. If there is new information available in the feed, the publisher notifies the hub, and the hub will on its turn send this new information to all of the subscribers of the hub.

[3]There is also Libertree itself that allows users to create their own social networks. Libertree itself is still undergoing rapid development and is in an alpha stage. Therefore, Libertree itself was not considered in scope for this project at this time.

Salmon is used in the OStatus protocol to send user-to-user notifications from a server to another server. This can, for example, be a comment on a certain post available in a feed that a publisher made available to its subscribers. Salmon is compatible with the Activity Streams protocol as is shown below.

> Salmon is fully compatible with Activity Streams [AAE] Salmon MAY be activities, by having at least one activity:verb child element and one or more activity:object child elements. Salmon endpoints SHOULD accept appropriate activity verbs. Salmon endpoints MAY reject unsupported activities. Note that a Salmon endpoint which is not aware of activity streams may simply accept and store (via the provenance element) the activity in question, but will treat it as a basic Atom entry.
>
> If an activity verb is included in a salmon reply, the 'Post' Verb is the most appropriate generic verb to use. Other verbs are possible and salmon generators SHOULD use the most specific verb they can identify for their use case.
>
> Nearly any activity verb could be appropriate for a salmon mention Section 3.3. The 'Post', 'Share', 'Save', and 'Start Following' verbs [AABS] are particularly pertinent and salmon generators SHOULD use the most specific verb they can identify. (From: [33].)

The Activity Streams protocol [24] is used for describing social interactions more in detail. Instead of just posting text or a photo, the Activity Streams allow these events to be marked with a verb, for example, post, follow, favorite, and update amongst a variety of other verbs. These verbs give more context to the messages and it provides means to add more metadata to a message. Portable Contacts [50] is a specification for accessing address books and friend lists in a secure way and is used in OStatus to provide profile information.

2.2.6 Salmon

The Salmon protocol [33] is a protocol for sending unsolicited notifications. This includes comments on a feed, but can also include likes or dislikes of feed items. The protocol is also used for notifying a Salmon endpoint that a user or content has been mentioned externally. Therefore, this protocol can be used when someone on another server comments on a post that has reached its user through the use of an aggregator or by other means. The following statement is made in their protocol specification and explains that the *protocol defines the 'rules of the road' for these mechanisms, tying together and relying on lower level protocols and specifications for implementation* [33]. It can be used by different implementations to notify the upstream about events.

2.2.7 Tent

Tent [22] can be used for a variety of purposes from microblogging to personal data logging. A Tent server stores one's posts and sends copies of these posts to subscribers. A user can set up its own Tent server or use Tent servers from a Service Provider. Since Tent itself is content agnostic it can handle a variety of data types. The protocol allows for decentralised communications and can be used to exchange information between Tent servers. The user's data is stored on the Tent server and lets a user stay in control of their data. It states on their website that data from the Tent servers can seamlessly be migrated to other service providers, or one's own host, and updates address books to support seamless migration [21].

2.2.8 WebFinger

WebFinger [34] is not a protocol like the aforementioned protocols, instead it is used for a more specific purpose, namely discovery. To be more specific, it can be used to discover information about identities and more specifically people. The path component of the WebFinder URI is the well-known path */.well-known/webfinger*. With a GET request to this well known path, information can be requested for a specific WebFinger resource. This can be for example an account with the name *social-identity@example.com*. During the request, the *rel* parameter can be used as a sort of filter to only receive data for that specific link relation type.

2.2.9 Webmention

Webmention [49] is a protocol to notify an URL when one links that URL on its own site. For the receiver, the one that receives a Webmention, it is a notification mechanism and notifies the receiver that a site has linked to one of its URLs. It is a relatively simple and small protocol and they state on their website that it is *a modern alternative to Pingback and other forms of Linkback* [49]. On the website of IndieWebCamp [19] it is noted that besides regular mentions, mentions that notify the receiver that content has been linked, it also supports mentioning likes, reposts, and replies.

2.2.10 Zot2

We will discuss Zot2, as the first version was deprecated soon after it was released [30]. The Zot2 protocol [31], hereafter referred to as Zot or Zot protocol, is as outlined on their wiki page a *web framework for implementing secure decentralised communications and services* [31].

Although Zot2 and DFRN originate from the same developer, they differ quite a lot from each other. First of all it supports *nomadic identities*, which allow users to move between hosts without breaking the already established relationships. Because of this ability DNS is not used to identify a user. However, it is used to have a user friendly mechanism to let people connect with each other. It is also used for the discovery procedure and subsequent communications. However, there is an abstraction layer that allows to move an identity across nodes and gracefully recover from such an occasion such that relationships stay intact. Therefore, there is a direct requirement that an identity can move between locations and that such an identity can communicate with other identities from other locations or multiple locations at the same time. The Globally Unique Identifier (GUID) is used to identify an identity and stays the same across all locations.

The GUID can therefore live across locations and the locations can be updated with identity messages. With the use of key pairs and signatures the receiver can validate if the hub, which sent the identity message, is in the possession of the key pair. If it is, the receiver of the identity message will update the location of his friend. Currently, an identity can only have one primary hub where messages from other friends, which are addressed to that identity, are sent to.

With Zot, messages are more efficiently routed to different hubs than with DFRN. There is the notion of batching, which allows one hub to send multiple messages in a single transaction to the other hub. It also allows to send a message in the same transaction to multiple recipients that are located on the same hub. Depending on the nature of the message, the message might be encrypted when it is in transition from one hub to the other, even when the hub does not support SSL or TLS.

The discovery procedure used in Zot is quite different than WebFinger, which is outlined in Section 2.2.8. There is a well known location for the discovery procedure on a host, namely */.well-known/zot-info*. One can issue a POST request to discover a user and the hub will respond with information about that specific user. The information that is returned includes the permissions applicable to the target user, the current hub locations of the user, and profile information.

One novel concept in Zot is *magic auth*. One can see it as a profile that roams across hubs when a certain identity visits another identity on another hub. For example, if one wants to visit a friend that is located on another hub, his identity will automatically be authenticated on that hub and the user can see information that his friend made available for him on his channel. The messages are encrypted with an AES-CBC 256-bit symmetric cipher and which is on its turn encrypted by public key cryptography that is based on 4096-bit RSA keys associated with that channel [7].

For more information about the Zot protocol and the message flows it is advised to read the protocol specification [31].

2.3 Features

The previous sections have shown what the aim of social networks is and what the protocols allow implementations to do. The main goal of a social network seems to be to letting users stay in touch by sharing status updates with each other, with the possibility to provide feedback. These updates can be accompanied by photos and other interactive visualisations. To make even more social interaction possible some social networks allow the users to 'like' a status update. It is also possible with most social networks to comment on status updates to have more interactivity with one's friends and create a lively discussion. Some of the social updates might not be intended to be read by everyone and should be restricted to a limited set of people. Although not all current social networks provide advanced privacy features, it becomes more important when a platform is moved to a distributed platform. Status updates in a distributed social network might be completely public and can be read by everyone, whereas with a standard centralised social network it might only be possible to access content if one's registered, which allows the social network providers and the user to be more in control. Therefore, it is important that information is easily accessible by all kinds of users, but also to have the option to make a social update available to a certain audience instead of making an update visible for everyone. As it is hard to measure convenience, and outside the scope of this research, this has not been analysed.

We also looked at the existing large centralised social networks (i.e. Facebook, LinkedIn, Google+, and Twitter). They all provide somewhat similar functionalities and some of them provide additional features. These additional features, e.g. birthday calendar and games, are not the reason why these social networks exist. If one looks more closely at the functionalities they provide and how these are provided to the user, they all boil down to some basic feature set that is presented in another visualisation to the user or the user uses the functionalities with a different goal. A good example might be the difference between a more professional social network like LinkedIn and a well known social network like Facebook. The users can still post status updates, accompanied with a photo, however most users use LinkedIn as a professional social medium. The general idea behind the concept is that a co-worker will more easily see one's status updates and that photos taken in a bar are generally not shared on this social network, but are more likely to be shared on Facebook instead. However, they both support the functionality of publishing status updates. The profile of a user differs in these social networks as well. For example, on Facebook people might publish what kind of music they like and what their favourite movie is, whereas LinkedIn users show their certificates and their work experience. As can be seen, the general idea is the same. Publishing information and making it possible to have interaction with different users.

All of the current large centralised social networks (i.e. Facebook, Google+, LinkedIn and

Twitter) support the features that are shown below. These features are also mentioned in the research outlined in Section 2.1. However, the features shown below are only the ones that can be found on all of the social networks and have a direct relationship with a social network (e.g. we have omitted the additional features, e.g. birthday and the group chat features, as they can be provided by other solutions that are not directly related to social networks).

- Broadcast a message (post an 'update')
- Receive notifications of incoming messages
- (Re-)distribute a message
- Reply or publish a comment linked to a message
- Publish and view user profiles
- Like a post or comment
- Subscribe to messages from a user (might be known as friendships, connections or followers)
- Partition potential audience into subsets by creating groups, channels or access control lists

Social updates can contain just text or a richer representation of information like a photo. The update may be targeted to the public or to a specific set of people. The features shown above are the core features of social networks and as such they should be supported by decentralised social networks. They will hereafter be referred to as the basic feature set[4]. We added the functionality of liking a status update to the list, which might be known as *+1* in other social networks. Note that the 'favorite' mechanism such as implemented by Twitter [56] is slightly different from the generic 'like' mechanism, in that it also acts as a bookmarking mechanism and that this tag seems more persistent and is meant to convey something about the user as much as the endorsed object. While a user is no longer expected to revisit a previous like, viewing one's favorites is more likely to occur on a regular basis. As such it can be seen as an indication that a slightly richer semantic than just 'like' can lead to new use cases, and as such is something that may allow decentralised social networks to provide additional value. From a technical point of view, there are no different requirements with regards to protocols - beyond the ability to display more than a single counter.

[4]Some of the centralised social networks might apply certain restrictions to these features, such that the functions provided by the social networks are different from the ones that are provided by another social network. For example, the restriction in the number of characters of a Twitter message.

Chapter 3

Analysis

In this chapter the various projects are discussed that provide users with a decentralised social network. It will also outline which functions are provided by the various projects and how they resolve some of the problems that exist in implementing a social network (e.g. lookup, security and notion of identity).

3.1 Introduction of the implementations

This section gives an introduction to the analysed implementations of decentralised social networks. Some implementations were not considered in scope for this project at this moment. The projects that were not considered in scope at this moment are listed in Appendix A along with the reasons why this decision has been made. The projects that were considered in scope for this project are shown below.

- diaspora*
- Friendica
- GNU social
- IndieWebCamp[1]
- pump.io
- Movim
- RedMatrix
- rstat.us

The projects mentioned above were considered in scope at the beginning of the project. However, after analysing the projects it seemed that Movim and rstat.us were not mature enough to be considered an option and were missing a few basic features. Some of the following features were missing from these two implementations: sharing photos, sharing locations, advanced privacy settings, and notification settings. All the other projects provide the basic feature set[2] mentioned in Section 2.3. One important aspect that also needs to be supported is some form of privacy. In the following section we will first discuss the various implementations. We will then outline what other features the implementations

[1] IndieWebCamp will be discussed in a separate section as IndieWebCamp itself is not an implementation. However, the projects of IndieWebCamp and the protocols used in these projects seem to have some similarities with decentralised social networks. This will be discussed in Section 3.3.

[2] diaspora* did have the functionality to like or favourite comments in the past. However, there is still an ongoing discussion if this needs to be implemented again [17]. We have made the decision to consider diaspora* still in scope, even when it does not support this feature.

provide. As the basic feature set is already supported by these projects, it is therefore more interesting to outline what kind of other features they provide that can come into interest to the user base.

3.1.1 About the implementations

In this section we will give a quick introduction about the various projects and what their strengths are.

diaspora*

> diaspora* is based on three key philosophies:
>
> *Decentralization*
> Instead of everyones data being contained on huge central servers owned by a large organization, local servers (pods) can be set up anywhere in the world. You choose which pod to register with - perhaps your local pod - and seamlessly connect with the diaspora* community worldwide.
>
> *Freedom*
> You can be whoever you want to be in diaspora*. Unlike some networks, you dont have to use your real identity. You can interact with whomever you choose in whatever way you want. The only limit is your imagination. diaspora* is also Free Software, giving you liberty to use it as you wish.
>
> *Privacy*
> In diaspora* you own your data. You do not sign over any rights to a corporation or other interest who could use it. With diaspora*, your friends, your habits, and your content is your business ... not ours! In addition, you choose who sees what you share, using Aspects. (From: [15].)

Diaspora*, hereafter referred to as Diaspora, focuses on creating a decentralised social network where the user is under the control of his or her data. In Diaspora people can be grouped in so called aspects, which are also used in the data sharing model. If one wants to make an item visible to a selected number of people, the user is able to do this by making a certain post available for a selected aspect. Diaspora makes use of Salmon for data distribution between different users on different hosts and WebFinger for the lookup procedure. However, compared to the official Salmon specification the message exchange is done a bit differently in Diaspora [5]. The message encoding of the messages itself, including likes, is an own protocol based on XML.

Friendica

> The Friendica Project is a world-wide consortium of software developers creating decentralised social platforms and technology for the coming post-Facebook world. We aren't as flashy and well known as some of the other projects working on a decentralised/federated social web, but we've been quietly working behind the scenes to provide the most reliable, full-featured, and extensible alternative to the monolithic providers. (From: [16].)

Friendica has quite a different approach and feels more like a social network to most users. It is possible to share status updates, create birthday events and exchange the photos with one's friends. One of the aimers of the project is also interconnectivity with other social networks, albeit decentralised or centralised social networks. It is therefore possible with Friendica to connect with remote users of Friendica, Diaspora, pump.io, Facebook, Twitter and a variety of other systems. It makes use of DFRN as the message protocol, where the messages itself are encoded in Activity Streams and where WebFinger is used to discover information about users. Salmon is used to exchange messages regarding replies and mentions with OStatus implementations and would otherwise use DFRN between Friendica instances. Portable Contacts is used for friend lists and PubSubHubbub is used for interconnectivity with OStatus compatible implementations. However, as posts are syndicated to other social networks the same privacy guarantees might not be guaranteed anymore as the information is in control of the other social networks as well.

GNU social

> GNU social is a continuation of the StatusNet project. It is social communication software for both public and private communications. It is widely supported and has a large userbase. It is already used by the Free Software Foundation, and Richard Stallman himself. (From: [6].)

As shown in the statement of GNU social it is the continuation of the StatusNet project. It provides the user with a web interface to post activities, just like Facebook and Twitter. It is possible to create groups and to mention other people on remote hosts, just like the other implementations. GNU social is based on the OStatus protocol stack. It therefore uses all of the protocols that are described in Section 2.2.5.

pump.io

> This is pump.io. It's a stream server that does most of what people really want from a social network.
>
> [...]

> I post something and my followers see it. That's the rough idea behind the pump. (From: [13].)

This short statement already reveals what pump.io is about, it is about posting activities with ease and sharing it with the followers. Audiences can be grouped together in lists and posts can be restricted to certain people or made publicly available. It uses Activity Streams as the data format and OAuth for authentication purposes. Web Host Metadata [32] is used for the discovery procedure of remote identities.

RedMatrix

> Personal Web Publishing
>
> A decentralised web platform for sharing your online identity, digital media, and thoughts with those you wish - and only those you wish.
>
> Public when you want it, privacy when you require it. (From: [14].)

RedMatrix is quite distinct from the other implementations, as it originally is a personal web publishing platform. However, RedMatrix can also be used for a decentralised social network as it provides the same features as the current social networks and even more features like WebDAV and group chat. It is really focused on privacy and information itself is posted in a channel. A channel can be used for grouping related information together, for example, car related posts. However, the user is free to post in a channel whatever he feels like. RedMatrix makes use of the Zot2 protocol as explained in Section 2.2.10.

3.1.2 Analysed versions

In this section we will provide the version number of the implementations that we have tested. All statements made in this report are based on code analysis of the implementations with the specific version numbers that are shown in Table 3.1, communication with the developers, analysing network traffic, and when we used the implementations in our test environment. The dates of the Git commits are shown in Table 3.2.

Project	Git commit	Repository
Diaspora	6c164fe2364def5d10db44ea4d603bd747435e62	GitHub.com/diaspora
Friendica	beb4980f020e09625357114159a860f64f030004	GitHub.com/friendica
GNU social	3294d704a44203eb891d4b6485452fd16976ec2e	git.gnu.io/gnu
pump.io	1029ac71b94dec2b51f6b2aa461b5b2a514e585a	GitHub.com/e14n
RedMatrix	edd2d1e8d47be1ef4fe38edf624335472a2e73bd	GitHub.com/redmatrix

Table 3.1: Analysed versions.

Project	Date (DD/MM/YYYY)
Diaspora	06/05/2015
Friendica	20/05/2015
GNU social	30/05/2015
pump.io	22/06/2014
RedMatrix	10/06/2015

Table 3.2: Date of Git commits.

3.2 Functional breakdown

As was discussed earlier, the basic features are already provided by the implementations that we analysed. However, in a decentralised system these features need to be differently implemented than with a centralised system. Sharing status updates, for instance, is possible with all of the implementations. However, in a decentralised system it is important that the messages are distributed in an efficient manner. These considerations and the extra features that are provided by the implementations are discussed in this section. This section will therefore provide a functional breakdown of the features supported by the implementations and also outline how these are implemented or can be used by a user.

3.2.1 Advanced privacy settings

In Section 2.1 it was outlined that people want more privacy and to be more in control of their data. Therefore, it is even more important how certain privacy aspects are managed in these distributed social networks. All of the projects have some basic set of privacy options to regulate which users can see a certain status update. However, in a distributed solution this needs to be taken one step further.

Diaspora, Friendica, GNU social and RedMatrix have privacy options that allow a user to be more in control of their data. For example, if someone sends a post to a limited subset of his friends (in a certain aspect, channel or group) that post must only be sent to them. This should be taken one step further, such that persons in that group can not re-share such a post. If this is not properly protected, it will be possible to re-share a post that was previously considered private. With Diaspora, GNU social, Friendica and RedMatrix it is not possible to re-share a post that has been targeted to a certain audience or that has been marked private. Only public posts can be re-shared. However, this form of privacy has not been implemented in pump.io. Another problem exists with GNU social, where groups can be created but only public posts can be made available to users on remote hosts. A private message that is posted in a group can not be seen by users on remote hosts. With Diaspora there is the notion of aspects that can be used to make posts available for a certain public. It is however not possible, as is possible with Friendica, pump.io and RedMatrix, to only make a post available for certain users without creating an aspect.

Depending on the type of content one wants to share, and with which intention, a user might benefit from having the option to have multiple profiles. For example, one might want to have a profile that is used to stay in contact with co-workers and another profile to stay in touch with old friends. This might not at first sight sound like a privacy feature but with such a feature it is possible to segment one's profiles for different audiences. With the advanced privacy options available in Friendica and RedMatrix, it is possible to have several profiles connected to the same user account. Users can visit the default profile, but another profile can be assigned to a friend when the friend request is accepted. The next time the other person will visit the user's profile he will see the assigned profile.

Having multiple profiles can already give users the opportunity to take control back into their own hands and tune what information should be made available to certain users. However, RedMatrix goes one step further and this really goes back to the core of RedMatrix. RedMatrix is not really a decentralised social network, it is more than that. It can be used as a decentralised social network but it originally is a personal publishing system with decentralised privacy. There is the notion of channels where information is posted to. Ideally a channel relates to a certain topic but it is up to the channel owner to post whatever information he feels like. However, with the notion of channels we can segment even the information between categories and use different profiles to only make information about a user available to the people we want to.

It is also important that photos are shared with the intended recipients. The aforementioned projects all provide a basic feature set to share photos to one's friends, however RedMatrix takes privacy serious. With Friendica for example it is possible to upload photos and configure its audience, this audience are the only ones who can see these pictures. However, as RedMatrix is a personal publishing system, it can also share arbitrary data in its own cloud. Different permissions can be given to the arbitrary data and it supports

the WebDAV protocol to upload data to the cloud. This permission system does not only apply to content but also to channels itself. There are a lot of permission when one opens the advanced privacy settings of his channel. It is therefore possible to specify that users can not see the connections of a channel. There are currently 18 options to modify the privacy settings of a single channel to the user's demands. These advanced configuration options are not available in other implementations.

3.2.2 Admin interface

An admin interface is necessary to manage the users on one's hub[3] and block users who violate the Terms of Service. The admin interface is very limited of scope in GNU social, it is more an admin interface to manage the website instead of the users. However, with GNU social it is possible to go to the profile of a user and delete his profile or his comments. The default installation of pump.io does not provide an admin interface. However, with Diaspora, Friendica and RedMatrix a web interface is provided for an administrator. The web interfaces of Friendica and RedMatrix are however very limited. Users can be blocked or deleted but it is not possible to view or delete a certain post that may have been marked private. Therefore, it is not possible to look from within the admin interface at private posts and hand them over to a federal government. This is a bit of a controversial topic, because when an administrator needs to remove a certain message he also needs to be able to view the message. No true privacy can be guaranteed here as an administrator can still look at the data. However, with the current implementation node owners can always look at the data as it is not stored encrypted on the servers and if it is, the keys used to decrypt the messages are stored on the server as well. This is not possible with end-to-end encryption, which will be discussed in Section 3.2.3. As this is quite a controversial topic on how much data an administrator needs to see and still be able to maintain the social network nodes, the current admin interface seems sufficient. Diaspora provides a more advanced web interface where posts can be seen, even the ones that are shared with a specific set of people (i.e. aspect). With the admin interface it is also possible to look up the last IP address that has been used to sign into a specific account. Therefore, Diaspora provides more functionalities but an administrator can easier glance at data than with RedMatrix and Friendica.

3.2.3 Encryption

Messages in transit between two friends are encrypted with RedMatrix and Diaspora using an AES-CBC 256 bit symmetric cipher. The key used for encrypting the message is

[3]In other parts of this document we might refer to node, hub or server. This is the server where an implementation is running on.

encrypted with the public key of the recipient and added to the same message. The key is obtained using WebFinger with Diaspora and Zot2 with RedMatrix. The private key of the recipient can be used to decrypt the part of the message that contains the key that was used to encrypt the part of the message that contains the actual message. However, only with RedMatrix are private messages stored in an encrypted form on the server. It is still possible for the administrator to retrieve the content of the original message, as the keys are also stored on the server. With Diaspora the messages are not stored in an encrypted form on the server. With Friendica, the messages are only encrypted in transit when RINO is used and this is not enabled on all Friendica instances by default. Since it was not possible with GNU social to send a private message in a group with remote friends, as explained in Section 3.2.1, it was not possible to verify if encryption would be applied. However, all the messages that can be exchanged between GNU social instances are currently unencrypted. With pump.io the messages are not encrypted when they are in transit. However, these problems could be solved by using SSL/TLS.

In the third use case, as outlined in Section 1.3.1, we have described a scenario where a journalist wants to privately communicate with another journalist. Having end-to-end encryption[4] would benefit the journalist since no one, except the journalists, can see the content of the original message. In RedMatrix it is possible to use end-to-end encryption for private messages using AES256-CBC mode. The key used for encrypting the message can be entered by the user and an out-of-band mechanism is needed to exchange this key with the other party. By using end-to-end encryption it would in theory not be possible for administrators to look at the content of the encrypted messages. However, as noted by RedMatrix itself, it is still possible for hub administrators to inject code to retrieve the key [3]. The other implementations do not provide a feature to have end-to-end encryption for private messages.

3.2.4 Photo wall

Photo sharing is already possible with large social networks like Facebook and Twitter. It is also possible to share photos with one's friends using the aforementioned implementations. With Diaspora there is an extra tab on a friend's profile, which only shows the images that he made available. However, only Friendica and RedMatrix support the ability to access friends' photos in a convenient way and group photos together inside an album. If there is only the ability to share photos in a post, and when a user is a regular social network user, these photos will only be found when an eager user scrolls through a lot of status update. When having all these photos in a convenient place, users can easily share photos and still bridge the friendship by looking at photos of friends at any moment

[4]This feature can also be provided by specialised software, which does not provide an implementation of a social network and that is especially focused on secure message exchange.

without scrolling through a number of posts. However, during this research we looked at the default templates made available by the implementations. Therefore, it could be possible that with some other templates there is a functionality to group photos together and present this to the user.

3.2.5 Form of identity

All of the aforementioned implementations use a form of identity similar to *user@example.com*. The host part, e.g. *example.com*, is mostly used to allow communication between users that reside on different hosts. Some of the implementations use a GUID to identify a particular user, however the URI *user@example.com* is used by users to add a user to his circle of friends. This is also done to make it convenient for users to exchange usernames, instead of exchanging a long GUID. The host part of the URI, e.g. *example.com*, identifies the host a user resides on. If a certain implementation allows nomadic identities, which will be explained in Section 3.2.6, the host part will change. This allows the system to still be able to perform a lookup procedure on the host that the user currently resides on. It is therefore not possible with all of the implementations at the moment to have a nomadic identity with a username that will stay the same, even in the case when the user moves from one host to the other. A suggestion is included in Chapter 5 that describes how such a feature could be implemented.

3.2.6 Nomadic identities

Nomadic identities are identities who can roam between different hubs and are still able to exchange messages with its relations. However, this requires that the relations are also automatically updated to point to the new location where an identity resides. An identity is not bound to a single hub, as it can live across multiple ones, and update its current location in the decentralised network. If we want to offer the user a choice in selecting a node that he could use for his social activities, it would be beneficial for the user if he can moves his profile from one node to the other when the current node will go offline or when another node provides better services.

Such a functionality is provided by RedMatrix and has also been included in Friendica. With RedMatrix it is possible to migrate the content of the channel between the different hubs and update the primary location of a channel. Currently, only one hub can be assigned as the primary location, and all messages that are exchanged with that channel are sent to this primary hub. Migration, or cloning, can be done by providing the current location of the channel and its credentials. However, when the old hub is down this would not work any more. Therefore, another option has been included that allows one to migrate a channel. The user would need a file that contains the keys for that channel including

a list of the contacts. This file can be exported by a hub and can be saved on a device. Therefore, it is necessary to have access to this file before the old hub is down. With the use of the keys, which are provided by the user, the hub that the user wants to migrate his channel to can notify all the contacts of the new location. These migration messages can be verified at the receiving end by verifying the signature. If verification succeeds the location of the moved identity can be updated on the hubs of the user's friends. With Friendica it is also possible to move a profile to a different node and which basically follows the same approach as RedMatrix using the backup file. However, both with RedMatrix and Friendica the channel name or username will change, since the host part of the user's identity needs to be updated to point to the new location. With Diaspora it is not possible at the moment to move from one node to the other. However, on their FAQ [4] it is stated that this function will be implemented in the future. All the other implementations do not provide this functionality.

3.2.7 Proof of identity

GNU social, Diaspora, Friendica and RedMatrix use some form of authentication and validation of the identity when sending messages from one user to the other. When GNU social sends a message that mentions a user, it will use Salmon in combination with Magic Signatures [42], which contains a signature of the data. By validating the signature of the payload that was sent it can verify if this Salmon message indeed came from the identity that it claims to be. The public key is obtained using the draft[5] version of the WebFinger protocol [18]. Friendica will use DFRN for replies, likes and mentions between Friendica instances. They use a challenge mechanism that is used for user authentication. The data that is exchanged is not signed, which allows the data to be modified when the data is in transit and which can not be noticed by the receiving end. However, when RINO is enabled the packet will be encrypted and the packet could only have been encrypted, in theory, by the original author using the key pair that exists per relationship. Diaspora uses Salmon and its own protocol together to exchange messages. It is still possible to verify the Magic Envelope and thereby verifying the integrity and authenticity of the message. However, when a user of GNU social sends a public message or a reply it will not be sent using Salmon, it will use PuSH instead. The message will contain a Hub Signature, which is based on HMAC-SHA1 [29]. The secret key that is used for the HMAC is exchanged when the PuSH subscription was made. If SSL or TLS is not used, this key will be sent in cleartext.

The hub of the users who follow a RedMatrix channel will be notified when there is a new post ready to be fetched, even in the case of a public message. As noted on their wiki [8],

[5]Not only GNU social uses the WebFinger draft version for discovery purposes. Friendica and Diaspora use this version as well.

all messages should be signed, which includes the notification messages. If the message is later on fetched by the endpoint, the message will contain a signature and with this signature it is possible to verify that the content has not been modified in transit. The receiver can verify the message by obtaining the public key of the author of the message by using the Zot2 protocol. Messages in the Atom feed of a channel that are publicly available for everyone are not signed. However, the public feed of a hub, the public feed functionality as explained in Section 3.2.16, does include the signatures.

With pump.io a node should be first authenticated before it can send messages. Authentication is performed with OAuth and when a message is posted no use is made of OAuth Request Body Hash [25]. Since the content type of the POST message is not *application/x-www-form-urlencoded*, the signature that is included when the message was sent to the remote end does not apply on the body, which means that the signature can not be used to validate the information that is exchanged between the servers.

3.2.8 Message distribution

Message distribution is an important aspect from a scalability perspective. For example, in the second use case, as outlined in Section 1.3.1, a civil rights group wants to announce a new campaign. If a lot of users are following this profile, this announcement needs to be made available to the followers efficiently. From a scalability perspective, if messages are more efficiently routed to the users, the implementation would allow to have more users connected to this network without creating a bottleneck. Message distribution should be as efficient as possible and on the other hand create a consistent view of a persons feed amongst all of the followers. We have subdivided these two items. In this section we will talk about the message distribution and in Section 3.2.9 we will talk about consistency.

Two implementations, namely Friendica and pump.io, distribute the messages inefficiently. These two implementations send a message to every user even if these users live on the same host. A better message distribution algorithm would look at the destinations of the message and only send a message to the same server once. This approach is taken by RedMatrix and GNU social. In RedMatrix a *sitekey* [8] is used to encrypt the message and this encrypted message is sent to a hub. This hub can decrypt the message and distribute the message internally to its local users. With GNU social, PuSH technology is used to send a message to each server only once. It distributes the message locally just like RedMatrix. This is the case since the PuSH hub lives on the same host as where GNU social is installed. The approaches of RedMatrix and GNU social causes the sending host to send fewer messages to the hubs and results in a more efficient utilisation of resources on both hosts. However, there is an important difference between RedMatrix and GNU social when we look at the message distribution mechanisms. With RedMatrix the remote hub is notified that a new message is available and the remote host can fetch the data whenever it

wants. With GNU social, a push model is used and the whole message is sent to the remote host at once. In the case of RedMatrix the host is able to postpone the task of fetching the message if, for example, there is a high load on the hub. The remote hub is therefore more in control on how it wants to receive the messages. Diaspora uses another technique that lies somewhere in between the solutions implemented by Friendica and GNU social. Diaspora will efficiently route the messages that have been marked to be publicly available. However, with messages that are shared with some aspects, the messages will be directly send to the user (e.g. send the message multiple times to the same server when multiple users use that server). It therefore really depends on the use of aspects by the users if all the messages are efficiently delivered to the recipients.

3.2.9 Message consistency

Message consistency, as mentioned in Section 3.2.8, is another important aspect of a decentralised social network. With social networks users expect to have eventual consistency throughout the network in the order of messages. Temporary node unavailability may invoke race conditions and thus merge conflicts which makes such consistency more difficult. Most users will not notice if a post is missing for a few minutes, however it is important that the post will eventual appear in an acceptable time interval and in a consistent manner. In a decentralised social network it is harder to create such a consistency as nodes will be removed from the grid and nodes can be temporarily offline without an administrator noticing. It is therefore important that messages are queued and sent whenever possible. If a node is currently offline, for maintenance or for another reason, such a message should be delayed and a node should later on try to send the message again. However, all of the implementations showed inconsistencies when we tested the following scenario in our test environment. The specific scenario that we tested was with two hosts, namely A and B, and an exchange of comments between two users that lived on one of the hosts. The following steps were taken during these tests.

- User on host A makes a post available and comments on it
- User on host B comments twice on the post while host A is offline
- Host A comes back online, while host B is offline, and the user on host A comments twice on the post as well
- Both hosts are available again and could reach each other
- Both users comment twice on the post

Both Friendica and RedMatrix showed inconsistencies between the timelines of the two users on the different hosts. When we executed the above sketched scenarios the users' timelines would differ when there was a clock skew of a few minutes (e.g. two to four

minutes). During three of the four tests we saw with Friendica a different ordering of comments and with RedMatrix this has been observed with all of the four tests, while if we synchronised the clocks and ran the test ten times we would not see inconsistencies in the ordering of the messages. With Diaspora, with or without a clock skew, there would always be inconsistencies in the ordering of the messages and this has been tested five times. It seemed that when a message is queued after a delivery failure, and when after a while Diaspora tries to resend it, the message will be ordered on arrival time. This has been observed during all of the five tests. With GNU social we noticed that one of the users would always miss one message. This has been observed when ten messages were posted (e.g. item four was repeated a few times). In two out of the six cases we saw that both sides were missing a message. In an ideal scenario one wants to have eventual consistency. The ordering of comments on a certain topic should converge at a certain moment and the same ordering should be shown. Although the user might see that messages have been added and are shown before he made his comment, this also happens with the centralised social networks (e.g. a user wants to comment on a post but another user posted another comment earlier). However, with the current implementations the comments of a message that are shown on the timelines of different users are shown in a different order in the aforementioned scenario.

Pump.io seems to be the most affected by inconsistencies, as during normal operations not all comments from remote users were shown on a user's timeline. Packet captures revealed that the messages arrived on the host, and the host accepted the message, but still did not show up on the user's timeline.

However, with pump.io other inconsistencies have been observed as well. With pump.io the messages are directly sent to the user, which is also the cause why it does not efficiently distribute the messages as explained in Section 3.2.8. When a message has been sent and delivery has been unsuccessful it will not try to re-send that specific message (e.g. it does not have a queueing mechanism). It was observed that messages or even whole conversations went missing. One of the most noteworthy observations was in the case when a message was published on a node and the followers were located on another node. When the node of the followers went offline they would not receive the message. However, when this node came back online and the original author responded to the original posting, this comment would not be shown on the timelines of the followers. The log messages revealed that the author was notifying the followers of a new comment but since the original post was missing the comments could not be shown on the timelines of the followers.

With GNU social, Friendica, Diaspora, and RedMatrix messages are queued in the case of a message delivery failure. However, depending on the configuration of an implementation, it will drop messages after it has tried several times to deliver a particular message, which would result in inconsistencies on a user's timeline when the node is available again. However, with RedMatrix and Friendica an extra poller has been implemented to poll the

feeds once a day in case something went haywire during delivery (e.g. when a site was overloaded or when messages were dropped after a few attempts). Therefore, this extra polling mechanism can fetch some of the missed messages but the inconsistencies that were outlined before were not resolved. Within Facebook, this consistency is not guaranteed as well. There are algorithms that mix paid results with organic messages. Large users have to pay for reaching their entire audience.

However, with pump.io, Friendica, and RedMatrix it is possible to authenticate on a remote site while the account resides on another site. Since authentication and authorisation are both supported, the messages that have been targeted at one's identity can still be viewed when one is authenticated on the site of the author and looks at the profile of this author.

3.2.10 Message relay

If one looks at gossips that we get to hear from our real life friends, some of these messages are about people that the receiver of the gossip has no relationship with. This friend is relaying the message of the original story, since the receiver of the gossip did not hear the story from the original person. These kind of real life scenarios also apply to social networks. Assume that there are three persons, namely A, B, and C. B has a relationship with A and C, but A and C have no relationship with each other. If B posts a public message and A comments on it, the expected behaviour would be that C can see this comment. This is quite common in current centralised social networks. To provide this feature in decentralised social networks, the receiver of the comment should act as a relay server, since the one that comments on the post does not know about all of the relationships that B has. This relay option is implemented in Diaspora, Friendica, and RedMatrix. However, during some of the tests we have performed on GNU social, it was concluded that the relay function is not implemented and that the problem also applies to remote likes. However, if A and C were registered on the same server they could see each other comments. When the users were not registered on the same node, they would only see the comments of their own friends. With pump.io this feature has not been implemented as well, as there is still an open issue on their Github page [45].

3.2.11 Directory server

One of the core values of a social network is social interactivity. For having social interactivity it is a prerequisite to have a partner that one is having the social interactivity with. Therefore, it would be a desirable feature to have a directory server, or lookup server, to find one's friends by using their username (without the host part of the identity). With pump.io there is no functionality to search for friends and the implementation of GNU

social seems to be broken at the moment. However, wandering through the directory of *https://www.gnusocial.no/* it only shows accounts that are locally known to the hub. However, with Friendica there is the global directory and RedMatrix has the notion of directory servers that took the role of a directory server. With this function it is possible to search for users that are located across all hubs and that have chosen to publish his profile or channel to the global directory. With Diaspora it is only possible to search for information that is locally known, which includes searches that have been performed in the past. For example, when one searches for a remote friend named 'Alice' and he uses her Diaspora ID (i.e. with the @host.com part) the result will be cached by the Diaspora node. Any searches that only look for 'Alice' will also show Alice's profile of the remote host. Content that has been sent to the node, and marked as public, will also be visible during searches. However, as with the lookup of contacts, it is not possible to search for content that has not been cached and that resides on a remote host. Both Friendica, GNU social, and RedMatrix provide the ability to search through content that is locally known. This includes information that has been received from a remote hub, which was targeted to one of the users on the local node. However, with RedMatrix the polling message also polls public posts from hubs that it knows. These hubs are discovered when users start to create friendships with remote hosts. Therefore, when the user wants to discover posts from several nodes, it will receive much more remote content when it uses RedMatrix.

3.2.12 Unwanted messages

Unsolicited mail, also known as SPAM, can also occur with decentralised social networks. A desirable function would be to have some kind of filter to limit the number of unsolicited messages or to define if identities, which have no known friendship establishments with one, can send messages.

With pump.io one should follow someone in order to see notes from that user on his timeline. However, it is possible to target a specific user with a post, which will result that the post will show up on the timeline of the targeted user. If someone wants to send a message to a specific user, he should follow that user. However, everybody can follow someone and it is not possible to block a user from following him. It is a desired feature to be able to block someone and this feature seems to be requested as of March 2013, but still has not been implemented [1]. There is no remedy in this situation and a user is not able to delete the messages from his timeline. Therefore, SPAM could become a real problem in pump.io.

With Friendica and Redmatrix it is possible to configure the privacy options and specify what should be done with messages from users that are not in the circle of friends. However, with RedMatrix it is still possible to get overloaded with friendship requests, which could be seen as unsolicited messages. With Friendica it is possible to limit the number of friendship

requests that one receives per day.

With GNU social it is possible to mention a user in a notice. This user will be notified of this event and the notice will appear on his timeline. However, this is even possible when the user that is receiving the SPAM is not following that identity. The identity that is causing the SPAM is not required to follow the target of the SPAM either. The main problem is that users on other hosts can not be blocked at the moment what could open up the possibility of receiving unsolicited messages.

Diaspora has the same problem as with pump.io and GNU social. If a user is mentioned in a post the user will be notified of this event and it will show up on his timeline. It is not possible to specify that a user should first be added to an aspect before he is allowed to mention this user and that the notification would only be shown on the timeline of the mentioned target when he has been added to an aspect of the target. It is however possible to block a user but this should be done once the user detected that it is a misbehaving account.

3.2.13 Reputation system

Friendica makes use of the DFRN protocol for the exchange of messages. In the DFRN protocol specification, as explained in Section 2.2.1, there is a reputation system where friends are asked about the reputation of an identity. This information can be used to decide if the user needs to accept this friendship request. However, in the current implementation of Friendica, no such reputation system seems to be implemented.

However, the RedMatrix system makes use of Zot2 as explained in Section 2.2.10. The implementation of RedMatrix provides a reputation system for users. A user can give other users a grade and note why this grade has been given. This information can be seen by all other people and can be used by a user to make a decision if it should accept a certain friendship request.

No reputation system is implemented in GNU social, Diaspora or pump.io nor described in the used protocols of these implementations.

3.2.14 Integration with social networks

Friendica is the social network with most integration possibilities with the described decentralised social networks and existing social networks like Facebook and Twitter. By using plugins it is possible to connect with all of the other implementations. However, it can cause extra resources to share all of these posts to the different social networks.

With RedMatrix it is possible to communicate with Diaspora, GNU social and Friendica, although this is currently noted as experimental so a user should be aware of this. It is possible with plugins to let Friendica and RedMatrix communicate with pump.io although privacy can not always be guaranteed as one is used to in RedMatrix.

3.2.15 Hidden contacts

In one of the use cases it has been outlined that users could benefit from the fact that not all of their contacts can be seen by the public. With Diaspora it is possible to specify if users in an aspect can see the other users that have been added to the same aspect. This seems to only work on local nodes. When we used Diaspora and added a remote user to an aspect that user could not see all the other users that have been added to the same aspect, while this option has been enabled. This option only applies to aspects, it is therefore not possible to specify that a specific user can not see all the other contacts or that a specific user can not be seen by all other users in the same aspect. However, with Friendica and RedMatrix it is possible to hide contacts, but this depends on the subscribed user or channel as well (one might hide its relationship but the relationship can still be seen on the other end of the relationship). With Friendica it is possible for the remote end to hide a certain relationship (e.g. the relationship is not shown on their profile). With RedMatrix it is not possible to specify this per relationship, but specify it for the whole channel (e.g. all or nothing, people can see all the relationships or nothing). With GNU social and pump.io it is not possible to specify that a friendship should be marked as private, which makes it possible for the public to see the user's friendships.

3.2.16 Public feed

For search engines, or websites that aggregate information about certain topics, it would be beneficial to have an option of crawling through public posts. With RedMatrix it is possible to pull all public posts from a hub by issuing a GET request on, for example, *example.com/zotfeed*. If additional parameters are supplied during the GET request, it can be specified that only posts that have been published from a specific moment in time should be shown. GNU social provides a similar feature and makes an Atom feed available where the public posts are listed. It is not possible to specify from which moment in time the posts should have been published. However, it will provide a limited set of public posts. Pump.io provides another approach and makes it possible to push all public posts to a firehose [44]. Applications can use the information made available by the firehose to look for the information it is interested in. Diaspora and Friendica, as far as can be seen after analysis of the code and a live implementation, only provides means to see the public posts of a single user through the use of a feed. Therefore, the application needs to know

beforehand from which users it wants to crawl the public posts.

3.2.17 Summary

Tables 3.3, 3.4, and 3.5 provide an overview of all the projects and their functions. The symbols that are used in the tables have the following meaning.

- -: Not provided, broken, inconsistent, or buggy
- *v*: Is somewhat implemented
- *+*: Is implemented and provides limited options
- *++*: Is implemented and provides the users with a lot of options

Project	Privacy	Admin int.	Encryption	E2E enc.	Photo wall	Nomadic id.
Diaspora	+	++	++	-	+	-
Friendica	+	+	v	-	++	++
GNU social	v	+	-	-	-	-
pump.io	v	-	-	-	-	-
RedMatrix	++	+	++	++	++	++

Table 3.3: Summary of projects

Project	Proof of id.	Distribution	Consistency	Relay	Directory
Diaspora	++	+	v	++	v
Friendica	v	-	v	++	+
GNU social	++	++	v	-	v
pump.io	-	-	-	-	-
RedMatrix	++	++	v	++	++

Table 3.4: Summary of projects

Project	SPAM	Reputation	Integration	Hidden contacts	Public feed
Diaspora	+	-	-	+	+
Friendica	++	-	++	++	+
GNU social	-	-	-	-	+
pump.io	-	-	-	-	+
RedMatrix	++	++	++	+	++

Table 3.5: Summary of projects

3.3 Implementations with a different approach

IndieWebCamp is a community and movement that wants to create an alternative to the 'corporate web'. They have certain principles [12] that should be obeyed by the Indie Web projects. One of the most important ones is that users own their data and post copies of the data to silos, which is the POSSE approach [11]. It is also possible to do it the other way around, publish on a silo and then syndicate to one's personal website, which is named PESOS. There is also the possibility, for when one does not have a site, to syndicate everything to a particular silo, which is named PESETAS. These silos can be, for example, Twitter or Facebook. Although the project itself is not focused on creating a decentralised social network, it provides however means to do so. There are two interesting projects, namely Red Wind [37] and Known [36], which are respectively a microblogging platform and a social publishing platform. These two platforms provide means to create content, share it with users, and have interactivity based on this information exchange (e.g. like a photo or comment on a post).

Microblogging is the exchange of smaller messages, a function provided by the typical social networks like Facebook and Twitter. With Red Wind it is for example possible to publish posts and comment on them. The current functions provided by Red Wind are relatively similar to the current social networks, however it provides lesser privacy as with the other projects. It is not really possible to create a private conversation, or a group conversation for that matter, between different friends. There is not really a function to group people together and target a certain message to these people. These functions are however not always typical in a microblogging environment. However, these are desired features in a social network.

With Known one can also publish content and mention other users. Interactivity can be created by exchanging comments on a post and starring posts. There are also options to publish other content like photos and check-ins. However, the same problem exists as with Red Wind that not enough options exist to guarantee privacy. Posts are public and there is not an option to exchange private messages.

It might not be obvious at first sight why these projects are mentioned, however they have some similarities with decentralised social networks. First of all, it is possible to have interactivity with people on different hosts with the typical short messages. The protocol mostly used is Webmention to notify the users and uses PuSH notifications for new posts in the case of Red Wind. With Bridgy [2] it is possible to syndicate the content to Facebook and Twitter. The aforementioned features are also the features that one expects on typical centralised social network websites. In the case of Red Wind authentication is provided by IndieAuth [9], which allows users to use one's own domain name to sign in to websites. Replies are mostly to other microblogging posts and not really to a certain user. As the microblog posts are most likely to be generated on a user's own website, the user keeps the

same identity and uses IndieAuth for logging in to the website. As the posts stays on the same website and therefore the reference to these posts and the user, a user can migrate from one host to the other as the DNS records are most likely to be changed as well.

However, the current solutions are not really suited to be used as a decentralised social network but it could be interesting for other implementations to look at their approach. The biggest problem would be the message distribution as the general idea behind this concept is that every user has its own website. If a lot of friends are mentioned in a topic, it could not be as efficiently routed as RedMatrix for example. Although the idea behind Red Wind and Known is not necessarily the same as social networks, it provides some interesting insights how decentralised communications take place in an other landscape.

3.4 Use cases

In Section 1.3.1 three use cases were shown that should be supported by the implementations. In this section we will outline how these use cases are supported by the implementations and what the limitations are.

3.4.1 First use case

The first use case was about a group of friends in a bar who wanted to share photos privately with each other. The aforementioned implementations support such a use case. With pump.io it is possible to share photos to a list, which could be the group of friends. However, it is possible for these group of friends to share this post, which contains the photo, in a single click to all of their followers. As these photos should stay private, it should not be be possible to share this photo from within the web interface. With GNU social[6], Diaspora, Friendica, and RedMatrix this would not be possible as messages that are marked as private can not be (re-)shared. We only want to add one side note here, it is of course possible to make a screenshot of such a post and share this screenshot or just make a copy of the photo and share it. However, such use cases can not be prevented and would pose a problem for all of the platforms.

3.4.2 Second use case

With the second use case an identity wants to share information with a big audience. It wants to announce, for example, that a new campaign is launched. Such a message that needs to be sent to a lot of people might only be sent to a specific set of people and might

[6]Private messages can only be shared to local users at this moment.

not be publicly available, as the audience might not want to publicly acknowledge that they support the campaign. With all of the implementations it is possible to send a message to a specific set of people, however if the message needs to be marked private and sent to users on remote hosts then this would not be possible with GNU social. As pump.io and GNU social do not provide the relay functionality, it would not be possible for all users to see all of the comments of a certain post (in the case that not everyone is following each other). However, as outlined in Section 3.2.8, the message distribution model used by the different implementations differ quite from each other. It is therefore important, for identities that have a lot of followers or friendships, to use an implementation that efficiently distributes these messages (i.e. RedMatrix).

3.4.3 Third use case

The third use case describes a scenario where a citizen journalist wants to share a short status update about emerging news to an audience and wants to have a private conversation with a fellow journalist. This is supported by the implementations to a certain extent. As with the previous use case, it is important to have an efficient message distribution model, which is not provided by all of the implementations. However, it would still be possible to reach a big audience, albeit that it is done less efficiently (e.g. could cause messages to be received later than with efficient message distribution models). A more efficient message distribution model also translates back into how soon people can respond to these messages, as they first need to have received the message before they can respond to it. As explained in the previous section, some implementations do not provide the relay functionality, which causes that not all of the comments of a certain post might be available.

The private conversations with one another are limited in certain scenarios. With Diaspora, Friendica, and RedMatrix it is possible to send private messages. It could be a one-to-one correspondence of the conversation and only the sender and recipient will see the messages[7]. However, with pump.io messages are not well protected and the fellow journalist could easily share the message exchange by accident or on purpose. With GNU social the implementation of direct messages seems to be a bit unstable, which results that a private message exchange is not possible at this moment. It is also important that the relationship between the journalists can be hidden, which is possible with Diaspora, RedMatrix and Friendica. With Friendica and RedMatrix it is also possible for the journalists to create different profiles, and different channels, that are only used between a certain group of journalists. By having different profiles and channels, the user is in total control about which information may be made publicly available. Another solution is to use end-to-end encryption that is supported by RedMatrix as explained in Section 3.2.3. With such a solution only the journalists are able to see the original content of a message.

[7] We exclude the possibility that an administrator looks at the messages in this scenario.

The use case also outlines that it is important that this news update can be discovered by news websites. In Section 3.2.16 it is outlined that all of the implementations provide feeds to crawl the publics posts of users. However, with pump.io, GNU social and RedMatrix it is possible to get all the public posts of a single website. With RedMatrix it is possible as well to specify that it only wants to receive posts that have been published from a specific moment in time.

3.5 Standardisation

Some of the implementations are currently compatible with each other albeit with wrappers that makes these platforms compatible. Although some of the implementations use the same protocols, for example Salmon, their implementations differ and the exchange of the Salmon slap differ as well. One of the biggest problems are how messages can be exchanged between the different platforms and how the lookup and discovery procedure takes place. Some of the implementations follow the WebFinger protocol and others use their own implementation. It would be beneficial for users of the different social networks that more standardisation takes place. Both in the message exchange and in the lookup procedure for the users. This also applies to the APIs that differ amongst the implementations.

It would be better to have a general API, with a standard message structure, that could be used for interconnectivity between different platforms. Another API might be available for interconnectivity between the same platforms providing, for example, a richer feature set to the users. However, if the basic feature set of social networks is supported by the API that is used between different platforms, numerous scenarios would already be covered. Being able to communicate with another platform also means that its users should be authenticated, which also implies that some form of authorisation should take place. The current models for authentication, authorisation, and privacy are implemented differently and no real standard is used. A standard should allow for remote authentication and authorisation so that users can view the profiles of friends on a remote host. The content that is presented to the user should only be the content that the friend has explicitly allowed to be visible for this user. This is for example provided by Friendica and RedMatrix. It will be hard with the current implementations to create a standard for authentication, authorisation and a coherent view of privacy as each project has a slightly different philosophy and therefore a different implementation. However, a single standard for privacy in the form of an ACL could be used to make different systems interoperable. The more advanced privacy features might only be available between the same implementations.

Another important aspect that has not been standardised and adopted is the form of identity, identity proof, and secure message exchange. This causes interoperability issues as all of the implementations use a different standard or their own implementation. A general

standard should be created, or used, to overcome this problem. A standard should also account for efficiently distributing the messages across the decentralised social networks. As was shown in Section 3.2.8, some of the implementation do efficiently route the messages. The solution of RedMatrix is to use a *sitekey* for encryption so that a message for multiple recipients on the same server only has to be sent once to each server. The message is then internally distributed to its intended recipients. Other social networks use yet another message distribution mechanism. In the standard it should also be specified how the messages can be validated by, for example, using signatures. The form of an identity is how a user is represented in the decentralised social network. Currently the implementations have a form of *user@example.com* where *user* is the username known locally to the host that is represented by *@example.com*.

Something that also needs to be accounted for is the misuse of a decentralised social network. With some of the current available implementations it is relatively easy to send SPAM to different users. A standardised system should be used that can detect this behaviour and temporarily block a user or perform another action such as notifying an administrator. With the current centralised social networks it is easier to regulate this. We do not want to have a regulatory authority that detects accounts that are sending SPAM messages. However, misbehaving users should be easily blocked from a user's perspective or the network's perspective. It should be possible to block a user without the participation of an administrator on another host. The reason that it is not always desired to only let an administrator block a user is because the misbehaving user can well be the administrator of the hub.

The main problem is that there are currently too many protocols to choose from, that protocols have been slightly modified, or that projects have created their own protocol that has not been adopted by other social networks, which causes interoperability issues. As can be seen in Section 2.2 there are a lot of protocols that facilitate the exchange of messages in decentralised social networks, but none of the implementations use these protocols in such a way that interoperability exists. Because of this discrepancy, no real out-of-the-box compatibility exists between implementations without the use of bridges or wrappers. The aforementioned paragraphs mentioned what kind of protocols should be standardised and adopted by the community.

3.6 Solution for hosting providers

The focus of this research is to analyse the various implementations and analyse which implementation is most suited to be provided as a service by a hosting company. This should also include the mid-size web hosting companies, which mostly provide hosting for ASP or PHP websites. Friendica, GNU social, and RedMatrix are programmed in PHP

and use some extensions of PHP and some modules of Apache2 (another webserver could be used as well). This should not be a real problem for most web hosting companies. However, Diaspora is written in Ruby on Rails and pump.io is written in Node.JS that might not be supported by all of the web hosting companies.

Currently, the implementation that is most suited as an alternative to a centralised social network is RedMatrix. Other implementations pose some problems when it comes to message distribution efficiency, in the case of pump.io and Friendica, or are missing some interesting features that are provided by RedMatrix. Where GNU social and Diaspora provide a better message distribution mechanism than pump.io and Friendica, they are however more prone to SPAM. Federation in GNU social does not seem to be working in certain use cases as was outlined in Section 3.2.1 and 3.2.10. RedMatrix also provides better privacy mechanisms that is completely in control of the users. As can be seen from Section 3.1 RedMatrix also ships with a lot of features, is really focused on privacy, and has a great message distribution mechanism.

However, when a hub serves a lot of users it needs to scale out. This has been tested with RedMatrix in a test setup using HAProxy, a set of RedMatrix instances, and a MySQL back end. During the tests no problems were encountered in running such an instance and *magic auth* still worked as expected. However, if the web hosting company experiences scalability issues with other back ends, for example with MySQL, it would still be possible to set up another autonomous node. The web hosting company could ask some of the clients to move to the new autonomous node, which would not be a problem since RedMatrix supports nomadic identities. Even without further adjustments on-site by web hosting companies, installing RedMatrix is possible for users as it runs on the ubiquitous LAMP stack. It is also end-user upgradeable. Given the above, we consider RedMatrix to be the most suited candidate that can be provided as a service by web hosting companies.

Chapter 4

Conclusion

A variety of functionalities exist in the typical social networks like Facebook, Google+, LinkedIn, and Twitter. The reason why social networks are used by people, what the most important functions are, and the basic feature set that should be supported by the decentralised social networks are outlined in Chapter 2. The features that are supported by all of the existing large social networks are: broadcasting a message, receiving notifications of incoming messages, redistributing a message, reply or publish a comment, publish a profile and view profiles, liking a post or comment, subscribing to messages from a user, and partitioning an audience in subsets. The basic feature set, which needs to be supported by the decentralised social networks, consists of the aforementioned features. The social networks might put restrictions on certain items, like a limitation on the number of characters one could use for his message. As these limitations give the social network a certain goal, for example sharing short status updates, the limitations itself are not a new feature. These use cases could therefore still be supported by this basic feature set.

There are a number of open source projects that could be used as an alternative to the centralised social networks. Some implementations were considered out of scope for a variety of reasons. The ones that are mature enough and provide the basic feature set are Diaspora, Friendica, GNU social, pump.io, and RedMatrix.

All of the implementations provide the same basic feature set, although the implementations differ quite from each other. A functional breakdown is given in Section 3.2 and shows that some of the implementations provide extra features or provide an improved version of these features compared to the other implementations. RedMatrix, Friendica and Diaspora provide the most features and really puts the user in control of their data, which includes the privacy of the user and their data. However, the message distribution model and extra features provided by RedMatrix seem to make it most suited as an alternative to the current social network landscape. The message distribution model of RedMatrix is better compared to Friendica's and Diaspora's message distribution model, which is quite important in a decentralised solution. RedMatrix, which has been focused from the start on privacy, also supports nomadic identities. This allows users to move their identity between hosts if the user wants to migrate from one host to the other and puts the user in a position to choose a hub that provides the best services. With the directory server the user can easier lookup his friends and discover remote content as remote friendships are made on the hub. The only problem that all of the implementations have is that consistency can not be guaranteed when there is a clock skew in the scenario that we tested, as is explained

in Section 3.2.9.

Among the implementations that exist at the moment, RedMatrix is currently most suited to be provided as a service by hosting providers. The solution can be scaled and also allows a hosting company to create multiple autonomonous nodes where the users can move between these nodes. If a user decides to switch from a hosting company it would not jeopardise the relationships between users, since relationships will be maintained if a user migrates between the hosting companies.

To conclude the research, RedMatrix is currently most suited to be provided as an alternative to the current centralised social networks, which can be provided as a service by hosting companies. It is currently more mature than some of the other implementations and puts the user back in control of their data.

Chapter 5

Recommendations

In this chapter we will give some recommendations or suggestions to improve the examined decentralised social networks. Some of these features will benefit their user community and others are more related to the efficiency of the message distribution model applied in these networks.

5.1 Permanent usernames

With the current available implementations there is not a possibility to have a permanent username. This is due to the lookup procedure that uses the username to locate the current node the user is using. However, with one extra level of indirection in Zot2 this could be solved. The reason for choosing Zot2 is that it is used by RedMatrix, efficiently routes messages, and supports roaming profiles. We suggest to introduce an additional username for Zot2, called the primary username that will stay the same over the lifetime of the identity[1]. The second username is the temporary username and has the same functionality as the current usernames (e.g. a username that also identifies the current location of the user). We assume that the user has a website if he wants to use a primary username and if he does not have one, he can not have a primary username and will use the current model of having an assigned username (e.g. the user will only have a temporary username and the location of the identity is included in the username).

The user will give the primary username to all of his friends and which will be used for establishing friendships. The primary username is used to get the temporary username, which is used to locate the identity and subsequent communications, and may well have a completely different username to avoid clashes. The mapping between the primary and temporary username is provided by using WebFinger, as the protocol is suited for this and can be used to get more information about the primary username. WebFinger should be provided by the user's website, but this could be implemented by having a trivial script that looks at the resource in the GET request and responds with a static answer. The *rel* is a well known URI used by the decentralised social networks and the *href* will contain the temporary username, which is also an URI. The WebFinger discovery procedure will then proceed with the temporary username, just as it would have done with the current implementations.

[1] However, there should be a possibility to update a primary username if the user wants to.

When a friendship request is established it should be specified if the username is a permanent or a temporary username. This should be specified to support the use case when an identity moves from one location to another one while supporting the scenarios where a user might only have a temporary username or having both (e.g. a primary username and a temporary username). If a user moves from one host to another host he will notify his friends of this event. With the GUID, which also stays the same, and with the key pairs it can be verified that it is still the same user. When this happens the temporary username needs to be updated, to point to the new location, and the relationships should be notified of this event. The WebFinger responses should be updated accordingly to map to the new temporary username. A use case where an identity is available on multiple locations is also supported, as it is possible with WebFinger to have multiple links that have the same *rel* but a different *href*.

The reason for using the temporary username for subsequent communication has to do with location discovery. The temporary username reveals the location of the user. This location is used for the information exchange, which requires that a connection is set up with the hub. This can be done by resolving the host part of the temporary username with a DNS lookup. The temporary username should not be shown to the friends, instead the primary username should be shown to the user whenever possible (e.g. the user has a primary username).

5.2 Message distribution

Some of the message distribution models used by the various implementations are more effective than others. However, depending on the adoption of decentralised social networks by users and the instances running, it could become a problem to distribute all of the messages to people. When a lot of hubs are installed and a lot of messages are sent, bottlenecks could occur. In FETHR [48], as explained in Section 1.1, a gossip system is proposed to gossip a message to subscribers. Such an implementation might improve the efficiency of distributing the message to the subscribers as some subscribers can distribute the messages for the author of the post. This would however require changes in the implementations of the current decentralised social networks.

The only problem that exists with such an approach is trust. There could be a scenario where the subscribers may not know who all the recipients are of a certain post. This could be solved by letting the author distribute the messages for these kind of scenarios. However, while trust is a problem it could also be used as a solution, as it is likely that some friends are more closely related to a user than others. If trust can be used as a model to distribute the messages, even the ones that should not be shared with others and where we trust on the discretion of a friend, it would be possible to let some friends distribute

the messages.

This could be solved by having a session key like the one implemented in Megaphone [43]. It is more likely that a person will send messages to a certain group, which can be represented by aspects, lists, channels, or groups. A group can share a session key that is known to all of the participants of the group. When a message is sent to a specific group it can be signed by the original author (i.e. the sender) and encrypted by using the session key. When the message needs to be distributed to such a group, it only needs to be sent to a few friends. Since the session key is shared to everyone, and the recipients are included in the message, the message can also be distributed via the friends. These friends can then also send the message and the recipients can still verify that the message came from the original author by verifying the signature. It is however important to have consistency, such that the message will reach all of the participants. It is therefore required that the original author is notified by the friends if all delivery attempts have been successful, this message could contain a signature to validate the message of the friend.

However, in ad-hoc scenarios where the group was not created beforehand there are two options. One option is to create a group, create a session key, and share this session key. Another approach is based on public key cryptography, when a message needs to be sent to a set of people only once and creating a group is not possible or desirable. In the latter case we suggest that every identity has a key pair. The message is signed by the author's private key such that receivers can validate the message and can verify that the message indeed originated from the author. The public key of the recipients hub can be used, like the *sitekey* in RedMatrix, to encrypt the message. An envelop would be necessary to specify the destination and we rely here on the discretion of the friends to not make this public. The only information that is revealed to the friends are the targets of the message (i.e. the envelope). If this is acceptable for certain messages this method could be used. There is only one problem with such an approach, as the message needs to be encrypted for every hub, like RedMatrix does, which costs processing time.

It would be more efficient to use a session key, such that a message only needs to be signed with the private key and encrypted with the session key once. Whereas, with the other suggestion the message needs to be signed and possibly encrypted for every hub. When persons are removed from the group a new session key needs to be generated and shared amongst the participants, such that the person that left the group can no longer decrypt new message exchanges.

Chapter 6

Future work

Decentralised social networks are an interesting topic and a lot is still unexplored. It would therefore be interesting to see other research that is also focused on decentralised social networks. In this chapter we will outline what research can be performed on decentralised social networks.

6.1 Deadlock

Depending on how decentralised social networks will evolve, it could be interesting to see if a deadlock could occur in the network. If a lot of users will use the decentralised social network, it would be likely that a lot of servers will be made available to support this user base. However, if more users get dispersed over numerous hosts, more messages need to be exchanged between these hosts as relationships will be created between users on different hosts. There would be a lot of duplicate messages, since every message that needs to be sent to a remote host will be stored on that remote host as well.

As people comment on posts from other users on remote hosts, notifications will be sent to these remote hosts. This will not only cost bandwidth, but also storage. A not truly far fetched scenario could occur where a deadlock between different systems occur if a server has no storage space available. If a relay server needs to send a received comment to all of it subscribers, and where delivery failed because some of these servers have no storage space available, the relay will try to resend this comment later on. When a popular post is made public and a lot of people respond on it, it could however swallow up a lot of resources if queues start to grow. It would be interesting to see if the implementations discussed in this paper have some kind of mechanism to prevent this and if such a scenario could happen in real life.

6.2 Security

In this paper we have described some of the security mechanisms that are implemented in a number of decentralised social networks. However, during this research we have not really examined how the developers implemented these security mechanisms. It would therefore be useful to see if the security mechanisms have been correctly implemented and

no vulnerabilities exist in the implementations. A good security audit will benefit the community of these decentralised social networks. Note that some of the aforementioned implementations already had some security deficiencies in the past [38, 46, 10].

6.3 Benchmark

In Section 3.2.8 it has been outlined that the implementations use different forms of message distribution. In the same section it has also been outlined which implementations used the more efficient models (e.g. look at the destination and only send the data once for each server). It would be interesting to see what the performance is of the different implementations that use the efficient model of message distribution. The benchmarks also need to look at what the performance of difference setups are. Different setups could vary from a lot of users on a few nodes to a setup where users are scattered amongst a lot of hosts.

6.4 Stale data and accounts

In Section 3.2.9 it has been outlined that queues would drop messages after they have tried to deliver them several times. This would lead to data inconsistencies but could also lead to stale data and accounts in the database. In RedMatrix and Friendica it seems that accounts are archived after a certain time, such that new messages are not sent to archived user accounts. If a user comes back online the users could synchronise using the pollers. In GNU social the PuSH system uses lease times for subscriptions. It would be interesting to see how common it is that a node is removed from the grid and how the different implementations handle such a scenario. Implementations that still try to send data to a node that has been offline for days will cause unnecessary overhead.

6.5 Proof of concept of suggestions

In Chapter 5 we have made some recommendations and suggestions for decentralised social networks to implement. It would be interesting to see if a proof of concept could provide insights if these recommendations are possible and if their use would be more efficient than the current solutions. However, it would require some fundamental changes to the current implementations.

References

[1] Block a person - Issue #450 - e14n/pump.io. https://github.com/e14n/pump.io/issues/450 Accessed on: 2015-06-19.

[2] Bridgy - IndieWebCamp. https://indiewebcamp.com/Bridgy Accessed on: 2015-06-19.

[3] Builtin Automatic Encryption. https://github.com/redmatrix/redmatrix/blob/master/doc/encryption.bb Accessed on: 2015-06-16.

[4] FAQ for users - Diaspora*. https://wiki.diasporafoundation.org/FAQ_for_users Accessed on: 2015-06-16.

[5] Federation protocol overview - diaspora* project wiki. https://wiki.diasporafoundation.org/Federation_protocol_overview#Post_the_message_to_Bob Accessed on: 2015-06-22.

[6] GNU social and GNU FM. https://gnu.io Accessed on: 2015-06-19.

[7] Help: Features. https://redmatrix.me/help/features Accessed on: 2015-06-24.

[8] Help: zot. https://redmatrix.me/help/zot Accessed on: 2015-06-22.

[9] IndieAuth - Sign in with your domain name. https://indieauth.com/ Accessed on: 2015-06-19.

[10] Many bad symmetric cryptography implementations in include/crypto.php. https://github.com/friendica/friendica/issues/1655 Accessed on: 2015-06-25.

[11] POSSE. https://indiewebcamp.com/POSSE Accessed on: 2015-06-27.

[12] principles - IndieWebCamp. https://indiewebcamp.com/principles Accessed on: 2015-06-19.

[13] pump.io by e14n. http://pump.io/ Accessed on: 2015-06-19.

[14] redmatrix. https://redmatrix.me/ Accessed on: 2015-06-22.

[15] The diaspora* Project. https://diasporafoundation.org Accessed on: 2015-06-19.

[16] The internet is our social network — friendica. http://friendica.com Accessed on: 2015-06-19.

[17] Users should be able to like comments again #2999. https://github.com/diaspora/diaspora/issues/2999 Accessed on: 2015-06-29.

[18] WebFinger Protocol. https://code.google.com/p/webfinger/wiki/WebFingerProtocol Accessed on: 2015-06-16.

[19] Webmention - IndieWebCamp. http://indiewebcamp.com/Webmention Accessed on: 2015-06-22.

[20] Altmann, Alexander. Vergleich und Bewertung Sozialer Netzwerke im Hinblick auf Architektur, Sicherheit, Datenschutz und Anbieterunabhangigkeit. 2013.

[21] Apollic Software, LLC. About Tent. https://tent.io/about Accessed on: 2015-06-13.

[22] Apollic Software, LLC. Protocol Introduction Tent. https://tent.io/docs Accessed on: 2015-06-13.

[23] ARPA2 project. ARPA2.net projects overview, 2015. http://www.arpa2.net/ Accessed on: 2015-06-03.

[24] Atkins, Martin and Norris, Will and Messina, Chris and Wilkinson, Monica and Dolin, Rob. Atom Activity Streams 1.0. http://activitystrea.ms/specs/atom/1.0/ Accessed on: 2015-06-13.

[25] Eaton, Brian and Hammer-Lahav, Eran. OAuth Request Body Hash. https://oauth.googlecode.com/svn/spec/ext/body_hash/1.0/oauth-bodyhash.html Accessed on: 2015-06-22.

[26] Ellison, Nicole B and Steinfield, Charles and Lampe, Cliff. The benefits of Facebook friends: Social capital and college students use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.

[27] Eric Vicenti. OpenBook. https://github.com/ericvicenti/OpenBook Accessed on: 2015-06-14.

[28] Facebook. Company Info — Facebook Newsroom, 2015. http://newsroom.fb.com/company-info/ Accessed on: 2015-06-11.

[29] Fitzpatrick, Brad and Slatkin, Brett and Atkins, Martin and Genestoux, Julien. PubSubHubbub Core 0.4 – Working Draft. http://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html Accessed on: 2015-06-13.

[30] Friendica. What happened to Zot? — friendica. http://friendica.com/node/24 Accessed on: 2015-06-12.

[31] Friendica. zot friendica/red wiki. https://github.com/friendica/red/wiki/zot Accessed on: 2015-06-12.

[32] Hammer-Lahav, E. and Cook, B. Web Host Metadata. RFC 6415 (Proposed Standard), Oct. 2011.

[33] John Panzer. Draft: The Salmon Protocol. http://salmon-protocol.googlecode.com/svn/trunk/draft-panzer-salmon-00.html Accessed on: 2015-06-13.

[34] Jones, P. and Salgueiro, G. and Jones, M. and Smarr, J. .

[35] Juste, P.S. and Wolinsky, D. and Boykin, P.O. and Figueiredo, R.J. Litter: A Lightweight Peer-to-Peer Microblogging Service. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 900–903, Oct 2011.

[36] Known, Inc. Known: create a single website for all your content. https://withknown.com/ Accessed on: 2015-06-19.

[37] Mahan, Kyle. Red Wind. https://github.com/kylewm/redwind Accessed on: 2015-06-19.

[38] McKenzie, Patrick. Security Lessons Learned From The Diaspora Launch. http://www.kalzumeus.com/2010/09/22/security-lessons-learned-from-the-diaspora-launch/ Accessed on: 2015-06-25.

[39] Mike Macgirvin. DFRN–the Distributed Friends & Relations Network, 2010. https://macgirvin.com/spec/dfrn2.pdf Accessed on: 2015-06-12.

[40] Nadkarni, Ashwini and Hofmann, Stefan G. Why do people use Facebook? *Personality and individual differences*, 52(3):243–249, 2012.

[41] Nottingham, M. and Sayre, R. The Atom Syndication Format. RFC 4287 (Proposed Standard), Dec. 2005. Updated by RFC 5988.

[42] Panzer, John and Laurie, Ben and Balfanz, Dirk. Magic Signatures. http://salmon-protocol.googlecode.com/svn/trunk/draft-panzer-magicsig-01.html Accessed on: 2015-06-16.

[43] Perfitt, T. and Englert, B. Megaphone: Fault Tolerant, Scalable, and Trustworthy P2P Microblogging. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 469–477, May 2010.

[44] Prodromou, Evan. Firehose for the federated social web. https://github.com/e14n/ofirehose Accessed on: 2015-06-25.

[45] Prodromou, Evan. Redistribute comments to other recipients - Issue #479 - e14n/pump.io. https://github.com/e14n/pump.io/issues/479 Accessed on: 2015-06-22.

[46] Prodromou, Evan. Security alert: cross-site scripting vulnerability in pump.io 0.2.x and 0.3.x (master). https://e14n.com/evan/note/m0B0MlAySEiC8ey4aQ6nPA Accessed on: 2015-06-25.

[47] Prodromou, Evan, and Vibber, Brion and Walker, James and Copley, Zach. OStatus 1.0 Draft 2. http://www.w3.org/community/ostatus/wiki/images/9/93/OStatus_1.0_Draft_2.pdf Accessed on: 2015-06-13.

[48] Sandler, Daniel and Wallach, Dan S. Birds of a FETHR: open, decentralized micropublishing. In *IPTPS*, page 1, 2009.

[49] Shetty, Sandeep and Parecki, Aaron and Walter, Barnaby. Webmention 0.2 (RC1). https://github.com/converspace/webmention/blob/master/README.md Accessed on: 2015-06-13.

[50] Smarr, Joseph. Portable Contacts 1.0 Draft C. http://portablecontacts.net/draft-spec.html Accessed on: 2015-06-13.

[51] Snell, J. Atom Threading Extensions. RFC 4685 (Proposed Standard), Sept. 2006.

[52] Snell, J. The Atom "deleted-entry" Element. RFC 6721 (Proposed Standard), Sept. 2012.

[53] The Libertree Project. Libertree Specification. https://github.com/Libertree/libertree/blob/master/specification.md Accessed on: 2015-06-13.

[54] Thiel, Simon and Bourimi, Mohamed and Gimnez, Rafael and Scerri, Simon and Schuller, Andreas and Valla, Massimo and Wrobel, Sophie and Fr, Cristina and Hermann, Fabian. A Requirements-Driven Approach Towards Decentralized Social Networks. In J. J. (Jong Hyuk) Park, V. C. Leung, C.-L. Wang, and T. Shon, editors, *Future Information Technology, Application, and Service*, volume 164 of *Lecture Notes in Electrical Engineering*, pages 709–718. Springer Netherlands, 2012.

[55] Thurston, Dr. Adrian D. DSNP: A Protocol for Personal Identity and Communication on the Web. http://www.colm.net/files/dsnp/dsnp-overview.pdf Accessed on: 2015-06-13.

[56] Twitter, Inc. Favoriting a Tweet. https://support.twitter.com/articles/20169874-favoriting-a-tweet Accessed on: 2015-07-06.

[57] Xu, Tianyin and Chen, Yang and Fu, Xiaoming and Hui, Pan. Twittering by Cuckoo: Decentralized and Socio-aware Online Microblogging Services. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 473–474, New York, NY, USA, 2010. ACM.

[58] Xu, Tianyin and Chen, Yang and Zhao, Jin and Fu, Xiaoming. Cuckoo: Towards Decentralized, Socio-aware Online Microblogging Services and Data Measurements.

In *Proceedings of the 2Nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, HotPlanet '10, pages 4:1–4:6, New York, NY, USA, 2010. ACM.

Appendix A

Projects not considered in scope

There are a variety of projects that provide a user with a decentralised social network or a platform to exchange information between an established relationship. However, some of the projects that call themselves a social network, or are referred to as a social network, have not been thoroughly analysed. This is a deliberate choice, as in the early stages of analysing these projects it seemed that some of the basic features were missing at this point in time or because of some other reasons that will be discussed shortly. These projects were therefore not considered in scope anymore at this point or were not considered a fully decentralised social network. However, some projects might be considered in scope in the future when the projects progress. The list below shows these projects.

- Avatar
- Bitmessage
- Buddycloud
- Buddypress
- duuit
- Elgg
- Ethereum
- Helloworld
- Higgins
- Jappix
- Kopal
- Kune
- Libertree
- Lorea
- Maidsafe
- masques
- Meomni
- Noosefero
- NXTmemo
- ODS
- OpenAutonomy
- Phoenix
- Pixelpark
- Pond
- Secushare
- Sone
- Tent.io
- Themineproject
- Tonika
- Trsst

The projects mentioned above are considered out of scope for this project for a variety of reasons at this moment. The reasons that the projects are considered out of scope at this point are shown in the next paragraphs.

Can not be used in a production environment
This is by far the top reason why some of the aforementioned projects are considered out of scope. Projects that are still in an alpha stage of a software cycle are not considered appropriate to be run in a production environment. Beta versions were considered out of scope only if there would not be a release candidate or stable release in the near future. Software that was still in a beta stage but provided a lot of features that are necessary in a social network, or had a novel concept were considered in scope.

Not broadly accessible
A social network needs to be accessible by a variety of devices. According to Facebook, there were 1.25 billion mobile active Facebook users as of March 2015 [28]. It is therefore important that a social network can be accessed by mobile devices by means of a specialised application or a mobile friendly web interface. Some of the aforementioned projects were peer-to-peer (P2P) decentralised social networks that could not be used by mobile devices. Even if there is a specialised application for most mobile devices, the mobile market is still under development and if there is not a web interface available these new devices are left out of the P2P social network. These P2P solutions were not considered a solution and were therefore considered out of scope for this project.

Abandoned projects
Projects that were not maintained any more by a lead developer or a community were not considered an option. As social networks are still growing and user demands might change over time, it is necessary to be agile and be able to implement new features. An abandoned project might also give an indication that the project was not a success and was not adopted by a lot of people. However, if an abandoned project provided novel concepts we were still open for the idea of analysing such a project. However, this has not been the case.

Other philosophy
Some of the projects were referred to as a decentralised social network but had a different philosophy than that of current social networks. Although this is not a real problem if it provides the basic feature set of a social network, it would be a problem if this was not the case. It would be hard to convince the community or the developers to implement features that have no relationship with the philosophy of such a project. However, it is of course possible to implement these features if the license allows it, but it would be convenient to have these changes at least merged upstream. If the philosophy collides with these new features it could be the case that these changes are not merged upstream, which would mean that a fork needs to be created of that specific project. This would also mean that this fork has to be maintained and new features implemented upstream might not be compatible with the fork. As the research is mostly focused on existing projects that could be provided as a service by hosting providers these projects were considered out of scope.

Missing cross-server message exchange
In order to be a decentralised social network it must be possible to exchange messages between servers on behalf of its users. If this is not the case, the design only allows to have isolated islands where the users of such an island can only communicate with other users on the same island. This is not a decentralised social network in its purest sense and would cause a lock-in of an island, which would limit the adoption by users. Users would need to register user accounts on multiple islands to be able to communicate with all of

its friends or one island would become the biggest island where all users are registered. To be able to create a truly decentralised social network, where users are in the possession of their own data and can choose which node to use based on its reputation and agreements, communication between servers should be possible. This translates back in the use of protocols that are able to communicate between servers and have a lookup procedure to discover the location of a user.